Workload Modeling

for Computer Systems Performance Evaluation

Dror G. Feitelson

The Rachel and Selim Benin School
of Computer Science and Engineering
The Hebrew University of Jerusalem, Israel

Version 1.0.4, typeset on June 10, 2023

To T. P.,
Who taught me that one needs to be aware of one's limitations
in order to make the load workable

Contents

Preface

In 1994 I wrote a long survey about parallel job scheduling [229]. This work described and classified the scheduling schemes of 76 systems, as well as many others that were proposed but never implemented, backed by 638 references. In retrospect, one of the things that struck me was that practically any paper that proposed a new scheme also proved it to be better than competing schemes. Upon reflection, my conclusion was that the source of the problem was in different assumptions and mindsets, including about the properties of the workloads that would run on these systems. The operational conclusion was that it may be more important to understand the workloads than to design new scheduling schemes.

At about the same time, in work on parallel I/O, I was exposed to the Charisma I/O traces collected by David Kotz and Nils Nieuwejaar [517]. Among the voluminous data on I/O operations were a few records about the jobs to which they belonged. This led to an interaction with Bill Nitzberg who provided me with data regarding three months of jobs from the NASA Ames iPSC/860 system, and to the publication of the first analysis of such a workload log [244]. Several years later, this log became one of the first to be included in the Parallel Workloads Archive [534]. This archive has been instrumental in facilitating research based on real data rather than on baseless assumptions.

Fast forward to 2014. It is now widely accepted that workload characterization and workload modeling are very important for reliable performance evaluations of computer systems. If the workload is wrong, the results will be wrong too — not in the mathematical sense, but in the sense that they will not apply to the situation at hand. Regrettably, workloads are sometimes (and maybe often) still treated as an afterthought, despite a lot of work that has been done on this topic.

At least part of the problem is that there is a gap between what is studied in basic probability and statistics courses and what needs to be used in workload modeling and performance evaluation. In particular, topics such as heavy-tailed distributions and self-similarity are advanced statistical concepts that are not covered in basic courses. To make matters worse, books and research papers on these topics tend to start at a level of mathematical sophistication that is beyond that achieved in basic probability and statistics courses. This makes much of the relevant material inaccessible to many practitioners who want to use the ideas, but not spend a lot of time studying all the underlying theory.

One goal of this book is to fill this gap. Specifically, I attempt to make definitions

and techniques accessible to practitioners by emphasizing the intuition behind them. Although math is used to avoid misunderstandings and explain derivations, this is typically done at a rather elementary level, forgoing mathematical rigor in the interest of making the material more understandable. The book does assume a basic working knowledge of probability, but beyond that it provides a relatively detailed discussion that does not assume the reader can fill in the details. Moreover, we specifically avoid a full and detailed discussion of all the latest bleeding-edge research on advanced statistics.

A further problem with the workloads used in performance evaluation studies is that they are often based on assumptions rather than measurements. Therefore another goal of this book is to encourage and promote the experimental aspects of computer science. To further this goal, the book emphasizes the use of real data and contains numerous examples based on real datasets. The datasets used are listed in the Appendix and linked from the book's website.

Using real data to illustrate various concepts is also a means to help build an intuition of what definitions mean and how real data behaves — including cases where data tends to misbehave. This is extremely important, because mathematical techniques will provide some sort of results even when they are misapplied. Developing an intuition regarding your data is therefore an important first step in successful evaluations, and knowing how to look at the data, and in particular how to create illuminating statistical graphs, is an important skill.

In developing my ideas about computer workloads and their modeling I was privileged to work with several outstanding students. The ones who contributed the most to this subject were Uri Lublin, Dan Tsafrir, David Talby, Edi Shmueli, Yoav Etsion, and Netanel Zakay.

By far the most mathematically advanced material is contained in Chapter 7 on self-similarity. In writing about this material (and understanding it myself) I received immense help from Benjamin Yakir, both in explaining the mathematical procedures and in bringing to light the insights behind them. Thanks also to Thomas Mikosch for setting me straight in some places. Daniel Nevo did his best to proofread the statistical parts of the text and tried to convince me to make it more rigorous. Naturally errors and misrepresentations remain my responsibility.

Heartfelt thanks are due to all those who have made their workload data available on the Internet for the benefit of the research community. I hope that in the future this will be taken for granted, and much more data will be available for use. The book's website is at http://www.cs.huji.ac.il/%7Efeit/wlmod and includes links to data sources. Updates and errata will be posted there as well.

The book can be used as the basis for a course on workload modeling or as a supplementary text for a course on performance evaluation. However, it is intended for use by practitioners no less than by academics. I wrote it because I found no source from which I myself could learn and understand the more advanced concepts, based on data and intuition rather than formal proofs. I do not know of any other book like it. I hope you find it useful.

Dror Feitelson
Jerusalem, February 2014

1

Introduction

Performance evaluation is a basic element of experimental computer science. It is used to compare design alternatives when building new systems, to tune parameter values of existing systems, and to assess capacity requirements when setting up systems for production use. Lack of adequate performance evaluation can lead to bad decisions, which result either in an inability to accomplish mission objectives or an inefficient use of resources. A good evaluation study, in contrast, can be instrumental in the design and realization of an efficient and useful system.

There are three main factors that affect the performance of a computer system:

1. The system's design
2. The system's implementation
3. The workload to which the system is subjected

The first two factors are typically covered with some depth in vocational training and academic computer science curricula. Courses on data structures and algorithms provide the theoretical background for a solid design, and courses on computer architecture and operating systems provide case studies and examples of successful designs. Courses on performance-oriented programming, on object-oriented design, and programming labs provide the working knowledge required to create and evaluate implementations. But there is typically little or no coverage of performance evaluation methodology in general and of workload modeling in particular.

Regrettably, performance evaluation is similar to many other endeavors in that it follows the GIGO principle: garbage-in-garbage-out. Evaluating a system with the wrong workloads will most probably lead to irrelevant results, which cannot be relied upon. This motivates the quest for the "correct" workload model [717, 256, 654, 19, 733, 103, 235, 636]. It is the goal of this book to help propagate the knowledge and experience that have accumulated in the research community regarding workload modeling, and to make it accessible to practitioners of performance evaluation.

To read more: Although performance evaluation in general and workload modeling in particular are typically not given much consideration in vocational and academic curricula, there has nev-

ertheless been much research activity in this area. Good places to read about this are textbooks on performance evaluation, including the following:

- Jain [367] cites arguments about what workload is most appropriate as the deepest rat-hole that an evaluation project may fall into (page 161). Nevertheless, he does provide several chapters that deal with the characterization and selection of a workload.
- Law and Kelton [427] provide a very detailed presentation of distribution fitting, which is arguably at the core of workload modeling.
- Le Boudec [429] has perhaps the most practical and down-to-earth exposition of performance evaluation, including a discussion of model fitting and heavy-tailed distributions. And it has the advantage of being available for free from http://perfeval.epfl.ch/.
- The book on self-similar network traffic edited by Park and Willinger [537] provides good coverage of heavy tails and self-similarity.

In addition there are numerous research papers, many of which are cited in this book and appear in the bibliography. For an overview, see the survey papers by Calzarossa and co-authors [103, 101]. Another good read is the classic paper by Ferrari [258]. This book has its roots in a tutorial presented at Performance 2002 [234].

1.1 The Importance of Workloads

The study of algorithms involves an analysis of their performance. When we say that one sorting algorithm is $O(n \log n)$, whereas another is $O(n^2)$, we mean that the first is faster and therefore better. But this is typically a worst-case analysis, which may occur, for example, only for a specific ordering of the input array. In fact, different inputs may lead to very different performance results. The same algorithm may terminate in linear time if the input is already sorted to begin with, but may require quadratic time if the input is sorted in the opposite order.

The same phenomena may happen when evaluating complete systems: they may perform well for one workload, but not for another[1]. To demonstrate the importance of workloads we therefore describe three examples in which the workload makes a large difference to the evaluation results.

Example 1: Scheduling Parallel Jobs by Size

A simple model of parallel jobs considers them as rectangles in processors×time space: each job needs a certain number of processors for a certain interval of time. Scheduling is then the packing of these job-rectangles into a larger rectangle that represents the available resources.

It is well known that average response time is reduced by scheduling short jobs first (the SJF algorithm). The problem is that the runtime is typically not known in advance.

[1] Incidentally, this is true for all types of systems, not only for computer systems. A female computer scientist once told me that an important side benefit of her chosen career is that she typically does not have to wait in line for the ladies room during breaks in male-dominated computer science conferences. But this benefit was lost when she attended a conference dedicated to encouraging female students to pursue a career in computer science...

But in parallel systems, scheduling according to job *size* may unintentionally also lead to scheduling by *duration*, if there is some statistical correlation between these two job attributes.

As it turns out, the question of whether such a correlation exists is not easy to settle. Three application scaling models have been proposed in the literature [742, 630]:

- *Fixed work*. This assumes that the work done by a job is fixed, and parallelism is used to solve the same problems faster. Therefore the runtime is assumed to be inversely proportional to the degree of parallelism (negative correlation). This model is the basis for Amdahl's law.
- *Fixed time*. Here it is assumed that parallelism is used to solve increasingly larger problems, under the constraint that the total runtime stays fixed. In this case, the runtime distribution is independent of the degree of parallelism (no correlation).
- *Memory bound*. If the problem size is increased to fill the available memory associated with a larger number of processors, the amount of productive work typically grows at least linearly with the parallelism. The overheads associated with parallelism always grow superlinearly. Thus the total execution time actually increases with added parallelism (a positive correlation).

Evaluating job scheduling schemes with workloads that conform to the different models leads to drastically different results. Consider a workload that is composed of jobs that use power-of-two processors. In this case a reasonable scheduling algorithm is to cycle through the different sizes, because the jobs of each size pack well together [419]. This works well for negatively correlated and even uncorrelated workloads, but is bad for positively correlated workloads [419, 450]. The reason is that under a positive correlation the largest jobs dominate the machine for a long time, blocking out all others. As a result, the average response time of all other jobs grows considerably.

But which model actually reflects reality? Evaluation results depend on the selected model of scaling; without knowing which model is more realistic, we cannot use the performance evaluation results. As it turns out, the constant time or memory-bound models are more realistic than the constant work model. Therefore scheduling parallel jobs by size with a preference for large jobs is at odds with the desire to schedule short jobs first, and can be expected to lead to high average response times.

Example 2: Processor Allocation Using a Buddy System

Gang scheduling is a method for scheduling parallel jobs using time slicing, with coordinated context switching on all the processors. In other words, first the processes of one job are scheduled on all the processors, and then they are all switched simultaneously with the processes of another job. The data structure used to describe this is an Ousterhout matrix [529], in which columns represent processors and rows represent time slots.

An important question is how to pack jobs into rows of the matrix. One example is provided by the DHC scheme [245], in which a buddy system is used for processor

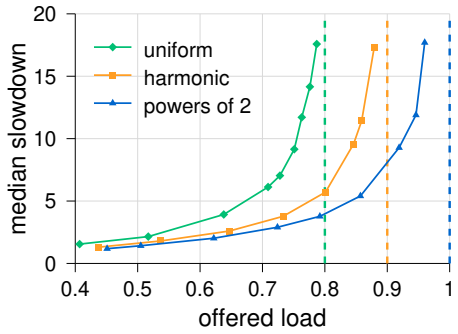


Figure 1.1: *Simulation results showing normalized response time (slowdown) as a function of load for processor allocation to parallel jobs using DHC, from [245]. The three curves are for exactly the same system — the only difference is in the distribution of job sizes. The dashed lines are proven bounds on the achievable utilization for these three workloads.*

allocation: each request is extended to the next power of two, and allocations are always done in power-of-two blocks of processors. This scheme leads to using the same blocks of processors in different slots, which is desirable because it enables a job to run in more than one slot if its processors happen to be free in another slot.

The quality of the produced packing obviously depends on the distribution of job sizes. The DHC scheme has been evaluated with three different distributions: a uniform distribution in which all sizes are equally likely, a harmonic distribution in which the probability of size s is proportional to $1/s$, and a uniform distribution on powers of two. Both analysis and simulations showed significant differences between the utilizations that could be achieved for the three distributions (Figure 1.1) [245]. These differences corresponds to different degrees of fragmentation that are inherent to packing jobs that come from these distributions. For example, with a uniform distribution, rounding each request size up to the next power of two leads to a 25% loss to fragmentation — the average between no loss (if the request is an exact power of two) and a nearly 50% loss (if the request is just above a power of two, and we round up to the next one). The DHC scheme recovers part of this lost space, so there is actually only 20% loss, as shown in Figure 1.1.

Note that this analysis tells us what to expect in terms of performance, provided we know the distribution of job sizes. But what is a typical distribution encountered in real systems in production use? Without such knowledge, the evaluation cannot provide a definitive answer. As it turns out, empirical distributions have many small jobs (similar to the harmonic distribution) and many jobs that are powers of two. Thus using a buddy system is indeed effective for real workloads, but it would not be if workloads were more uniform with respect to job size.

Example 3: Load Balancing on a Unix Cluster

A process running on an overloaded machine will receive worse service than a process running on an unloaded machine. Load balancing is the activity of migrating a running process from an overloaded machine to an underloaded one. When loads are balanced, processes receive equitable levels of service.

One problem with load balancing is choosing which process to migrate. Migration involves considerable overhead. If the process terminates soon after being migrated, that overhead has been wasted. In addition, the process cannot run during the time it is being migrated. Again, if it terminates soon after the migration, it would have been better off staying in its place.

Thus it would be most beneficial if we could identify processes that may be expected to continue to run for a long time, and select them for migration. But how can we know in advance whether a process is going to terminate soon or not? The answer is that it depends on the statistics of process runtimes.

It is well known that the exponential distribution is memoryless. Therefore if we assume that process runtimes are exponentially distributed, we cannot use the time that a process has run so far to learn how much longer it is expected to run: this expectation is always equal to the mean of the distribution. In mathematical terms the probability that the runtime T of a process will grow by an additional τ, given that it has already run for time t, is equal to the probability that it will run for more than τ in the first place:

$$\Pr(T > t + \tau \mid T > t) = \Pr(T > \tau)$$

But the runtimes of real processes on Unix systems, at least long-lived processes, are not exponentially distributed. In fact, they are heavy-tailed [435, 320]. Specifically, the probability that a process run for more than τ time has been found to decay polynomially rather than exponentially:

$$\Pr(T > \tau) \propto \tau^{-\alpha} \qquad \alpha \approx 1$$

This means that most processes are short, but a small number are very long. If we condition the probability that a process will run for additional time on how much it has already run, we find that a process that has already run for t time may be expected to run for an additional t time: the expectation actually grows with how long the process has already run! (The derivation is given in Section 5.2.1.) This makes the long-lived processes easy to identify: they are the ones that have run the longest so far. And selecting processes for migration based on runtime will be much better than selecting at random, because a random process will most likely be very short. But note that selection based on runtime depends on a detailed characterization of the workload, which in fact is valid only for the specific workload that is indeed observed empirically.

Sensitivity to Workloads

The above three examples are, of course, not unique. There are many examples in which workload features have a significant effect on performance. Importantly, not every workload feature has the same effect: in some cases it is one specific workload feature that is

the most important. The problem is that it is not always obvious in advance which feature is the most important, and even if it *seems* obvious, we might be wrong [237, 439, 248]. This motivates the practice of conservative workload modeling, where an attempt is made to correctly model all known workload features, regardless of their perceived importance [248]. Alternatively, it motivates the use of real workloads to drive evaluations, because real workloads may contain features that we do not know about and therefore cannot model.

1.2 Types of Workloads

Workloads appear in many contexts and therefore have many different types.

1.2.1 Workloads in Different Domains

The previous three examples are all from the field of scheduling by an operating system, where the workload items are jobs that are submitted by users. This type of workload is characterized by many attributes. If only the scheduling of the CPU is of interest, the relevant attributes are each job's arrival and running times. If memory usage is also being investigated, the total memory usage and locality of reference also come into play, because memory pressure can have an important effect on scheduling and lead to swapping. I/O can also have a great effect on scheduling. Modeling it involves the distribution of I/O sizes and how they interleave with the computation. For parallel jobs, the number of processors used is an additional parameter, which influences how well jobs pack together.

The level of detail needed in workload characterization depends on the goal of the evaluation. For example, in the context of operating system scheduling, it is enough to consider a process as "computing" for a certain time. But when studying CPU architectures and instruction sets, a much more detailed characterization is required. The instruction mix is important in determining the effect of adding more functional units of different types. Dependencies among instructions determine the benefits of pipelining, branch prediction, and out-of-order execution. Loop sizes determine the effectiveness of instruction caching. When evaluating the performance of a complete CPU, all these details have to be correct. Importantly, many of these attributes are input dependent, so representative workloads must include not only representative applications but also representative inputs [204].

I/O provides another example of workloads that can be quite complex. Attributes include the distribution of I/O sizes, the patterns of file access, and the use of read vs. write operations. It is interesting to note the duality between modeling I/O and computation, depending on the point of view. When modeling processes for scheduling, I/O is typically modeled as just taking some time between CPU bursts (if it is modeled at all). When modeling I/O, computation is modeled as just taking some time between consecutive I/O operations. Of course, it is possible to construct a fully detailed joint model, but the number of possible parameter combinations may grow too much for this to be practical.

Application-level workloads are also of interest for performance evaluation. A prime example is the sequence of transactions handled by a database. Databases account for a large part of the usage of large-scale computer systems such as mainframes, and are critical for many enterprise-level operations. Ensuring that systems meet desired performance goals without excessive (and expensive) over-provisioning is therefore of great importance. Again, reliable workload models are needed. For example, the transactions fielded by the database of a large bank can be expected to be quite different from those fielded by a database that provides data to a dynamic web server. The differences may lie in the behavior of the transactions (e.g. how many locks they hold and for how long, how many records they read and modify) and in the structure of the database itself (the number of tables, their sizes, and relationships among them).

The workload on web servers has provided a fertile ground for research, as has network traffic in general. Of major interest is the arrival process of packets. Research in the early 1990s showed that packet arrivals are correlated over long periods, as manifested by burstiness at many different time scales. This finding was in stark contrast with the Poisson model that was routinely assumed until that time. The new models based on this finding led to different performance evaluation results, especially with respect to queueing and packet loss under high loads.

The world wide web is especially interesting in terms of workload modeling because the workloads seen at the two ends of a connection are quite different. First, there is the many-to-many mapping of clients to servers. A given client only interacts with a limited number of servers, whereas the population as a whole may interact with many more servers and display rather different statistics. Servers, in contrast, typically interact with many clients at a time, so the statistics they see are closer to the population statistics than to the statistics of a single client. In addition, caching by proxies modifies the stream of requests en route [271, 266]. The stream between a client and a cache has more repetitions than the stream from the cache to the server. Uncacheable objects may also be expected to be more prevalent between the cache and the server than between the clients and the cache, because in the latter case they are intermixed with more cacheable objects.

1.2.2 Dynamic vs. Static Workloads

An important difference between workload types is their rate of events. A desktop machine used by a single user may process several hundreds of commands per day. This may correspond to thousands or millions of I/O operations, and to many billions of CPU instructions and memory accesses. A large-scale parallel supercomputer may serve only several hundred jobs a day from all users combined. A router on the Internet may handle billions of packets in the same time frame.

Note that in this discussion we talk of *rates* rather than *sizes*. A size implies that something is absolute and finite. A rate is a size per unit of time, and implies continuity. This is related to the distinction between static and dynamic workloads. A *static workload* is one in which a certain amount of work is given, and when it is done that is it. A

dynamic workload, in contrast, is one in which work continues to arrive all the time; it is never "done".

The differences between static and dynamic workloads may have the following subtle implications for performance evaluation.

- A dynamic workload requires the performance evaluator to create a changing mix of workload items (e.g., jobs). At the very minimum, doing so requires an identification of all possible jobs, and data regarding the popularity of each one, which may not be available. With static workloads you can use several combinations of a small set of given applications. For example, given applications A, B, and C, you can run three copies of each in isolation, or a combination of one job from each type. This is much simpler, but most probably further from being realistic. Benchmarks (discussed in the next section) are often static.

- A major difficulty with dynamic workloads is that they include an arrival process, which has to be characterized and analyzed in addition to the workload items themselves. A static workload does not impose this additional burden. Instead, it is assumed that all the work arrives at once at the outset.

- Distributions describing static or dynamic workloads may differ. For example, a snapshot of a running system may be quite different from a sample of the input distribution, due to correlations of workload attributes with residence time. Thus if the input includes many short jobs and few long jobs, sampling from a trace of all the jobs that were executed on the system (effectively sampling from the input distribution) will display a significant advantage for short jobs. But observing a snapshot of the live system may indicate that long jobs are more common, simply because they stay in the system longer, and therefore have a higher probability of being seen in a random sample.

- Perhaps the most important difference occurs because performance often depends on the system's state. A static workload being processed by a "clean" system may then be very different from the same set of jobs being processed by a system that had previously processed many other jobs in a dynamic manner. The reason is system aging, e.g., the fragmentation of resources [639].

- Aging is especially important when working on age-related failures (e.g., those due to memory leaks). Static workloads are incapable of supporting work on topics such as software rejuvenation [692]. A related example is the study of thrashing in paging systems [173]. Such effects cannot be seen when studying the page replacement of a single application with a fixed allocation. Rather, they only occur due to the dynamic interaction of multiple competing applications.

Several of these considerations indicate that static workloads cannot be considered as valid samples of real dynamic workloads. This book focuses on dynamic workloads.

1.2.3 Benchmarks

One of the important uses of performance evaluation is to compare different systems to each other, typically when trying to decide which one to buy. However, such comparisons are meaningful only if the systems are evaluated under equivalent conditions, and, in particular, with the same workload. This motivates the canonization of a select set of workloads that are then ported to different systems and used as the basis for comparison. Such standardized workloads are called *benchmarks*.

Benchmarks have a huge impact on the computer industry, because they are often used in marketing campaigns [723]. Moreover, they are also used in performance evaluation during the design of new systems and even in academic research, so their properties (and deficiencies) may shape the direction of new developments. It is thus of crucial importance that benchmarks be representative of real needs. To ensure the combination of representativeness and industry consensus, several independent benchmarking organizations have been created. Two of the best known are SPEC and TPC.

SPEC is the Systems Performance Evaluation Consortium [655]. This organization defines several benchmark suites aimed at evaluating computer systems. The most important of these suites is SPEC CPU, which dominates the field of evaluating the microarchitecture of computer processors. This benchmark comprises a set of applications, divided into two groups: one emphasizing integer and symbolic processing, and the other emphasizing floating point scientific processing. To ensure that the benchmark is representative of current needs, the set of applications is replaced every few years. New applications are selected from those submitted in an open competition.

TPC is the Transaction Processing Performance Council. This organization defines several benchmarks for evaluating database systems. Perhaps the most commonly used is TPC-C, which is used to evaluate online transaction processing. The benchmark simulates an environment in which sales clerks execute transactions at a warehouse. The simulation has to comply with realistic assumptions regarding how quickly human users can enter information.

The SPEC CPU suite and TPC-C are essentially complete, real applications. Other benchmarks are composed of kernels or of synthetic applications. Both of these approaches reduce realism in the interest of economy or focus on specific aspects of the system. *Kernels* are small parts of applications in which most of the processing occurs (e.g., the inner loops of scientific applications). Their measurement focuses on the performance of the processor in the most intensive part of the computation. *Synthetic applications* mimic the behavior of real applications without actually computing anything useful. Using them enables the measurement of distinct parts of the system in isolation or in carefully regulated mixtures; this is often facilitated by parameterizing the synthetic application, with parameter values governing various aspects of the program's behavior (e.g., the ratio of computation to I/O) [93]. Taking this to the extreme, *microbenchmarks* are small synthetic programs designed to measure a single system feature, such as memory bandwidth or the overhead to access a file. In this case there is no pretense of being representative of real workloads.

Benchmarking is often a contentious affair. Much argument and discussion are spent

on methodological issues, as vendors contend to promote features that will show their systems in a favorable light. There is also the problem of the benchmark becoming an end in itself, when vendors optimize their systems to cater to the benchmark, at the possible expense of other application types. It is therefore especially important to define what the benchmark is supposed to capture. There are two main options: to span the spectrum of possible workloads, or to be representative of real workloads.

Covering the space of possible workloads is useful for basic scientific insights and when confronted with completely new systems. In designing such benchmarks, workload attributes have to be identified and quantified, and then combinations of realistic values are used. The goal is to choose attributes and values that cover all important options, but without undue redundancy [193, 388]. The problem with this approach is that, by definition, it only measures what the benchmark designers dreamed up in advance. In other words, there is no guarantee that the benchmarks indeed cover all possibilities.

The other approach requires that benchmarks reflect real usage, and be representative of real workloads. To ensure that this is the case, workloads have to be analyzed and modeled. This should be done with considerable care and an eye to detail. For example, when designing a benchmark for CPUs, it is not enough to consider the instruction mix and their interdependencies — it is also necessary to consider the interaction of the benchmark with the memory hierarchy and its working set size.

Although benchmarks are not discussed in detail in this book, the methodologies covered are expected to be useful as background for the definition of benchmarks.

To read more: Several surveys on benchmarks were written by Weicker [724, 723]. One of the reasons they are interesting is that they show how benchmarks change over time.

1.3 Workload Modeling

Workload modeling is the attempt to create a simple and general model, which can then be used to generate synthetic workloads at will, possibly with slight (but well-controlled!) modifications. The goal is typically to be able to create workloads that can be used in performance evaluation studies, and the synthetic workload is supposed to be similar to those that occur in practice on real systems. This is a generalization of the concept of benchmarks, which is applicable when the consensus regarding the precise workload is less important.

1.3.1 What It Is

Workload modeling always starts with measured data about the workload. This data is often recorded as a trace, or log, of workload-related events that happened in a certain system. For example, a job log may include data about the arrival times of jobs, who ran them, and how many resources they required. Basing evaluations on such observations, rather than on baseless assumptions, is a basic principle of the scientific method.

The suggestion that workload modeling should be based on measurements has been made at least since the 1970s [256, 654, 19]. However, for a long time relatively few

models based on actual measurements were published. As a result, many performance studies did not use experimental workload models at all (and do not to this day). The current wave of using measurements to create detailed and sophisticated models started in the 1990s. It was based on two observations: one, that real workloads tend to differ from those often assumed in mathematical analyses, and two, that this makes a difference.

There are two common ways to use a measured workload to analyze or evaluate a system design [127, 256, 612]:

1. Use the traced workload directly to drive a simulation.
2. Create a model from the trace and use the model for either analysis or simulation.

For example, trace-driven simulations based on large address traces are often used to evaluate cache designs [635, 410, 401, 705]. But models of how applications traverse their address space have also been proposed, and provide interesting insights into program behavior [683, 684].

The essence of modeling, as opposed to just observing and recording, is one of abstraction. This means two things: generalization and simplification.

Measurements are inherently limited. Collecting data may be inconvenient or costly, and instrumentation may introduce overhead. The conditions under which data is collected may lead to a rather small sample of the space of interest. For example, given a server with 128 nodes, it is not possible to collect data about systems with 64 or 256 nodes. We need models to transcend these limitations.

But at the same time, we also want the models to be simpler than the recorded workload. A log with data about tens of thousands of jobs may contain hundreds of thousands of numbers. It is ludicrous to claim that the exact values of all these numbers are important. What is important are the underlying patterns. It is these patterns that we seek to uncover and articulate, typically in the form of statistical distributions. And we want the model to be parsimonious, meaning that it uses as few parameters as possible.

To read more: For an exposition on mathematical modeling, see the essay by McLaughlin [480]. The book on data analysis by Berthold et al. [71] combines motivation with modeling methodology.

1.3.2 Why Do It?

The main use of workload models is for performance evaluation. But there are other uses as well, e.g. to characterize normal operation conditions, with the goal of being able to recognize abnormal conditions.

Modeling for Performance Evaluation

As noted above, traced data about a workload can be used directly in an evaluation, or else it can be used as the basis for a model. The advantage of using a trace directly is that it is the most "real" test of the system: the workload reflects a real workload precisely, with all its complexities, even if they are not known to the person performing the analysis.

The drawback is that the trace reflects only a specific workload, and there is always the question of whether the results generalize to other systems or load conditions. In particular, there are cases where the workload depends on the system configuration, and therefore a given workload is not necessarily representative of workloads on systems with other configurations. Obviously, this makes the comparison of different configurations problematic. In addition, traces are often misleading if we have incomplete information about the circumstances in which they were collected. For example, workload traces may contain intervals when the machine was down or part of it was dedicated to a specific project, but this information is often not be available.

Workload models have a number of advantages over traces [190, 234, 629]:

Adjustment — Using a model, it is possible to adjust the workload to fit a certain situation. For example, consider a situation in which we have data from a 128-node machine, and we want to evaluate a 256-node machine. Using the data directly will result in a workload where the maximal job size is only half the system size. But with a model that accepts the system size as a parameter, we can create a new workload that matches the larger size.

Controlled modification — It is possible to modify the values of model parameters one at a time, in order to investigate the influence of each one, while keeping other parameters constant. This allows for direct measurement of system sensitivity to the different parameters. It is also possible to select model parameters that are expected to match the specific workload at a given site.

In general it is not possible to manipulate traces in this way, and even when it is possible, doing so can be problematic. For example, it is common practice to increase the modeled load on a system by reducing the average interarrival time. But this practice has the undesirable consequence of shrinking the daily load cycle as well. With a workload model, we can control the load independent of the daily cycle.

Repetitions — Using a model, it is possible to repeat experiments under statistically similar conditions that are nevertheless not identical. For example, a simulation can be run several times with workloads generated using different seeds for the random number generator. This is needed in order to compute confidence intervals. With a log, you only have a single data point.

Stationarity — In a related vein, models are better for computing confidence intervals because the workload is stationary by definition (that is, it does not change with time). Real workloads, in contrast, tend to be nonstationary: the workload parameters fluctuate with time, as usage patterns change (Figure 1.2). This raises serious questions regarding the common methodology of calculating confidence intervals based on the standard deviation of performance metrics.

Generalization — Models provide a generalization and avoid overfitting to a specific dataset. For example, Figure 1.3 shows the variability in the arrival process at different large-scale parallel supercomputers. Each plot gives the number of jobs

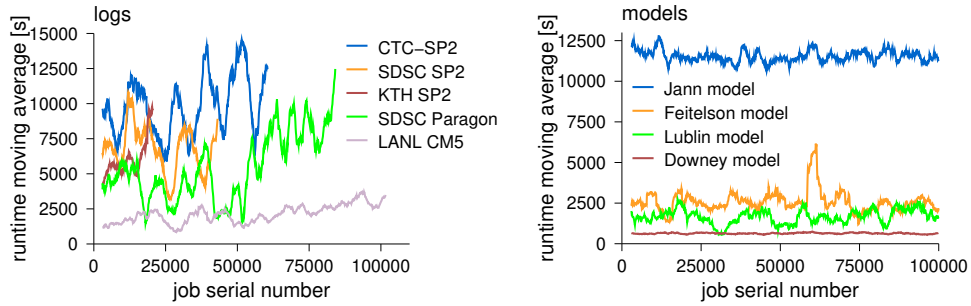


Figure 1.2: *Real workloads tend to be nonstationary, as opposed to workload models, as demonstrated by these graphs of the moving average of job runtimes in different logs and models of parallel jobs. A window size of 3000 jobs was used to calculate the average.*

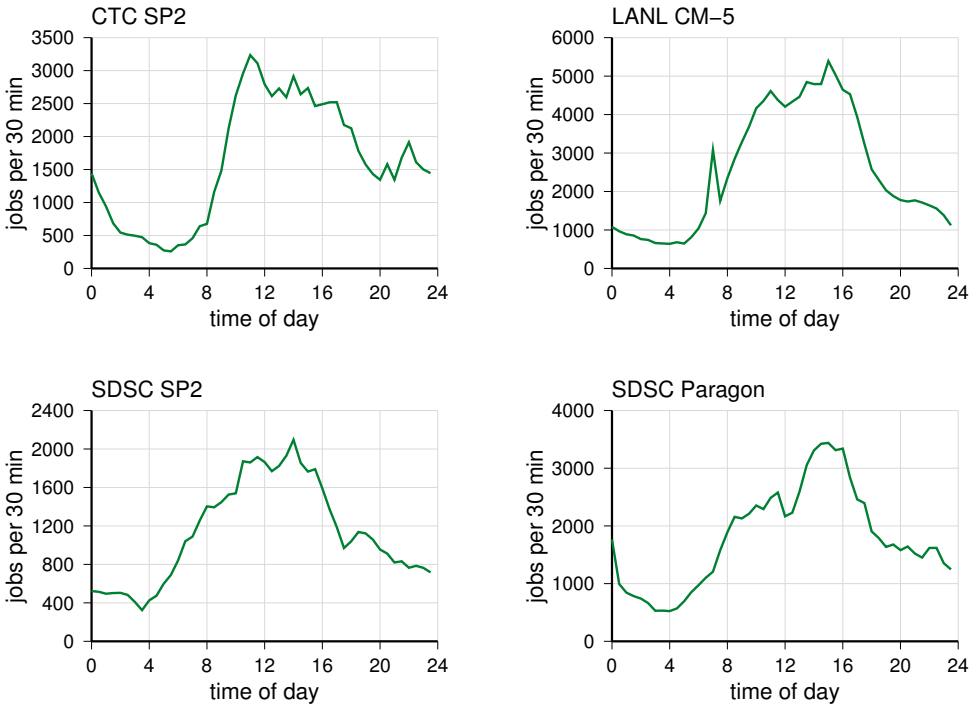


Figure 1.3: *Arrival patterns at four large-scale parallel machines exhibit a similar structure but different details.*

that arrived at different times of the day, averaged over the duration of the available trace (typically one or two years). Although all the graphs show a similar daily cycle with low activity at night and more activity during the day, the details are different. For example, the sharp rise in load in the CTC SP2 data is very different from the incremental growth in the SDSC Paragon. Thus using any of these

patterns as is risks overfitting and discriminating against some other pattern. This danger is reduced by using a model that generalizes all the observed patterns.

Avoiding noise — Logs may not represent the real workload due to various problems. For example, an administrative policy that places a limit of four hours may force users to break long jobs into multiple short jobs. Another example is that jobs killed by the system may be resubmitted. Often we don't know about such problems, so taking the data at face value may be misleading. Conversely, a modeler has full knowledge of model workload characteristics. For example, it is easy to know which workload parameters are correlated with each other because this information is part of the model.

In general, we may observe that when using a log the default is to include various abnormal situations that occurred when the log was recorded. These do not reflect normal usage, but we don't know about them and therefore cannot remove them. When using a model the situation is reversed: abnormal conditions will only occur if they are inserted explicitly as part of the model [249].

Understanding — Modeling also increases our understanding and can lead to new designs based on this understanding. For example, identifying the repetitive nature of job submittals can be used to learn about job requirements from the history. One can design a resource management policy that is parameterized by a workload model, and then use measured values for the local workload to tune the policy. As another example, knowing that certain workload parameters are heavy-tailed leads to resource management algorithms that exploit the big differences among jobs [154].

Added features — Models can also include features that cannot be included in logs. For example, a model can include an explicit feedback loop in which system performance affects subsequent load generation [614]. A log only contains a list of load items, and we don't know if and how they may have depended on whatever happened before.

Efficiency — Using models can be more efficient than using a trace. For starters, the description of a model is typically much more concise, because it only includes some model parameters; a trace, in contrast, may include thousands or even millions of individual numbers. And due to the stationarity and uniformity of a model, it may be possible to run much shorter simulations and still get convergent results [200, 198].

Privacy — Finally, real workload traces have the drawback that they may disclose proprietary information. As a result enterprises that operate large-scale systems are often loath to release workload data. But a model may gloss over the sensitive parts and still provide useful information [195, 564].

Modeling for Capacity Planning

A special case of performance evaluations that deserves individual attention is capacity planning. In a sense, capacity planning is performance evaluation in reverse: instead of deriving the performance of a given system configuration, we seek the configuration that will provide the desired performance [485]. This is central to provisioning resources so as to fulfill service-level agreements. Thus it is especially important for cloud infrastructures [275].

The required system capacity obviously depends on the workload intensity: we need more capacity to do more work. But the relationship is often not linear. Moreover, intensity is not the only factor that affects performance, and thus the required capacity.

Perhaps the most important workload attribute in terms of capacity planning is burstiness [110, 109]. Burstiness means that there are large fluctuations in the workload intensity, and therefore the average workload intensity is not representative of what we will observe in practice at different times. This burstiness can take either of two forms (or a combination of both): the arrival rate of new work can change, or the amount of work required to service each job can change. In order to provide adequate performance in worst-case load situations, significant over-provisioning may be necessary. Thus a good characterization and understanding of burstiness in the workload are crucial.

Modeling for Online Control

Another reason for modeling workloads is as part of predicting what will happen in the near future. Such predictions then allow for an optimization of the system's performance by the insightful allocation of resources.

In the simplest version, measuring the current workload online allows systems to adjust to their workload. This can take the form of tuning system parameters [243, 757], or switching among alternative algorithms [669]. In a sense, the system is learning its workload and taking steps to best serve it.

A more sophisticated approach is to actually include a model of the system behavior in the tuning procedure. To use this model, the workload is measured online. Combining the workload model with the system model allows for an analysis of the system's performance, including the effect of various changes to the system configuration. The results of the analysis are then used for automatic online control, thus optimizing the performance in accordance with the characteristics of the current workload, and with the ways in which they interact with the system design [756].

A special case of online control is providing the required quality of service so as to satisfy service-level agreements. For example, Internet service providers need to be able to estimate the available network bandwidth before they can commit to carrying additional traffic. This estimation can be based on models that use measured current conditions as an input, as is done in the network weather service [739].

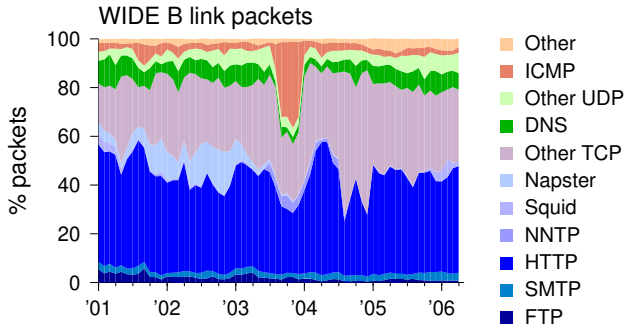


Figure 1.4: *The Internet exhibited a surge of ICMP traffic starting on 18 August 2003, when the Welchia worm started to spread.*

Modeling for Anomaly Detection

A fourth reason for workload modeling is the desire to characterize the "normal" usage patterns of a system. This information is useful for the identification of abnormal patterns, such as those that occur in the following situations:

- When a system is attacked its workload may be significantly different from the normal workload [96, 665, 709, 117]. For example, this happens with distributed denial-of-service (DDoS) attacks, and may also happen when it is infected by a computer virus.
- When a system is misconfigured or a software upgrade includes a new bug, its behavior may change significantly [58, 128]. Such situations may also lead to misconceptions regarding system capacity requirements and ultimately to excessive equipment procurement.

Identifying such situations is a necessary first step for dealing with them.

An example is shown in Figure 1.4. This displays monthly averages of the relative fraction of packets using different protocols in the Internet, as observed on access point B of the WIDE backbone (which is a transpacific link). Starting on 18 August 2003, there was a huge surge of ICMP packets, which resulted from the spread of the Welchia worm. This continued until the worm shut itself down on 1 January 2004. A preponderance of ICMP packets could thus serve as an indicator for infection by the worm.

The main problem with using statistical modeling for the detection of attacks is that it relies on the assumption that the characteristics of the attack lie beyond the normal variability of the workload (as was obviously the case for the Welchia worm). In many cases, however, the normal variability may be very big. Legitimate high activity may therefore be erroneously flagged as an attack.

A possible solution to this problem is to use more sophisticated multifaceted workload models. For example, identifying the spread of a virus solely based on the level of activity is risky, because high traffic may occur for legitimate reasons, and the normal variability between prime time and non-prime time is very big. An alternative is to

devise several independent tests for abnormality and use them together. In the context of virus spreading by email, these tests can include user cliques (the sets of users that typically communicate with each other), the relative popularity of different users (not all users receive the same amount of email), and the rates at which emails are sent (again different users have different profiles) [665]. If several of these change, virus activity may be suspected.

An even more sophisticated solution is to use machine learning techniques to learn to identify normal behavior based on training data; observed behavior that does not match what we have learned is then flagged as a possible attack [457, 709, 117]. Particular attention can be paid to patterns that are rare but still legitimate, so as to reduce false positives. This approach has the advantage of being able to flag suspicious behaviors even if they had not been seen before, thus alleviating one of the main drawbacks of systems that rely on identifying signatures of known attacks.

1.3.3 How It Is Done

Workload modeling is based on data analysis. While the models are typically statistical in nature [480, 427], it is in general impossible to extract all the required data by purely statistical means. Thus using graphical methods to observe and analyze the data is of great importance [31, 699, 700, 701, 702, 71]. Using common sense is also highly recommended [120].

Descriptive Modeling

Workload models fall into two classes: descriptive models and generative models. The difference is that *descriptive models* just try to mimic the phenomena observed in the workload, whereas *generative models* try to emulate the process that generated the workload in the first place.

The most common approach used in descriptive modeling is to create a statistical summary of an observed workload. This summarization is applied to all the workload attributes, e.g. computation, memory usage, I/O behavior, communication, etc. [654, 412]. It is typically assumed that the longer the observation period, the better. Thus we can summarize an entire year's workload by analyzing a record of all the jobs that ran on a given system during this year, and fitting distributions to the observed values of the different parameters (the topic of Chapter 4). A synthetic workload can then be generated according to the model, by sampling from the distributions that constitute the model (Figure 1.5). The model can also be used directly to parameterize a mathematical analysis.

Within the class of descriptive models, one finds different levels of abstraction on the one hand, and different levels of faithfulness to the original data on the other hand. The most strictly faithful models try to mimic the data directly (e.g., by using a distribution that has exactly the same shape as the empirical distribution). An example is the re-creation of a distribution's tail by using a detailed hyper-exponential model, as described in Section 4.4.3. The other alternative is to use the simplest abstract mathematical model

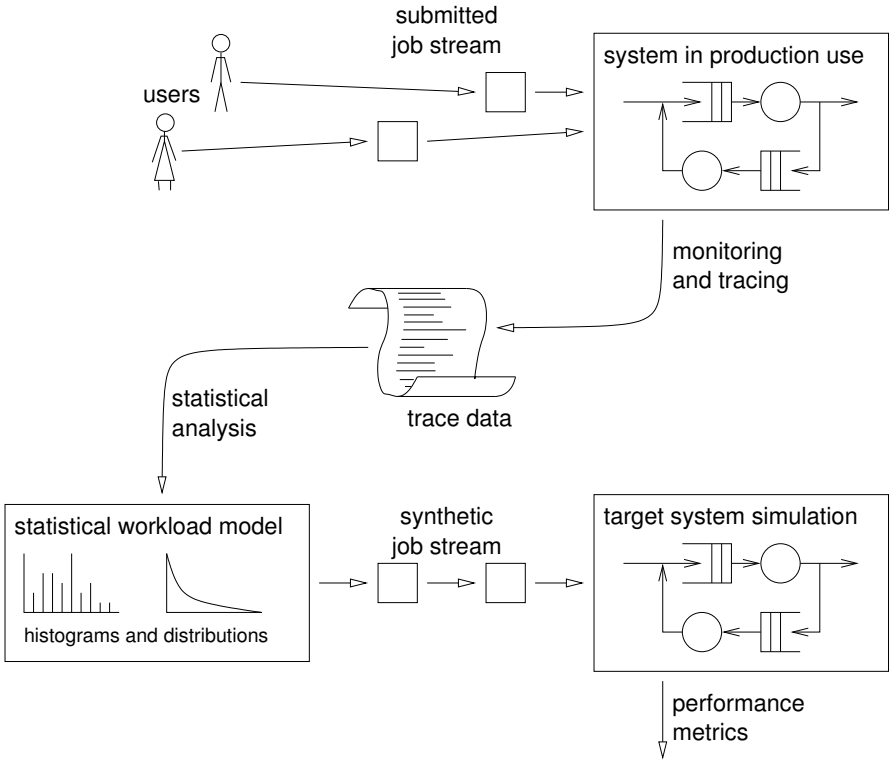


Figure 1.5: *Workload modeling based on a statistical summary of the workload that occurred on a real system (top), and its use in performance evaluation (bottom).*

that has certain properties that are considered to be the most important. For example, instead of describing a distribution by its shape, we can decide to focus on its moments (this is explained in more detail in Section 4.2.2). We can then select a distribution that matches say the first three moments, disregarding the question of whether or not it fits the shape of the original distribution.

Contrary to what might be thought, the question of preferring abstraction or strict faithfulness is more practical than philosophical. The main consideration in workload modeling is not necessarily striving for the "truth". Rather, it is the desire to capture a certain workload feature in a way that is good enough for a specific evaluation. However, if the effect of different workload features is not known, it is safer to try and mimic the observed workload as closely as possible [248].

To read more: In statistics the process leading to descriptive modeling is often called exploratory data analysis. The classic in this area is Tukey's book [702]. A modern exposition including pointers to software implementations is given by Berthold et al. [71]. Regrettably, these do not cover data with highly skewed distributions as is common in computer workloads.

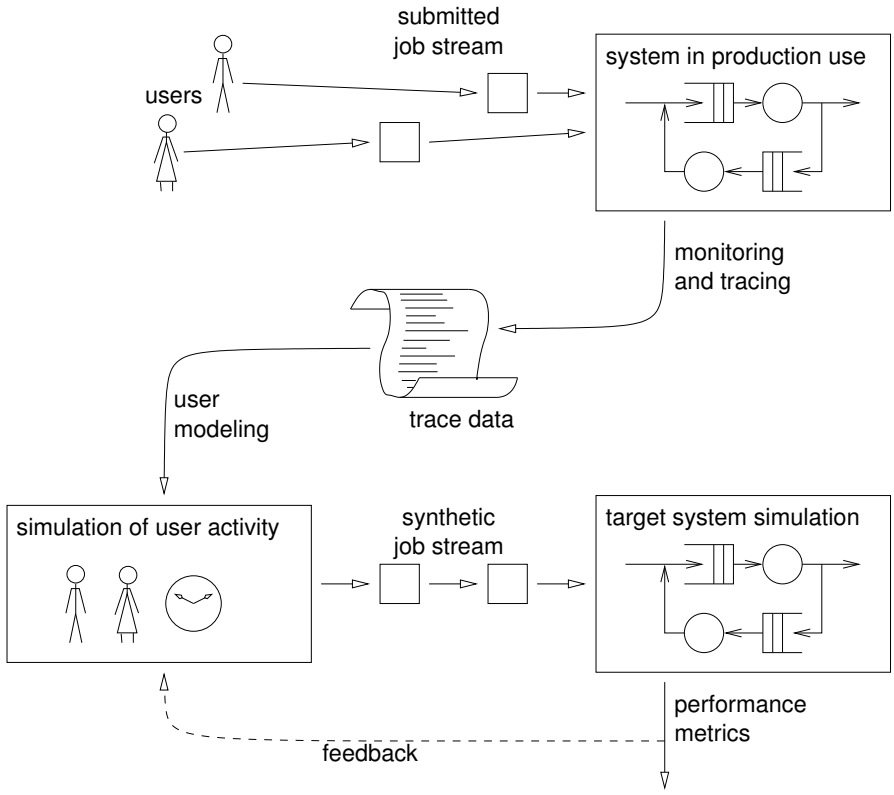


Figure 1.6: *Generative workload modeling based on analyzing how users behave when submitting jobs to a real system (top), and re-creating such behavior in performance evaluation (bottom).*

Generative Modeling

Generative models are indirect, in the sense that they do not model the workload distributions explicitly. Instead, they model the process that is supposed to have generated the workload. If this modeling is done well, it is expected that the correct distributions will be produced automatically. For example, assuming that files are created by modifying previous files, and that this can be modeled as multiplying the file size by a certain factor, leads to a lognormal file-size distribution [188]. Figure 1.6 compares this with the statistical approach. Note that the framework is the same as in Figure 1.5; the only difference is that the model is not a statistical summary of workload attributes but rather an operational description of how users behave when they generate the workload (e.g., how their behavior depends on the time of day and on system performance).

An important benefit of the indirect modeling achieved by the generative approach is that it facilitates manipulations of the workload. It is often desirable to be able to change the workload conditions in some way as part of the evaluation. Descriptive models do not offer any clues regarding how to do so. But with generative models, we can modify the workload-generation process to fit the desired conditions, and the workload itself

will follow suit. For example, we can model different types of file editing sessions, and get different file-size distributions that match them.

Another important benefit of the generative approach is that it supports the modeling and study of possible interactions between a system and its users. This is an extension of the well-known distinction between open and closed system models: in open models new work is completely independent of system conditions, whereas in closed models new work arrives only as a direct consequence of a previous termination. But in real life the situation is more complicated, and feedback from the system's performance to the users can affect the generation of additional work in various ways [614, 615].

Degree of Detail

The questions of what exactly to model and at what degree of detail are hard ones. On one hand, we want to fully characterize all important workload attributes. On the other hand a parsimonious model is more manageable, because there are less parameters whose values need to be assessed and whose influence needs to be studied. Also, in a detailed model there is a danger of overfitting a particular workload at the expense of generality.

In statistical terms (applicable when we are using a statistical model) this can be framed as a tradeoff between accuracy and complexity, as has been formulated using the Bayes information criterion (BIC) [603, 512]. Given a set of model parameters θ, define

$$BIC(\theta) = -2 \log L(\theta) + \frac{p}{2} \log n$$

where L is the likelihood of the data assuming the given parameters[2], and represents the goodness of the fit (that is, the accuracy), p is the number of parameters in the model, namely the size of θ, and n the number of data samples used to fit the model. So the second term reflects a penalty for the complexity of the modeling procedure. By selecting the model with the minimal BIC we trade off complexity and accuracy, and identify the model that gives the best results for the minimal cost. Akaike's information criterion (AIC) uses a similar formula, the main difference being that the penalty does not depend on n [20].

The main problem with models, as with traces, is that of representativeness. That is, to what degree does the model really represent the workload that the system will encounter in practice? The answer depends in part on the degree of detail that is included. For example, each job is composed of procedures that are built of instructions, and these procedures and instructions interact with the computer at different levels. One option is to model these levels explicitly, creating a hierarchy of interlocked models for the different levels [103, 100, 558] (an example of generative modeling). This has the obvious advantage of conveying a full and detailed picture of the structure of the workload. In fact, it is possible to create an entire spectrum of models spanning the range from condensed rudimentary models to direct use of a detailed trace.

[2] These notions are explained later in the book.

As another example, the sizes of parallel jobs need not be modeled independently. Rather, they can be derived from a lower level model of the jobs' structures [247]. Hence the combined model will be useful both for evaluating systems in which jobs are executed on predefined partitions, and for evaluating systems in which the partition size is defined at runtime to reflect the current load and the specific requirements of jobs.

The drawback of this approach is that, as more detailed levels are added, the complexity of the model increases. This is detrimental for three reasons. First, more detailed traces are needed in order to create the lower levels of the model. Second, it is commonly the case that there is greater diversity at lower levels. For example, there may be many jobs that use 32 nodes, but at a finer detail, some of them are coded as data parallel with serial and parallel phases, whereas others are written using MPI in an SPMD style. Creating a representative model that captures this diversity is hard, and possibly arbitrary decisions regarding the relative weight of the various options have to be made. Third, it is harder to handle such complex models. Although this consideration can be mitigated by automation [629, 404], it leaves the problem of having to check the importance and impact of very many different parameters.

The flip side of these considerations is that sometimes we actually want to fit a specific workload, and not be generally representative. In fact, performance evaluations may be roughly divided into two major categories:

- Evaluations of general approaches, as is typically done in academic research papers. Such evaluations strive to make general claims; hence they require generally representative workloads. They focus on invariant aspects of the workload and attempt to average out the differences.
- Evaluations of specific installations, as is typically done for tuning, adjusting configurations, and making upgrade decisions. In this case the particulars of the local workload may be more important than the features that are common with many other installations [426].

A related issue is the variability between workloads observed at different times, which may be just as big as the variability between workloads observed at different locations [426]. In fact, one of the biggest risks of workload modeling is the tendency to focus exclusively on invariants and averages and to disregard variability. The reason for this tendency is that large variations in the workload often lead to large variations in performance results, and subsequently to vague recommendations. But ignoring variability in the interest of crisp results does not make the problematic variability go away — it just makes the results unreliable.

Completeness

An important point that is often overlooked in workload modeling is that *everything* has to be modeled. It is not good to model one attribute with great precision, but to use baseless assumptions for the others.

The problem is that assumptions can be very tempting and reasonable, but still be totally untrue. For example, it is reasonable to assume that parallel jobs are used for

speedup, that is, to complete the computation faster. After all, this is the basis for Amdahl's Law. But other possibilities also exist — for example, parallelism can be used to solve the same problem with greater precision rather than faster. The problem is that assuming that speedup is the goal leads to a model in which parallelism is inversely correlated with runtime, and this has an effect on scheduling [450, 233]. Observations of real workloads indicate that this is not the case, as shown above. Thus it is important to model the correlation between runtime and size correctly.

Another reasonable assumption is that the daily cycle of user activity does not have much importance, because we are usually interested mainly in how the system operates under peak load. However, in non-interactive settings the mere existence of a non-peak period which can be used to make up for overload during the peak hours can make a big difference [248]. When interactive and batch workloads are mixed, delaying batch work and executing it at night may free up important resources for better support of the interactive work during the day. Thus modeling the daily cycle may actually be crucial for obtaining meaningful results.

The implications of assuming something is unimportant can be rather subtle. A case in point is the issue of user runtime estimates in parallel system. The reasonable assumption is that users will provide the system with accurate estimates of job runtimes when asked to do so. At least on large-scale parallel systems, users indeed spend significant effort tuning their applications, and may be expected to have this information. Moreover, backfilling schedulers reward low estimates but penalize underestimates, thus guiding users toward convergence to accurate estimates. Nevertheless, studies of user estimates reveal that they are often highly inaccurate, and may represent an overestimate by a full order of magnitude [507, 431]. Surprisingly, this can sway the conclusions when comparing schedulers that use the estimates to decide whether to backfill jobs (that is, to use them to fill holes in an existing schedule) [237, 697]. Thus assuming accurate estimates may lead to misleading evaluations [693].

1.4 Roadmap

As noted earlier, workload modeling is based on workload data. The next chapter therefore contains a discussion of where the data comes from. The next three chapters after that deal with distributions and fitting them to data: Chapter 3 provides essential background on commonly used distributions, Chapter 4 then presents the issues involved in fitting distributions to data, and Chapter 5 extends this discussion to the special case of heavy tails. Then come two chapters on correlation: first locality and correlation among workload attributes in Chapter 6, and then long-range dependence and self-similarity in Chapter 7. The book ends with chapters on hierarchical generative models (Chapter 8) and case studies (Chapter 9).

2

Workload Data

Workload modeling is based on the analysis of workload data. But where does the data come from? There are two main options: analyze data that is available anyway, or collect data specifically for the workload model. Collecting data can be done in two ways, using active or passive instrumentation. Whatever its source, the data needs to be carefully inspected to eliminate unrepresentative artifacts. Importantly, collected data can and should be made publicly available for use by other researchers.

Issues such as representativeness necessarily relate to modeling approaches as discussed later in the book. Thus we sometimes make use of concepts here that will only be defined and discussed in subsequent chapters. In most cases this should not be a problem.

2.1 Data Sources

Data about workloads is collected from real systems. With luck, the data may already be collected for you. Otherwise, you will need to instrument the system to collect the data yourself.

2.1.1 Using Available Logs

The most readily available source of data is from accounting or activity logs [654]. Such logs, which are kept by many systems for auditing, record selected attributes of all activities. For example, many computer systems keep a log of all executed jobs. In large-scale parallel systems, these logs can be quite detailed and are a rich source of information for workload studies. Web servers are also often configured to log all requests.

Note, however, that such logs do not always exist at the desired level of detail. For example, even if all communication on a web server is logged, this is done at the request level, not at the packet level. To obtain packet-level data, specialized instrumentation is needed.

Example: Analyzing an Accounting Log

A good example is provided by the analysis of activity on the 128-node NASA Ames iPSC/860 hypercube supercomputer, based on an accounting log. This log contained information about 42,264 jobs that were submitted over a period of three months. An excerpt of the log is shown in Figure 2.1. For each job, the following data was available:

- User ID. Special users (such as system administrators and Intel personnel) were singled out.
- Application identifier. Applications that were submitted directly were identified consistently, meaning that if the same application was run more than once the same identifier was used. This information was not available for jobs submitted via the NQS batch system, so these jobs were just numbered sequentially. Unix utilities (e.g. pwd, nsh) were identified explicitly.
- Number of nodes used by the job. Because each job has to run on a subcube, this number is always a power of two. Zero nodes indicates that the job ran on the host computer, rather than on a partition of the hypercube.
- Runtime in seconds.
- Start date and time.

In addition, the log contained special entries to record special situations, such as down-time and dedicated time in which only a select user could access the machine. For example, the special entry with code D in Figure 2.1 indicates nearly two hours of dedicated time, which seems to have gone unused in this case.

At first glance it seems that the information about each job is rather puny. However, an analysis of this log provided a wealth of data, part of which is shown in Figures 2.2 to 2.10 (for a full analysis, see [244]). At the time, all this represented new data regarding the workload on parallel supercomputers. Highlights include the following.

A whopping 28,960 jobs (68.5%) ran on a single node (Figure 2.2). Of these, 24,025 (56.8% of the total) were invocations of the Unix pwd command[1] by system support staff — apparently a practice used to check that the system was operational. But all the sequential jobs together used only 0.28% of the total node-seconds (Figure 2.3). Large jobs with 32 nodes or more accounted for 12.4% of the total jobs, but 92.5% of the node-seconds used. Thus the average resource usage of large jobs was higher than that of small jobs (Figure 2.4). This was due not only to the fact that they used more processors, but also to the fact that their runtimes tended to be longer. This also means that if you would pick a random job on this system, it would most likely be sequential. But if you would look at what a randomly-chosen node is running at an arbitrary instant, it would most probably be running a process belonging to a large job.

[1] Unix employs a hierarchical file system, in which files are identified by their path from the root of the file system. To make things simpler, the system supports the notion of a "working directory". File names that do not begin at the root are assumed to be relative to the working directory. pwd is a command that prints the path of the working directory.

```
user2     cmd2       1     13 10/19/93 18:05:14
sysadmin  pwd        1     21 10/19/93 18:06:03
user8     cmd33      1     31 10/19/93 18:06:10
sysadmin  pwd        1     16 10/19/93 18:06:57
sysadmin  pwd        1      3 10/19/93 18:08:27
intel0    cmd11     64    165 10/19/93 18:11:36
user2     cmd2       1     19 10/19/93 18:11:59
user2     cmd2       1     11 10/19/93 18:12:28
user2     nsh        0     10 10/19/93 18:16:23
user2     cmd1      32   2482 10/19/93 18:16:37
intel0    cmd11     32    221 10/19/93 18:20:12
user2     cmd2       1     11 10/19/93 18:23:47
user6     cmd8      32    167 10/19/93 18:30:45
user6     cmd8      32    336 10/19/93 18:38:58
user6     cmd8      32    278 10/19/93 18:45:07
user6     cmd8      32    149 10/19/93 18:50:19
user6     cmd8      32     83 10/19/93 18:53:25
user6     cmd8      32    123 10/19/93 18:55:56
special   CUBE       D   6780 10/19/93 19:00:00
user11    nqs126    64   4791 10/19/93 20:53:58
user2     nqs127    64  10926 10/19/93 20:53:58
sysadmin  pwd        1      3 10/19/93 22:14:50
sysadmin  pwd        1      4 10/19/93 22:21:57
sysadmin  pwd        1      3 10/19/93 22:29:15
user29    cmd211    64     29 10/19/93 22:31:46
user29    cmd211    64   4885 10/19/93 22:34:44
intel0    nsh        0     67 10/19/93 23:26:43
intel0    nsh        0     17 10/19/93 23:28:15
root      nsh        0     31 10/19/93 23:28:47
user0     nqs128   128   8825 10/19/93 23:56:12
user1     nqs129   128   9771 10/20/93 02:23:21
sysadmin  pwd        1     16 10/20/93 06:21:25
sysadmin  pwd        1     16 10/20/93 06:21:52
sysadmin  pwd        1     15 10/20/93 06:22:19
sysadmin  pwd        1     16 10/20/93 06:22:45
sysadmin  pwd        1     15 10/20/93 06:32:38
sysadmin  pwd        1     15 10/20/93 06:33:07
sysadmin  pwd        1     15 10/20/93 06:33:35
sysadmin  pwd        1     14 10/20/93 06:34:08
user2     nsh        0     10 10/20/93 06:44:05
user2     cmd1      64   4474 10/20/93 06:44:18
user2     cmd2       1     20 10/20/93 06:57:59
user7     cmd9       8    110 10/20/93 07:05:19
user2     cmd2       1     15 10/20/93 07:08:46
user7     cmd9       8     78 10/20/93 07:45:41
user7     cmd9      32     11 10/20/93 07:47:24
user7     cmd9       8    203 10/20/93 07:47:42
user2     cmd2       1     16 10/20/93 07:50:51
user7     cmd9       8    175 10/20/93 07:53:24
```

Figure 2.1: *Sanitized excerpt of data from the NASA Ames iPSC/860 log.*

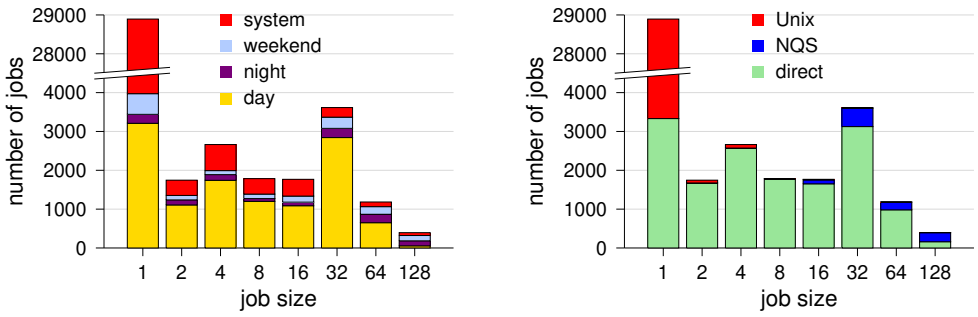


Figure 2.2: *The distribution of job sizes in the NASA Ames log, showing a classification according to source (system vs. users at day, night, or weekend) or according to type (direct interactive jobs vs. NQS batch jobs).*

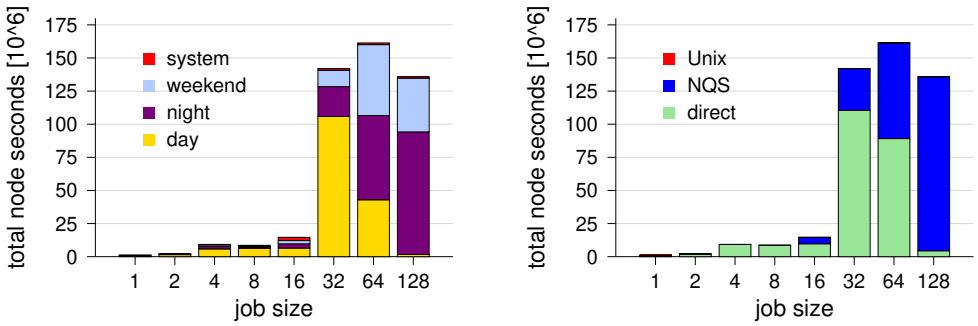


Figure 2.3: *The distribution of job sizes in the NASA log when jobs are weighted by their resource requirements in total node-seconds.*

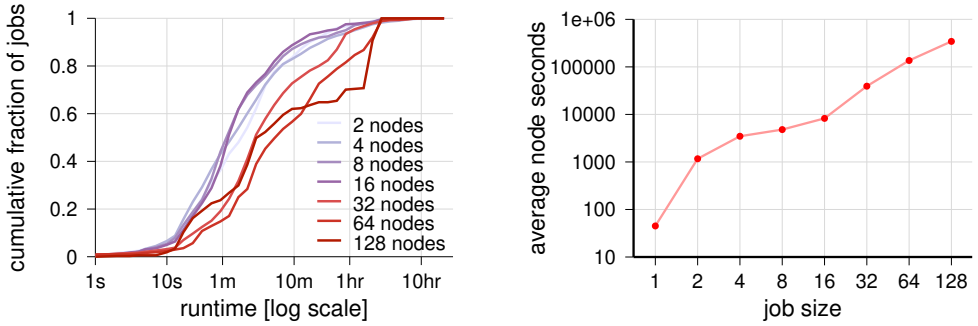


Figure 2.4: *The relationship between job runtimes and sizes. Left: distribution of runtimes for jobs with different sizes. Right: average node-seconds as a function of job size. Note use of a logarithmic scale.*

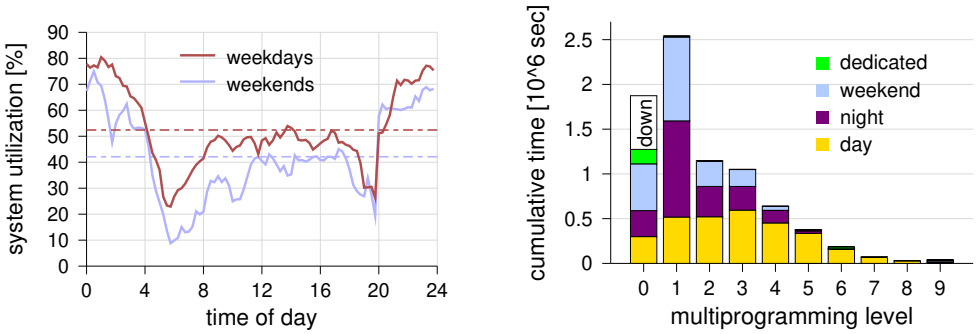


Figure 2.5: *Utilization and multiprogramming. Left: the NASA hypercube system utilization at different times of day, compared with the overall average. Right: the degrees of multiprogramming for the whole log.*

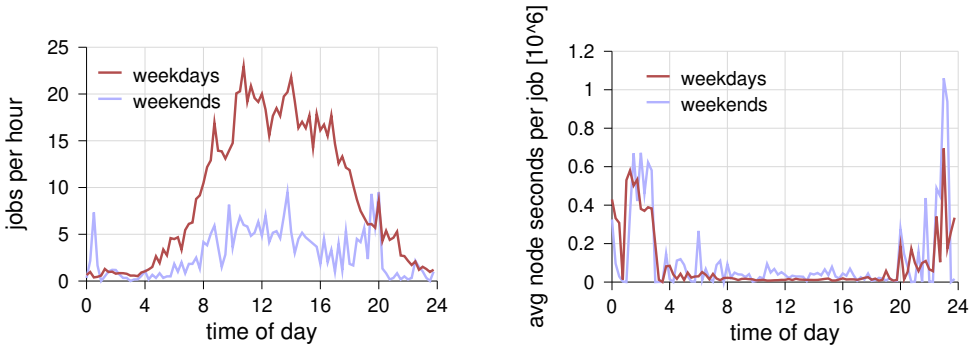


Figure 2.6: *Job submittal pattern. Left: distribution of job arrivals at different times of day. Right: average node-seconds required by jobs that arrive at different times.*

An important observation is that the data about start day and time implicitly contains information about what jobs ran during prime time or nonprime time and during the weekend. Furthermore, it is possible to determine the detailed distribution of jobs at different times of the day. This is also shown in Figures 2.2 and 2.3.

The average utilization of the machine over the whole log was 50%. It was higher on workdays than on weekends, and higher at night when the NQS batch system was active (Figure 2.5). The degree of multiprogramming tended to be higher during the day, as more smaller jobs could run. Recall that 128-node jobs that block the whole machine were typically run using NQS at night. Most jobs ran during work hours, with a slight dip for lunch (Figure 2.6). But jobs running at night required more resources. The distribution of interarrival times during the day seems to be more or less normal (bell shaped) in log-space, with most of the mass between about 30 seconds to about 10 minutes. During the night and on weekends the distribution is flatter (Figure 2.7).

Although most applications were only run once, some were run many times, and even on many different partition sizes (Figure 2.8). Moreover, the same application was

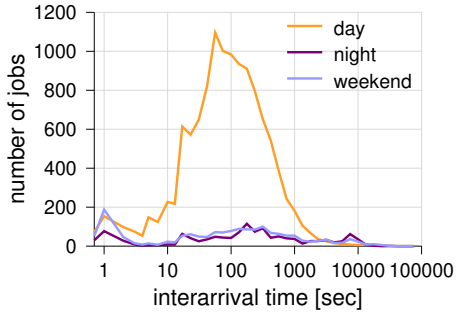


Figure 2.7: *The distribution of interarrival times.*

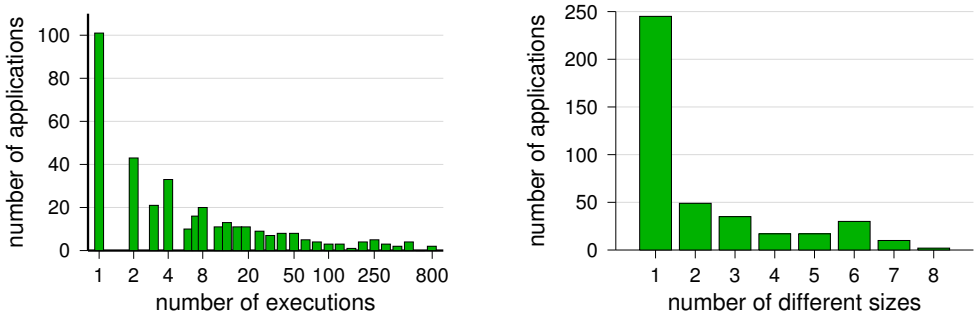


Figure 2.8: *Repeated execution of applications. Left: distribution of reuse of applications: some were run hundreds of times (note the logarithmic scale: bars represent increasing ranges of numbers). Right: some applications were also executed on multiple partition sizes.*

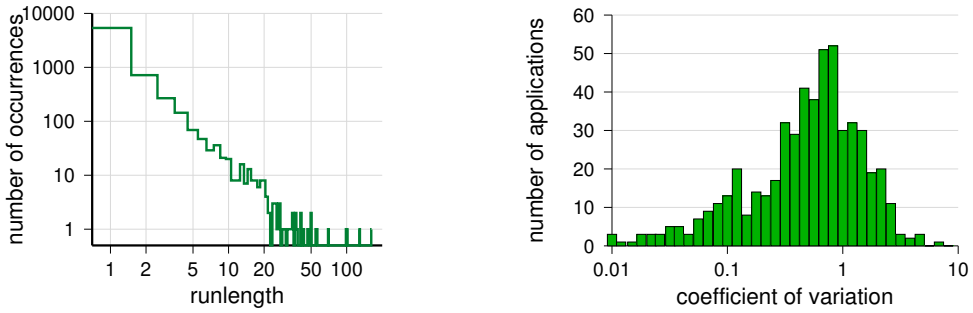


Figure 2.9: *Repetitive work by the same user. Left: distribution of run lengths. Some applications were run repeatedly by the same user more than a hundred times in a row. Right: when the same application was run repeatedly, the coefficient of variation of the runtimes tended to be lower than 1.*

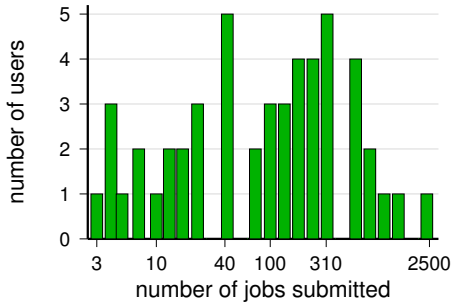


Figure 2.10: *Distribution of levels of activities of different users. Some submitted thousands of jobs (note the logarithmic scale: bars represent increasing ranges of numbers).*

often run repeatedly by the same user (Figure 2.9). Focusing on such situations, it seems that the dispersion of runtimes is rather low, indicating that it is possible to predict future runtimes based on past measurements. The level of activity displayed by different users is also very varied: some ran few jobs and others thousands (Figure 2.10).

Details Box: Web Server Logs

Analysis is simplified if the analyzed log has a standard format. One such example is the common log file format used by most web servers to log their HTTP activity. This format specifies that the log file be a plain ASCII file, with a line for each entry. Each entry includes the following space-separated fields.

Source — The hostname or IP address of the host from which the request was sent. This may indeed represent the client machine, but it may also be some proxy along the way. In particular, when clients access a web server via an ISP, it is possible for one IP address to represent multiple clients, and also for the requests of a single client to come from multiple IP addresses (if each one passes through a different proxy) [579, 461].

User — The user name of the user on the source host. Note, however, that many installations do not record user data because of privacy concerns. This and the next fields are then given as "-".

Authentication — The user name by which the user has authenticated him- or herself on the source host.

Timestamp — The date and time, surrounded by square brackets (e.g., [01/Jul/1995:00:00:01 -0400]). The -0400 is a time-zone offset — the difference from UTC (Coordinated Universal Time, the new Greenwich time) at the time and place that the log was recorded.

Request — The request itself, surrounded by double quotation marks (for example, "GET /~feit/wlmod/"). The request starts with the request type, also called the "method". Typical types are

GET — Retrieve the requested document. The GET can be conditional on some property of the requested document, e.g., that it has been recently modified. The document is always accompanied by HTTP headers.

HEAD — Retrieve HTTP header that would accompany an identical GET request, but do not retrieve the document itself. This can be used to check whether a cached copy is up to date.

POST — Send data to the server. This is typically used when filling out a web form.

Experience shows that the vast majority of requests are GET; even on a blogging site, where POST is used to post new content or comments, only 2.5% of requests are POST [377]. In addition there are PUT, DELETE, TRACE, and CONNECT requests, but these are also seldom used.

Status — The HTTP status code returned to the client. Common codes include

200 — The request was executed successfully.

204 — The request succeeded, but no content is being returned.

206 — The request succeeded, but only partial content is being returned.

301 or 302 — The requested page was moved either permanently or temporarily

304 — The requested document has not been modified, so it does not have to be returned again.

401 — The request failed because authorization is required. This may happen when a login attempt fails.

403 — The request failed because access is forbidden.

404 — The request failed because the requested document was not found.

504 — The server failed to fulfill the request because it timed out trying to get service from another server.

Size — The size in bytes of the contents of the document that was returned.

Regrettably, life is not always so simple. For starters, an extended common log format has also been defined. This appends another two fields to each entry:

Referrer — The page that contained the link that was clicked to generate this request, surrounded by double quotes. The semantics of this field are somewhat murky when frames are used [579].

User agent — The name and version of the browser used by the user, surrounded by double quotes. This sometimes indicates that the request does not come from a user browser at all, but from some robot crawling the web. Note, however, that the data may not be reliable, as some browsers may masquerade as others.

In addition, all web servers also have their own proprietary formats, which typically have the option to record many more data fields.

End Box

2.1.2 Active Data Collection

If data is not readily available, it should be collected. This is done by instrumenting the system with special facilities that record its activity. Major challenges with this process are doing it without being obtrusive or modifying the behavior of the system while measuring it.

Passive Instrumentation

Passive instrumentation refers to designs in which the system itself is not modified. The instrumentation is done by adding external components to the system that monitor system activity but do not interfere with it. This approach is commonly used in studies of communication, where it is relatively easy to add a node to a system that only listens to the traffic on the communication network [436, 698, 303, 261, 279]. It is also possible to monitor several locations in the network at the same time, and correlate the results [738].

An important consideration when monitoring network traffic is that both directions of flow should be monitored. Otherwise, one may see requests from a server but not the replies, or vice versa. At a lower level, one may see packets but not the acknowledgments, thereby losing important information on network conditions. Luckily, the best place to observe both directions is at end nodes, where it is also easiest to deploy monitoring devices [484].

When monitoring it is also possible to perform online analysis. For example, the Tstat tool analyzes packet headers in order to follow TCP flows [484]. This enables the identification of flows that were aborted without being closed properly. The tool can also gather statistics regarding the relaitve use of different protocols, the values of header fields such as time-to-live, which indicates how many hops the packet has passed, and many other parameters. It can also identify Applications based on packet signatures, finite-state machines, or more sophisticated classification schemes [261].

The amount of hardware needed for monitoring depends on the complexity of the system and the desired data. For example, in 2007 commodity hardware could be used to record packet traces on a 1 GB/s link, but not on a 10 GB/s link [597]. A rather extreme example is a proposal to add a shadow parallel machine to a production parallel machine, with each shadow node monitoring the corresponding production node, and all of them cooperating to filter and summarize the data [585].

Active Instrumentation

Active instrumentation refers to the modification of a system so that it will collect data about its activity. This instrumentation can be integrated into the original system design, as was done for example in the RP3 [403]. However, system modification is more commonly done after the fact, when a need to collect some specific data arises.

A potential problem with instrumentation is that it requires the system to be reinstalled. A good example is the Charisma project, which set out to characterize the I/O patterns on parallel machines [517]. This was done by instrumenting the I/O library and requesting users to relink their applications; when running with the instrumented library, all I/O activity was recorded for subsequent analysis. However, the activity of users who did not relink their applications was not recorded.

In the specific case of libraries this problem may be avoided if the library is loaded dynamically as is now common practice, rather than being compiled into the executable. But a more general solution is to perform the instrumentation itself dynamically. This

is the approach taken by the Dyninst framework, which can patch binaries of running programs without requiring a recompilation or even a restart [339, 94]. The idea is to first write a snippet of code that should be introduced into the running program (e.g. to record the occurrence of a certain event). Then identify the point in the program where the snippet should be inserted. Replace the instruction at that location by a branch instruction to the new snippet; at the end of the snippet add the original instruction and a branch back. This has the effect of running the snippet before the original instruction each time control arrives at this point in the program.

When performing active instrumentation, an important consideration is the scope of the collected information. An interesting example is provided by the PatchWrx toolset, which was designed to collect information about applications running under Windows NT. Such applications naturally make extensive use of libraries and system services. Thus, to gain insight into the behavior of applications and their usage of the system, both the applications themselves and the system need to be monitored. To do this, PatchWrx modifies the Privileged Architecture Library to reserve a buffer of physical memory before the system is booted (this is specific to the Digital Alpha architecture) [112]. It also patches the binary images of the kernel and all major libraries and adds instrumentation code that uses the Privileged Architecture Library to record events in this buffer. Using this pre-allocated buffer reduces overhead without having to make significant modifications to the system, because the system simply does not know about this memory at all. Moreover, the same buffer is used to store information across all applications and the operating system.

There are cases in which the required data is inaccessible, and clever techniques have to be devised in order to extract it indirectly. A case in point is harvesting the addresses of memory accesses. Modern microprocessors contain a sizable on-chip cache, so most of the memory references are never seen off-chip [705]. Only the misses can be observed directly, but the statistics of the misses can be quite different from the statistics of the original address stream.

The growing recognition that performance is important but hard to understand has led many modern systems to be designed with builtin performance monitors that can be activated at will. For example, Pentium and later processors include counters that can be configured to count a host of event types [122, 651, 652]. While most of these event types relate to the processor's performance (e.g., counting cache misses, various types of stalls, and branch mispredictions), some can be used to characterize the workload. Examples include counting multiplication operations, division operations, MMX operations, loads, and stores. Significantly, counting can be limited to user mode or kernel mode, thus enabling a characterization of user applications or of the operating system.

Example Box: Using Performance Counters with PAPI

A big problem with performance counters is that they are platform specific: the way to activate and access the counter of floating-point multiplication instructions is different on a Pentium, an Opteron, or a Power machine. Moreover, some architectures may not even provide a certain counter, and in others the desired metric may not be available directly but may be derived based on other counters that are available.

To make life simpler, several projects provide high-level interfaces to these counters. One is PAPI, which stands for "Performance API" (see URL icl.cs.utk.edu/projects/papi/). This provides a large set of predefined events that one may want to count, functions in C and Fortran to configure and access them, and implementations for a host of platforms including all the more commonly used ones. In addition, it also supports direct access to the platform's original events.

As an example of the data that may be obtained, here is a list of the PAPI predefined events relating to branch instructions:

PAPI_BR_INS	total branch instructions
PAPI_BR_CN	conditional branch instructions
PAPI_BR_TKN	conditional branch instructions that were taken
PAPI_BR_NTK	conditional branch instructions that were not taken
PAPI_BR_PRC	conditional branch instructions that were predicted correctly
PAPI_BR_MSP	conditional branch instructions that were mispredicted
PAPI_BR_UCN	unconditional branch instructions

End Box

The examples just described all relate to servers and PCs. But how does one instrument web-based activity? In that case the enterprises that operate the infrastructure have an advantage: Google and Yahoo! can collect web search data from their systems, and Facebook can collect data about social networking activity, but other researchers cannot. But in some cases manipulations are actually possible. For example, Nazir et al. wrote three Facebook applications using the Facebook Developer Platform [511]. The architecture of such applications is that Facebook forward user requests to application servers operated by the application writers. In this way they could collect data about how millions of users used their applications, including about the users' underlying social relations (one of the applications was for sending virtual hugs).

Reducing Interference

Obviously, instrumenting a system to collect data at runtime can affect the system's behavior and performance. This may not be very troublesome in the case of I/O activity, which suffers from high overhead anyway, but may be very problematic for the study of fine-grained events related to communication, synchronization, and memory usage.

Several procedures have been devised to reduce interference. One common approach is to buffer data (as done in PatchWrx [112]). This is based on the observation that recording events of interest can be partitioned into two steps: making a note of the event and storing it for future analysis. Instead of doing both at once, it is possible to note the event in an internal buffer and to output it later. This provides for more efficient data transfer, and amortizes the overhead over many events. Double buffering can be used to overlap the recording of events in one buffer with the storage of events already recorded in the other. Naturally care must be taken not to overflow the buffers and not to lose events.

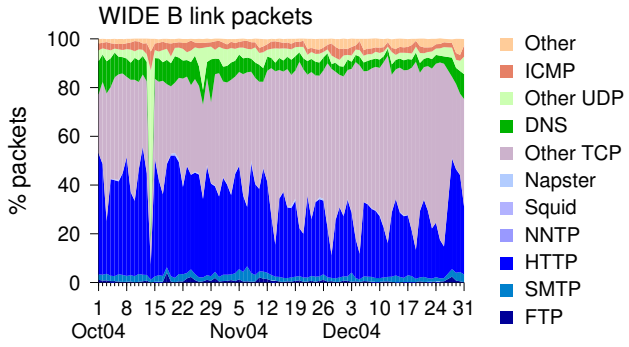


Figure 2.11: *Relative share of Internet protocols at a daily resolution, based on a 15-minute sample taken each day at 2:00 PM.*

Yet another possible solution to the problem of interference is to model the effect of the instrumentation, thereby enabling it to be factored out of the measurement results [463]. This leads to results that reflect real system behavior (that is, unaffected by the instrumentation), but does not address the problem of performance degradation while the measurements are being taken. An alternative is to selectively activate only those parts of the instrumentation that are needed at each instant, rather than collecting data about the whole system all the time. Remarkably, this can be done efficiently by modifying the system's object code as it runs [339].

Sampling

A potential problem with both passive and active data collection is the sheer volume of the data that are produced. For example, full address tracing for a microprocessor would need to record several addresses for each instruction that is executed. Assuming about 10^9 instructions per second, each generating only one address, already leads to a data rate of 4 GB/s. Handling this amount of data not only causes severe interference but can also overwhelm typical storage systems.

A mechanism that is often used in data-intensive situations is sampling. For example, recording only a small subset of the address data may reduce its volume considerably, alleviating both the interference and storage problems. But sampling has to be done correctly in order not to affect the integrity and usefulness of the data. On the one hand, it is preferable to use random sampling rather than a deterministic sampling of, say, every hundredth data item, which may lead to aliasing effects if the data happen to have some periodicity. On the other hand, the approach of selecting a random subset of the data may also be problematic.

An example of problems with sampling is shown in Figure 2.11. This shows the distribution of packets belonging to different Internet protocols, as recorded on a transpacific high-speed link. As can be seen, the distribution changes significantly from day to day. Moreover, there are occasional unique events such as the extremely high percentage

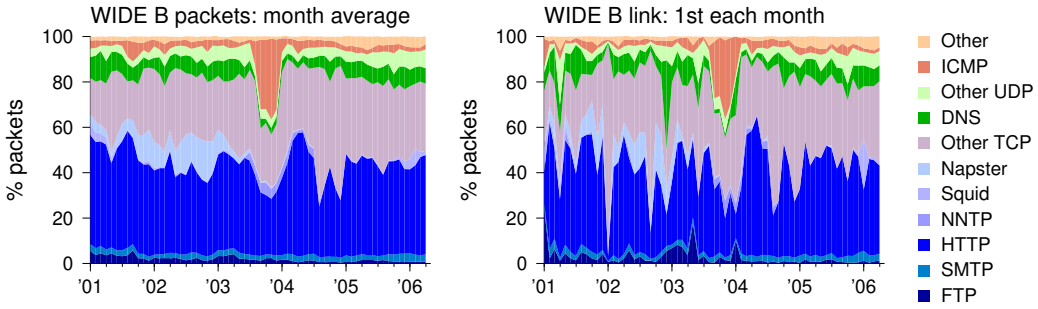


Figure 2.12: *Relative share of Internet protocols at a monthly resolution, comparing an average over the entire month with using the first day of the month as a representative sample.*

of UDP packets on 14 October 2004. If we were to observe the traffic in detail on any given day, we might get a distorted picture that is not generally representative.

In fact, the data as presented in Figure 2.11 may also be a victim of such sampling distortion. Because of the high volume of data, the figure does not really reflect all the traffic that flowed through the link on each day, but rather a 15-minute sample taken each day from 2:00 PM to 2:15 PM. Thus, for example, it is possible that the extremely high percentage of UDP packets observed on 14 October was a relatively short event that did not really affect the whole day's traffic in a significant way. To further illustrate this point, Figure 2.12 compares monthly data about the distribution of Internet protocols, using two approaches. On the left, each month is represented by the average of all its daily samples. On the right, each month is represented by a single sample, taken from the first day of the month. Obviously these single-day samples exhibit much more variability, including unique non-representative cases on 1 January 2002 and 1 December 2002. The aggregated data for each month is much smoother, leading to a feeling of stationarity and representativeness (these concepts are discussed at length later). However, the monthly averaging loses the variability that exists in the original data.

Other problemss occur when the data stream is actually a combination of several independent streams. Random sampling then allows us to estimate how many different streams exist, but may prevent us from fully assessing the characteristics of any given stream. For example, in address tracing, the addresses may be partitioned into disjoint sets that are handled independently by a set-associative cache. Sampling may then reduce the effective locality, affecting the results of cache evaluations. Thus it may be better to record a single associativity set fully than to sample the whole trace [401]. A similar effect occurs in web and Internet traffic, where requests or packets are addressed to different destinations.

The examples just described illustrate substreams that are addressed at distinct destinations. Similar problems may occur when the substreams come from distinct sources. For example, a stream of jobs submitted to a shared computer facility is actually the merging of substreams generated by independent users. It may be that each user's sub-

stream displays significant predictability, because the user repeatedly executes the same set of applications. Such predictability may be diminished or even lost if we sample the whole job stream randomly; a better approach would be to sample at the user level, retaining all the jobs belonging to a random subset of the users. Similar considerations apply when trying to characterize user behavior on the web [98].

To summarize, random sampling may cause problems in all cases in which the data include some internal structure. It is important to design the sampling strategy so as to preserve such structure.

2.2 Data Usability

Once we have data at hand, we should still consider whether or not the data is actually usable. Two concerns are the qualitative assessment of whether the data is representative and worth the effort of modeling, and the technical question of whether the data is stationary and suitable for statistical analysis.

2.2.1 Representativeness

An underlying assumption of basing workload models on logs is that the logs contain data about representative workloads. An important issue is therefore to determine the degree to which data is generally representative. The fear is that the data may simply be irrelevant for general models, because of the specific situation under which it was collected, or due to changes in the technology.

A related issue concerns problems that arise from "bad" data. There are two types of bad data. One is data that should simply be removed before the analysis. This is the subject of Section 2.3, which deals with data cleaning (i.e., how to clean up the data and remove the bad parts). The other is data that is missing or wrong; this problem is obviously harder to handle, and, if it involves large amounts of data, may cause the whole dataset to be unusable.

Missing Data

One problem that may occur is that not all the workload data appear in the log. This often occurs in web server activity logs. Such logs list all the requests served by the server, giving the timestamp, page served, page size, and status of each. The problem is that many client requests do not get to the server at all and therefore are not logged — they are served by proxies and caches along the way [579, 267, 266].

Note that whether this is a significant problem or not depends on the context. If you are interested in the workload experienced by the *server*, the server log gives exactly the right data. In fact, had client-side data been available, you would have faced the considerable difficulty of adapting it to account for all the relevant caching effects. But if you want to study caching, for example, you need *client-side* data, not server-side data. Then the data available from server logs cannot be used directly, as it may be misleading because of having significantly less repetitions than the original client-side workload

[309]. To reconstruct the original client request stream requires a model of the effect of the caches that filtered the requests that arrived at the server.

Another more general type of missing data is that of gaps in the log. Examples can be seen in the recording of traffic on the WIDE B transpacific link shown in Figure 2.31. The main reasons for such gaps are either that the system went down and there was nothing to log, or else the logging infrastructure failed and data was lost.

The effect of missing data on the data quality depends on what you are looking at. For example, when studying job lengths, having a small fraction of jobs go missing does not affect the results. But if we are interested in interarrival times the situation is different. Interarrivals are the intervals from one arrival to the next, so if the system goes down a very long interval will be recorded [131]. But this outlier is an error in the measurement, because it actually reflects the time needed to repair the system, not the time between arrivals. Thus including it in the data makes the data unrepresentative of the real workload.

Erroneous Data

In some cases the reported data may simply be wrong. One example is misinterpreting downtime as interarrival time, as described above. But there are many others.

A concrete example of wrong data is provided by the HTTP log from the France'98 Soccer World Cup. This extensive log contains data about 1.3 billion HTTP requests covering a span of about three months [35]. Among other data, each entry specifies the size of the file that was retrieved. Surprisingly, for more than 125 million requests this size is given as 4 GB, in stark contrast with the average file size of 15.5 KB and the maximal size of 61.2 MB of all other requests. But given that these anomalous large sizes are all exactly $2^{32}-1$, they are actually probably the result of writing a signed value of -1, meaning that the size is not known, in a field that is supposed to be unsigned. In the vast majority of cases this is not a problem, because these entries in the log represent requests that failed and therefore don't have an associated file size. But 8335 of these entries are for requests that were successful. While this is a vanishingly small number of requests relative to the full log, assuming that these requests indeed represent downloads of 4 GB each will completely distort the data.

Other cases may be harder to detect. An example again comes from traces of HTTP requests and responses. As noted earlier, the HTTP header includes a field specifying how much data is being transferred (the file size). But this can also be checked independently by looking at the actual sizes of the underlying IP packets. A study that did so revealed that the HTTP headers claimed to transmit 3.2 times as much data as was actually transmitted in practice [596]. In addition, 35% of the content being transmitted did not match the content type specified in the headers. But these errors in the headers data could not be detected without having access to a trace of the packets themselves.

A special type of errors occurs when the data is actually OK, but it is misinterpreted. For example, consider the data about MapReduce workloads at Facebook available from the SWIM project [125]. MapReduce applications have two stages, a map stage and a reduce stage (this is explained in the box on page 499). The workload data contains three

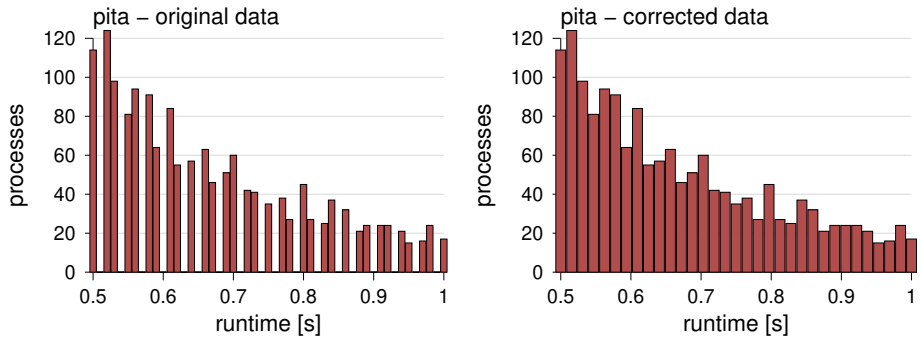


Figure 2.13: *Zoom into histogram of pita runtimes shows a resolution problem in the original data.*

fields that represent the data size at the input to the map stage, the data size transferred from the map stage to the reduce stage, and the data size at the output of the reduce stage. But for many jobs the input data is very small (only a few hundreds of bytes), the transfer size is zero, and the output is huge (megabytes or even gigabytes). A more correct interpretation may then be that these are reduce-only jobs and that the input is not really input to the map stage, but rather input to the job as a whole that specifies what the reduce should do.

Modified Data

Another problem that may occur is that the system modifies the data in some way, and thus the traced data no longer reflects the original workload as it was submitted to the system. Several examples can be found in networking:

- When large datasets are sent in a unit, they are typically fragmented by the underlying communication protocols to accommodate the network's maximum transmission unit. Moreover, fragmentation may happen more than once, with a lower level protocol further fragmenting the fragments that were previously created by a higher level protocol. If the evaluated system does not suffer from the same constraints that led to fragmentation on the traced system, the communication might have been done differently.
- As far as the application is concerned, when it sends a large dataset all the data is available for sending at once. However, the underlying protocols may pace the transmission in order to avoid congestion on the network. For example, this is done by the TCP congestion control mechanism. As a result, the arrival times of packets do not reflect the time that they were made available for transmission. Instead, they reflect the time that the network was ready to accept them [265].

Data can also be modified inadvertently when it is recorded. A bemusing example occurs in the pita dataset of Unix process runtimes from 1998. This data came from a

system log, as reported by the lastcomm command, which includes runtime in seconds with a resolution of two decimal digits. Surprisingly, upon inspection it turned out that there were no processes that ran for the following durations: 0.01, 0.04, 0.07, 0.10, 0.13, 0.15, 0.18, and so on. This is most probably due to using two decimal places to report values that were originally recorded in binary with a resolution of 1/64 of a second. Correcting this by increasing the resolution to six decimal digits leads to much more reasonable data (Figure 2.13).

Local Constraints

The size of the maximum transfer unit of a network is but one example of local procedures and constraints that may affect the collected workload data. For example, data on programs run on a machine equipped with only 512 MB of physical memory will show that programs do not have larger resident sets, but this is probably an artifact of this limit, and not a real characteristic of general workloads. Likewise, parallel jobs run on a supercomputer with an administrative limit of 18 hours will not run longer, even if the actual calculation takes several weeks. In this case users may use checkpointing to record the achieved progress and restart the calculation whenever it is killed by the system. The log will then show several 18-hour jobs instead of a single longer job.

Workload Evolution

Another problem is that workloads may evolve with time. For example, data from the SDSC Paragon show that the distribution of job runtimes changed significantly from 1995 to 1996 (Figure 2.14 left). Note that each of these distributions represents an entire year of activity by multiple users. So which of these two distributions is more representative in general? In some cases such differences may be the result of changing activity patterns, when users learn to use a new system or leave an old system in favor of a newer one [346]. It is therefore important to capture data from a mature system, and not a new (or legacy) one. On the other hand, one must expect workloads to evolve in systems based on new technologies that have not been available in the past. Examples from recent years include the introduction of Wi-Fi, P2P file sharing, microblogging, social networks, and cloud computing.

Modifications can also occur when a system is in the midst of production use. Human users are very adaptive, and quickly learn to exploit the characteristics of the system they are using. Thus changes to the system may induce changes to the workload, as users learn to adjust to the new conditions. For example, data from a Cray T3D supercomputer at LLNL showed that when the scheduling software on the machine was changed, the distribution of job sizes changed with it. January 1996 was the last full month in which the Unicos MAX scheduler was used. This scheduler did not provide good support for large jobs, so users learned to make do with smaller ones. It was replaced by a gang scheduler that prioritized large jobs, and users soon started to take advantage of this feature (Figure 2.14 right) [240]. The distribution of job sizes may also change due to a

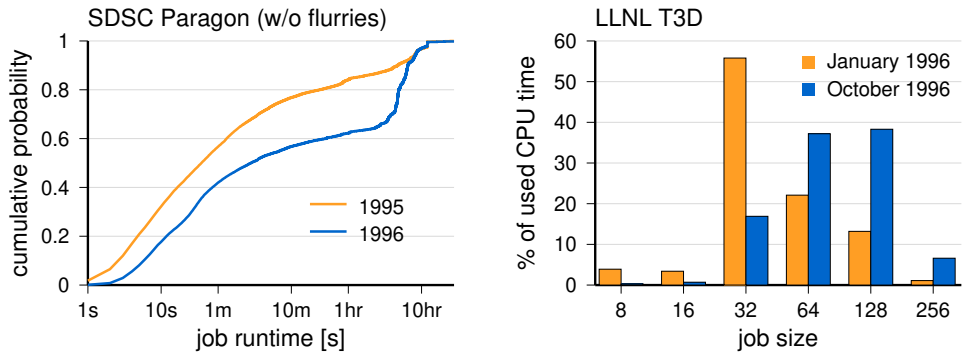


Figure 2.14: *Left: The distributions of job runtimes on the SDSC Paragon in successive years. 1996 had many more long jobs than 1995; is this representative? Right: Change in distribution of job sizes after the scheduler on the LLNL Cray T3D parallel system was replaced.*

configuration change of the underlying hardware, as occurs when additional processing nodes are added.

Another type of evolution occurs when the dominant applications of a system change. For example, the characteristics of Internet traffic have changed over the years, as applications changed from email and ftp to world wide web downloads and peer-to-peer file sharing [252, 328]. For instance, Park et al. report on a significant change in the self-similarity of packet traces between 2002 and 2003 as a result of activity by a new file-sharing application, that exhibited a unique pattern of small UDP packets for pinging neighbors and performing searches [535]. Similar changes have also been observed in wireless traffic [328]. Likewise, the structure of web traffic changed as more objects were embedded in web pages, and as banner ads and delivery services caused more data to be fetched from servers other than the one that served the "top-level" page [330].

Focus on Invariants

Deriving general workload models from data that is subject to local constraints or to evolutionary changes is obviously problematic. The only way to alleviate such problems is to try and collect data from several independent sources — different web servers, different supercomputers, different Internet routers — and analyze all the collected workload logs. Features that appear consistently in all the logs are probably "real" features that deserve to make it into the final model [37, 230, 265, 106]. Features that are unique to a single log are probably best left out. Throughout this book, we attempt to present data from several distinct datasets in our examples, and comparisons of several workloads are also used in the case studies of Chapter 9.

As a preview, consider Figure 2.15. This shows the following parallel workload attributes, all of which show considerable consistency across different installations:

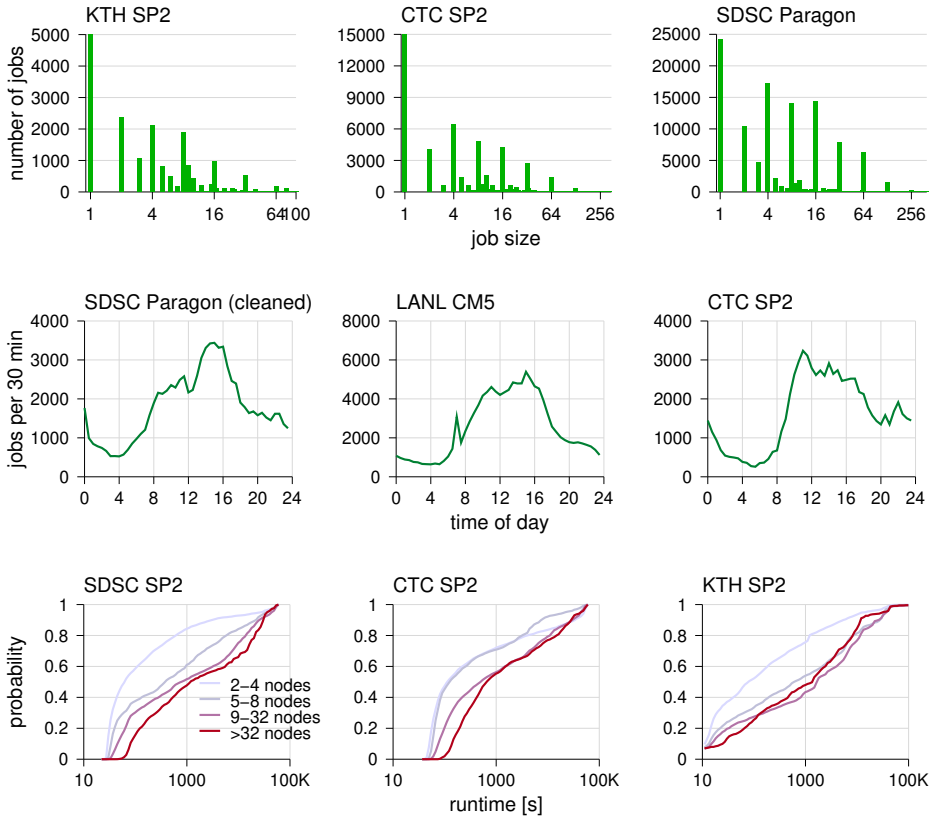


Figure 2.15: *Examples of invariants across distinct workloads.*

- Job sizes (in number of processes) in parallel supercomputers, showing the predominance of serial jobs and powers of two.
- Daily cycle of job arrivals at parallel supercomputers, with low activity after midnight, high activity during working hours, and an intermediate level in the evening.
- Distributions of parallel job runtimes for jobs in different ranges of sizes, showing that small jobs tend to have shorter runtimes, but serial jobs behave more like large jobs.

Of course, even getting data from several sources may not be sufficient to produce a general model. A counterexample is given in Figure 2.16, which shows the fraction of jobs that are serial and powers of two on different parallel systems. The data reveal not only large differences between different classes of systems, but also significant intraclass variability. For example, despite being on costly large-scale parallel machines, the workloads often include a large number of serial jobs. But the fraction of serial jobs varies from 11.6% to 43.5%. Likewise, the fraction of power-of-two jobs varies from 39.7% to 65.7%. It is hard to decide from this data what sort of distribution can be called "typical" or "representative". The systems with 77–97% serial jobs are those in which jobs are

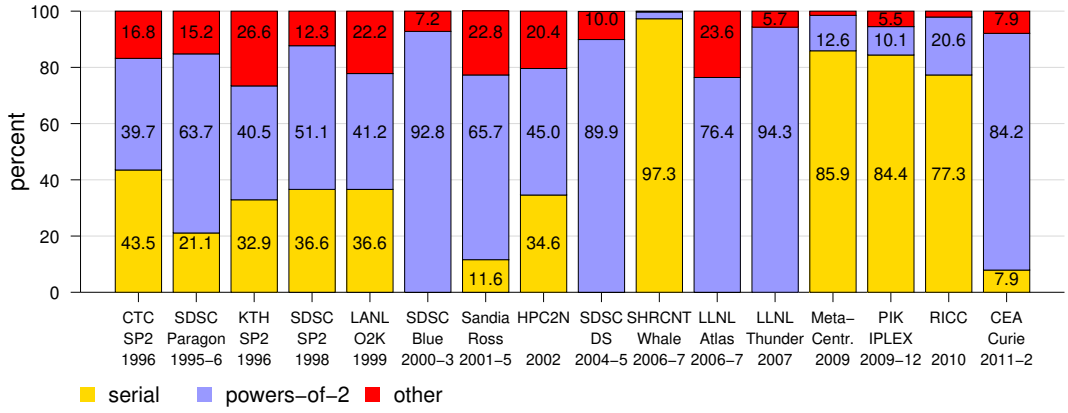


Figure 2.16: *The fraction of serial and power-of-two jobs differs among systems. (Data showing all jobs in each log, without cleaning.)*

usually structured as large collections of individual — hence serial — processes, called "bag-of-tasks". These are typically grid systems. At the other extreme, systems with no serial jobs are those in which the minimal unit of allocation is a node that has more than one CPU. In this case some of the jobs may in fact be serial, but we don't know it.

Finding Representative Slices

The question of representativeness is not unique to workload modeling. It also comes up when using traces directly to drive a simulation. In particular, it is sometimes desirable to find a representative slice of the trace, rather than using all of it. For example, this happens in architecture studies, where a trace can contain billions of instructions. Simulating the whole trace can then take many hours. By finding a representative slice, we can obtain good quality results at a fraction of the cost.

As concrete examples, in the context of evaluating computer architectures and cache designs, Lafage and Seznec suggest using short slices of applications [422], and Sherwood et al. suggest using basic blocks [613]. In both cases these are then clustered to find a small number of repeated segments that can represent the whole application. This approach works because of the high degree of regularity exhibited by typical computer programs, which iterate among a limited number of computational phases. It has an advntage over other approaches employed for trace reduction that use sampling to reduce the effort needed to collect and process a trace, but disregard the effects on representativeness [705].

Measuring Representativeness

The examples just discussed show that the question of representativeness is a difficult one. But what exactly do we mean when we say that a workload or model is representative?

The formulaic approach is to require that a representative workload should *lead to the same results* as the workloads it is supposed to represent [258, 705, 203, 420]. In other words, workload W is representative if using it leads to the same performance evaluation results as would be obtained using other workloads. Model M is representative if using it leads to the same results as would be obtained using real workloads. For example, in the context of microarchitecture studies the workload is a benchmark application. The superficial measure that a benchmark is representative is to check the instruction mix, and verify that it is similar to the mix in other applications. But what we really want is that the benchmark will lead to similar branch prediction behavior, similar data and instruction cache miss rates, and similar performance in terms of instruction-level parallelism and cycles per instruction [203].

The problem with this definition or representativeness is that it is not operational. One reason for this is that it depends on context — a workload may lead to consistent results in one context, i.e., for one specific set of performance measurements, but not in another context. The context in which a workload is used has two components: the system being evaluated, and the performance metric used in the evaluation. We can assess whether a certain workload or model is representative of other workloads only if we decide on a specific system and on a specific way in which to measure performance.

For example, if our workload consists of a sequence of web requests, the system can be a web server with a single disk that serves the requests in the order they arrive, and the performance metric can be the distribution of response times. A workload that is representative in this context will not necessarily also be representative in another context — for example, one in which the server stores its data on an array of a dozen independent disks, incoming requests are served in an order based on the requested page size, and the performance metric is the maximal response time experienced by a small page.

If we are willing to restrict ourselves to one specific context, it is possible to distill a "representative" workload model [420]. This is done by systematically considering different workload attributes, and checking their effects on performance within the specified context. The process is described in Figure 2.17. Starting with an initial list of workload attributes, these attributes are used to guide an analysis of a given workload, and to create a model that matches these attributes in the original workload. For example, an attribute can be the distribution of request sizes, and the model will be based on a histogram of the sizes that appear in the given workload. A simulation of the target system is then performed for both the original workload and the model. If the performance results match for the selected performance metrics, the model is declared representative. If they don't, additional workload attributes are tried.

But in general such an approach is not practical. We therefore typically settle for a definition that a workload is representative if it "looks like" the other workloads it is sup-

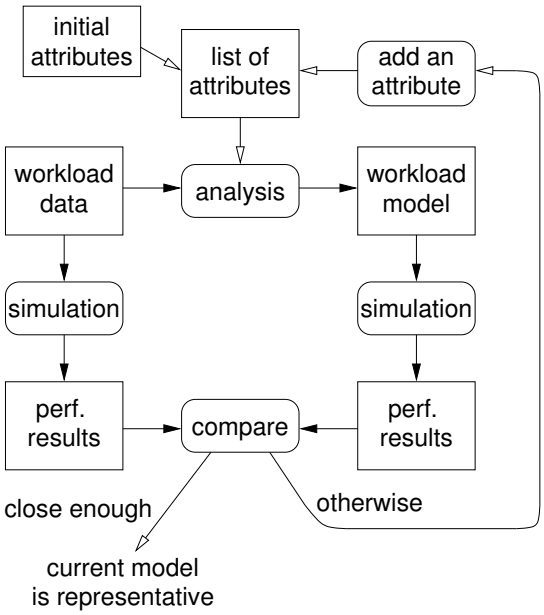


Figure 2.17: *Idealized process of distilling a representative workload (after [420]).*

posed to represent. This is typically interpreted as the workloads having the same statistics: the same distributions of important workload parameters, the same correlations, and so on. Moreover, we may require that workload elements not only look the same but that they appear in the correct context, i.e. that they experience the same competition and interactions with other workload elements as they would in the other workloads [363]. This definition also corresponds to our notion of modeling, in which a good workload model is one that captures the statistics of the real workloads [427].

2.2.2 Stationarity

Statistical modeling consists of finding a statistical model that fits given data. The simplest case is fitting a distribution: we assume that the data is actually samples from an unknown distribution, and try to find this distribution. Implied in this approach is the assumption that the distribution does not change from one moment to the next. In technical terms, this means that we assume that the data is stationary [256].

More intuitively, *stationarity* means that we assume the system is in a steady state. Thus the assumption of stationarity is not only a statistical requirement, but often also a prerequisite for meaningful performance evaluations. After all, if the system is not in a steady state, meaning that its operating conditions change all the time, what exactly are we evaluating? Nevertheless, some forms of nonstationarity, notably the daily cycle of activity, may be important to model in certain situations.

To read more: Stationarity is a basic concept in stochastic processes. It is covered in basic

probability modeling texts such as Ross [581, chap. 10] and in texts on time series analysis such as Chatfield [121].

Definitions

More formally, stationarity is defined as follows. Let us describe the workload as a stochastic process. Thus we have random variables $X_1, X_2, X_3, \ldots$, where X_i represents some property of the workload at instant i (the number of arrivals on day i, the runtime of job i, etc.). A minimal requirement is that all the X_is come from the same distribution, independent of i. This suffices if we are only interested in fitting the distribution. But a full statistical model also includes additional structure, namely all possible correlations between different elements.

To include this additional structure in the definition, we start by selecting a number n and set of indices $i_1, i_2, \ldots i_n$ of size n. This identifies a set of Xs: $X_{i_1}, X_{i_2}, \ldots X_{i_n}$, which have a joint distribution. Next, we can shift the indices by an amount s, yielding a different set of Xs: $X_{i_1+s}, X_{i_2+s}, \ldots X_{i_n+s}$. The process is said to be stationary if this new shifted set has the same joint distribution as the original set. Moreover, this property has to hold for every size n, for every set of indices $i_1 \ldots i_n$, and for every shift s.

The above definition has many consequences. For example, by taking $n = 1$ and different shifts s it implies that all the X_is come from the same distribution, so we do not need to specify this separately. As a result, they all also have the same expected value and variance. Likewise, the $n = 2$ case implies that they have the same covariance.

A simple example of a stationary process is when the X_is are selected independently from the same distribution, because in this case there are no correlations between them. But a process where the X_is are indeed dependent on each other can still be stationary. However, the requirement that all joint distributions be the same is a very strong one. Therefore it is more common to use a weaker version of stationarity. The definition of weak stationarity just requires that the first two moments and the covariance of the process be the same, namely that

1. All Xs have the same expectation, $\mathbb{E}[X_i] = m$, and
2. The covariance function γ of two Xs depends only on the *difference* of their indices, and not on the indices themselves:

$$\mathbb{C}\mathrm{ov}(X_i, X_j) = \gamma(i - j)$$

(Covariance is related to correlation and measures the degree to which two variables deviate from their means in a coordinated manner. This is explained on page 267.)

Stationarity in Workloads

When we look at workload data we can never be completely sure that it is stationary, because we always only see part of it and we don't know how it may change in the

future. But some types of workload data indeed seem to be (weakly) stationary, whereas others do not.

Stationary workloads include those that are generated by a uniform process. For example, the behavior of a single application is often stationary, and corresponds to the inner loop of the application that is repeated over and over again. This applies to architectural studies using instruction traces or address traces. If the application has several computational phases, it may be piecewise stationary — each phase is stationary, but different from the other phases.

High-throughput workloads that are actually the sum of very many individual contributions also tend to be stationary, especially over short intervals. For example, HTTP traffic on the Internet when observed for 90 seconds is practically the same as when observed over 4 hours [330]. But it will not be stationary if observed over long periods; at a minimum, the daily cycle implies that activity at night will be different from that during peak hours.

The stationarity of high-throughput workloads is a result of the fact that a very large number of independent workload items occur during a short period of time. But over longer periods the patterns may nevertheless change. A possible solution to the lack of stationarity in such cases is to settle again for piecewise stationarity. This means that time is divided into segments, such that each is stationary; transitions in the workload occur only at the boundaries of these segments [362]. For example, the daily cycle of job arrivals can be partitioned into three segments, representing high levels of activity during peak hours, intermediate levels in the evening, and low activity at night [131].

In low-throughput workloads a very long period of time is needed to collect large numbers of workload items. During such long periods conditions may change, and different users may become active, leading to changes in the workload and therefore nonstationarity. Examples of workload evolution were cited above, and shifting user activity is covered in Chapter 8.

Another problem that occurs over long time spans is that workloads may contain an inherent dispersiveness. As explained in Chapter 5, many distributions that describe workloads are heavy-tailed. This means that occasionally something "big" happens, which distorts the statistics of the workload seen before it occurred. Thus the workload is always in a transient condition [157, 307], meaning that:

- It takes a very long time to converge to stable statistics,
- It may happen that another "big event" occurs before convergence is achieved, and
- Therefore the observed statistics are actually not representative, because they are a combination of the "normal" workload and the "big events".

Consequently it may be better to try and capture the heterogeneity of the workload rather than trying to find a single representative configuration [264].

Dealing with Cycles

While modeling nonstationary data may be problematic, there exist simple types of nonstationarity that can in fact be handled easily. The most common type of nonstationarity

in computer workloads is to have a cycle — either the daily or the weekly cycle of activity. A workload with cycles can usually be modeled as a combination of a deterministic cyclic component plus a random stationary component:

$$X_i = C_{i \bmod T} + Z_i$$

where Z is the random part, C is the cyclic part, and T is its period or cycle time.

The first step in dealing with cycles is to identify them. This is done by the autocorrelation function, which identifies lags at which the data is similar to itself. For example, if we are looking at data that displays a daily cycle, the data will be similar to itself when shifted by 24 hours. (How this works is described on page 292.)

Once we know the period T, we can proceed to characterize the complete workload as a combination of its two components: the cyclic part and the random part. The first component, denoted by C_t for $0 \le t \le T$, gives the shape of the basic cycle. Given $t = i \bmod T$, C_t is supposed to be the same from cycle to cycle. Note that this does not have to be a nice mathematical function like a sine or polynomial. Thus an empirical function can be obtained by averaging over all the cycles in the available data. This is akin to finding the long-term average of the weather measured on the same date in successive years.

The second component represents the random fluctuations that occur at each instant, quantified as $Z_i = X_i - C_{i \bmod T}$. This is akin to discussing the weather on a certain day in terms of how it *deviates* from the long-term average of the weather measured on the same date in successive years.

Of course, life is not always so simple. For example, there may be slow, long-term changes from cycle to cycle, that preclude describing the cycles using the same function C_t for each one. Instead, C_t must change with time in some way. Returning to the weather analogy, this is analogous to weather patterns that change due to global warming. When such effects exist, a more complicated model is needed.

Dealing with Trends

Another simple type of nonstationarity that can be handled easily is a linear (or more generally, polynomial) trend. This means that the workload changes in a well-defined way with time. If the trend is linear, a certain increment is added with each time unit. Thus, if we subtract this trend, we are left with a stationary process. The full model is then the residual stationary process plus the trend.

The reason for focusing on (or hoping for) polynomial trends is that differentiation will eventually lead to a stationary process [83, 362]. Consider a process with a linear trend. This means that we assume that the process has the structure

$$X_i = A_0 + A \cdot i + Z_i$$

where A represents the deterministic linear trend and Z_i is a random stationary process. The first differences of this process are

$$X_{i+1} - X_i = A + Z'_i$$

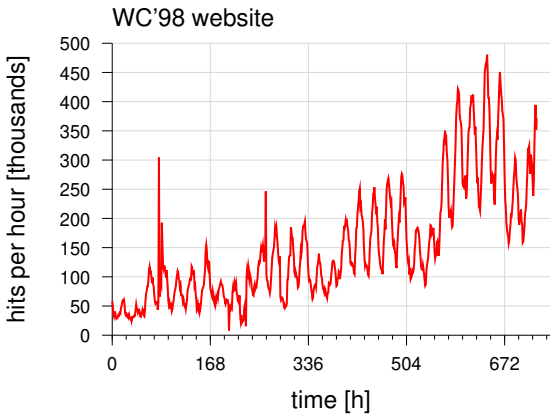


Figure 2.18: *Hits to the WC'98 website, during four and a half weeks before the tournament actually started.*

where $Z'_i = Z_{i+1} - Z_i$ is stationary because it is the difference between two stationary variables.

If the trend is quadratic, second differences need to be used, and so on for higher order polynomials. A simple way to assess whether additional differencing is needed is to look at the autocorrelation function; for a stationary process, it usually decreases to zero pretty quickly. In most cases first differences or at most second differences suffice.

What to Model

An example with both a trend and cyclic behavior is shown in Figure 2.18. The data is hits to the World Cup finals website from 1998, before the tournament actually started. It shows a daily cycle, a weekly cycle, and a steady increasing trend, all superimposed with random fluctuations.

Most of this book deals with techniques that are best applied to stationary data. This could be taken as implying that if workload data includes trends or cycles, these features should be removed before modeling. *This is most likely an incorrect conclusion*. If a workload exhibits a trend or a cyclic behavior at the time scales that are of interest for the evaluation at hand, then these features may be much more important than any other attributes of the workload. Thus one may need to focus on the cycle and trend rather than on what is left over after they are removed.

Modeling of data with trends and cycles, and specifically the data in Figure 2.18, is discussed in more detail in Section 6.5.

2.3 Data Filtering and Cleaning

Before data can be used to create a workload model, it often has to be cleaned up. This may be done for one of four main reasons [249].

1. One type of data that should be removed is erroneous or unusable data. For example, if a certain job has a start record but no end record, we cannot know how long it ran. In many cases the data about the job start is then useless.
2. Another type of data that may be removed is data about erroneous or useless work, e.g., jobs that were aborted or connections that failed. Their removal is justified by a desire to only consider the "good" work performed by the system. However, given that a real system also has to deal with "bad" work, removing such data should be considered with care.
3. A third situation in which some data may be filtered out is when the workload includes several classes, and we are only interested in some of them. In order to focus on the type of workload in which we are interested, we need to filter out the other classes.
4. Workload logs also sometimes include abnormal events that "don't make sense"; filtering them out enables us to focus on the "normal" workload and avoid artifacts [249, 128]. Of course, the decision that something is "abnormal" is subjective. The purist approach would be to leave everything in, because in fact it did happen in a real system. But on the other hand, while strange things may happen, it is difficult to argue for a specific one; if we leave it in a workload that is used to analyze systems, we run the risk of promoting systems that specifically cater for a singular unusual condition that is unlikely to ever occur again. And one must not forget the possibility that the abnormal event was actually an error.

Cleaning data is sometimes shunned due to fears of being accused of tinkering with the data. In such cases it may be useful to consider the opposite perspective of acceptance testing. Given that data that is not filtered out is accepted, you should ask yourself whether the quality of the data you are using is indeed acceptable. If you have doubts, parts may need to be cleaned.

2.3.1 Noise and Errors

One simple aspect of workload cleaning involves handling errors. The simplest of all is obvious logging errors, such as jobs that have a termination record but no initialization record; there is little that can be done with such partial data, so it should be deleted. Likewise, there is probably no use for jobs that had a negative runtime or used a negative amount of memory (unless, perhaps, if you are only interested in arrival times).

Another simple case is measurement errors. For example, consider a study of the process by which new work arrives, and specifically the tabulation of interarrival times (that is, the intervals from one arrival to the next). If the system goes down for three

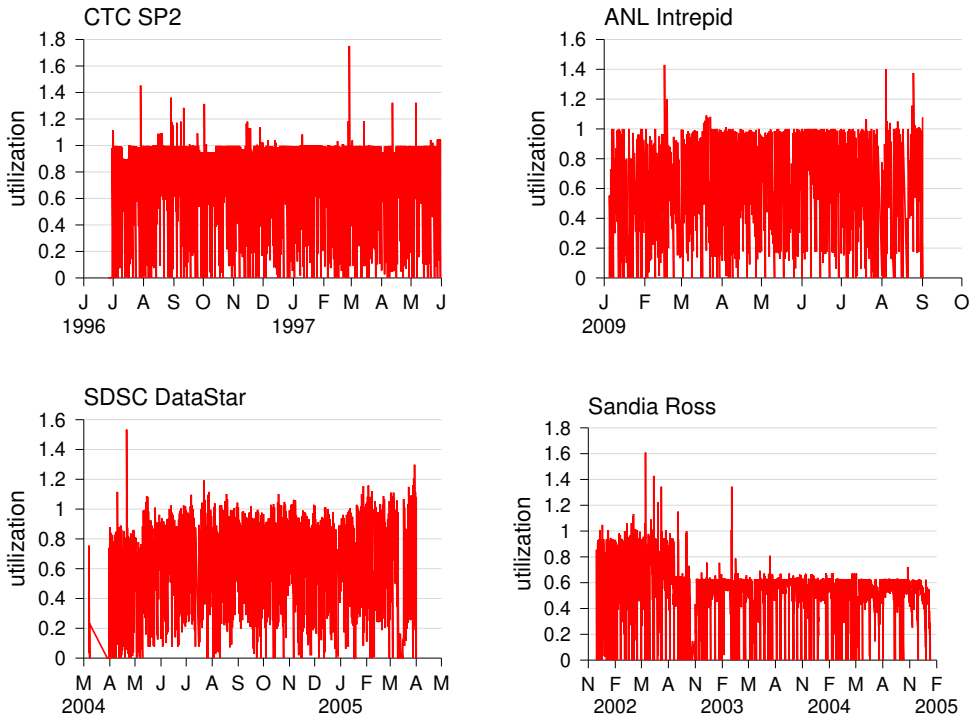


Figure 2.19: *Daily instantaneous utilization at several large-scale parallel systems exhibits load anomalies. The Ross cluster apparently also underwent a configuration change that reduced the number of processors.*

hours, no new work will be recorded in this period. This will cause a very large interarrival time to be tabulated, which should be removed from the interarrival time data [131]. Such cleaning of the data may be facilitated by noticing that no jobs are actually scheduled in this interval [521].

Regrettably, it is not always possible to identify the source of the errors. For example, consider the instantaneous utilization data at several large parallel machines shown in Figure 2.19, namely the fraction of processors that are actively serving some job at each instant; it is calculated by tracking the start and end time of each job, and summing the processors used by jobs that have started but have not yet terminated. Naturally, this should be bounded by the number of processors in the system, but the data seems to indicate that in many instances more than 100% of the processors were allocated [250]. However, such exceptions are not easily attributable to any specific subset of jobs, so it is not clear which jobs should be removed to fix the problem.

In other cases the data may be corrected instead of being removed. An example comes from the France'98 Soccer World Cup HTTP trace described above (page 37). The problem was that some entries specified file sizes of 4 GB, probably as a result of entering a value of –1 into an unsigned data field. In this specific case it is not necessary

to remove these entries; instead, it is possible to search for other entries referring to the same files and use the sizes specified there instead of the erroneous data.

Data logs may also include noise, i.e. data that is correct but irrelevant for the desired need. An example that includes several types of problems comes from a log of Unix sessions, as displayed using the last command (Figure 2.20). One major problem is the pseudo-user "reboot", which is logged whenever the system reboots. The session time given in parentheses is then the time that the system was up, rather than a genuine session time; in the example shown in the figure this is nearly a whole month. Related to this problem are sessions that were disrupted by the reboot, and therefore do not represent the real duration of a full user session (in such cases the end time of the session is given as "crash"). There are also incomplete sessions that simply did not terminate yet when the data was collected. Then there is the concern of how to regard two consecutive sessions by the same user only a minute or two apart: are these indeed individual sessions, or maybe only an artifact of an accidental disconnection, in which case their durations should be summed up? Finally, another potential issue is that two types of sessions are recorded here: some are local, but many are via dialup (as identified by the gateway's hostname).

As the example of crashed sessions demonstrates, workload logs may also contain data about activities that failed to complete successfully. A common example is jobs that were submitted and either failed or were killed by the user. Other examples are requests from a browser to a web server that result in a failure status, including the notorious 404 file-not-found error code, the related 403 permission-denied, and others. Should these jobs or requests be included or deleted from the data? On the one hand, they represent work that the system had to handle, even if nothing came of it. On the other hand, they do not represent useful work and may have been submitted again later. Thus keeping them in the data may lead to double counting.

An interesting compromise is to keep such data and explicitly include it in the workload model [137]. In other words, part of the workload model will be to model how often jobs fail or are aborted by the user who submitted them, or how often requests are made to download non-existent files. This will enable the study of how failed work affects system utilization and the performance of "good" work.

Distortions of real workloads due to double counting are not limited to repeating requests that had failed. For example, this problem happens in web server logs when redirection is employed to serve the requested page even if the client requested it using an outdated URL. But redirection does not occur automatically at the server: instead, the server returns the new URL, and the client's browser automatically requests it. Thus the request will appear twice in the server's log. Luckily, this situation can be easily identified because the original request will have a 301 or 302 status code [579]. A more difficult situation occurs when pages are updated automatically. In this case the repeated requests should probably be retained if we are interested in server load, but not if we are interested in user activity [726].

Yet another example is provided by the Enron email archive. This is a unique large-scale collection of email messages that was made public as part of the investigation into the company's collapse. It includes messages to and from about 150 users, organized

```
user1   ipaddr1      ipaddr1   Tue Apr 29 17:00 - 17:53  (00:53)
user2   ipaddr2      ipaddr2   Tue Apr 29 13:27   still logged in
user3   pts/1        dialup    Tue Apr 29 12:41 - 13:25  (00:43)
user4   pts/2        office1   Tue Apr 29 12:14 - 16:07  (03:52)
user5   pts/3        dialup    Tue Apr 29 11:22 - 13:24  (02:01)
user6   pts/4        office2   Tue Apr 29 10:40 - 10:41  (00:00)
reboot  system boot  cluster1  Tue Apr 29 10:21 - 15:25 (28+05:04)
user3   pts/5        dialup    Tue Apr 29 09:36 - crash  (00:45)
user7   ipaddr3      ipaddr3   Tue Apr 29 09:00 - crash  (01:21)
user8   pts/6        dialup    Tue Apr 29 00:02 - 02:29  (02:26)
user3   pts/7        dialup    Mon Apr 28 20:29 - 22:56  (02:27)
user9   pts/8        dialup    Mon Apr 28 19:51 - 22:20  (02:29)
user10  pts/6        dialup    Mon Apr 28 19:23 - 22:34  (03:11)
user11  pts/6        office3   Mon Apr 28 19:05 - 19:07  (00:02)
user11  pts/6        office3   Mon Apr 28 18:58 - 19:03  (00:04)
user1   pts/5        dialup    Mon Apr 28 12:35 - 12:55  (00:20)
user12  pts/9        dialup    Mon Apr 28 12:21 - 01:01  (12:40)
user13  ftpd0001     dialup    Mon Apr 28 11:59 - 12:07  (00:07)
user14  ftpd0002     dialup    Sun Apr 27 21:17 - 21:27  (00:10)
user12  pts/5        dialup    Sun Apr 27 17:47 - 00:20  (06:33)
user15  pts/9        office4   Sun Apr 27 17:36 - 20:59  (03:22)
user16  pts/10       office5   Sun Apr 27 17:19 - crash (1+17:01)
user17  pts/1        dialup    Sun Apr 27 16:38 - 16:41  (00:02)
user14  pts/1        office6   Sun Apr 27 14:21 - 14:24  (00:02)
user14  ftpd0003     dialup    Sun Apr 27 09:18 - 09:19  (00:01)
user1   pts/1        dialup    Sat Apr 26 21:28 - 21:30  (00:01)
user18  ftpd0004     dialup    Sat Apr 26 19:39 - 19:49  (00:10)
user19  ftpd0005     dialup    Sat Apr 26 14:06 - 14:23  (00:16)
user20  pts/10       dialup    Sat Apr 26 13:29 - 22:26  (08:57)
user18  ftpd0006     dialup    Sat Apr 26 12:51 - 13:01  (00:09)
user18  ftpd0007     dialup    Sat Apr 26 12:42 - 12:52  (00:10)
user21  pts/1        dialup    Sat Apr 26 11:27 - 14:02  (02:34)
user22  pts/1        dialup    Sat Apr 26 08:42 - 10:43  (02:00)
user23  pts/1        dialup    Fri Apr 25 18:05 - 18:12  (00:07)
user18  ftpd0008     dialup    Fri Apr 25 13:10 - 13:20  (00:10)
user18  pts/1        dialup    Fri Apr 25 11:25 - 14:05  (02:39)
user18  ftpd0009     dialup    Fri Apr 25 11:10 - 11:20  (00:10)
user24  pts/1        dialup    Thu Apr 24 21:47 - 21:54  (00:07)
user25  ftpd0010     dialup    Thu Apr 24 21:24 - 21:29  (00:05)
user21  pts/1        dialup    Thu Apr 24 17:33 - 21:40  (04:06)
user26  pts/1        dialup    Thu Apr 24 16:15 - 16:20  (00:05)
user1   pts/1        dialup    Thu Apr 24 13:57 - 14:07  (00:09)
user23  pts/1        dialup    Thu Apr 24 07:01 - 08:19  (01:18)
user27  pts/1        dialup    Thu Apr 24 01:51 - 02:00  (00:08)
user28  ftpd0011     dialup    Thu Apr 24 01:02 - 01:02  (00:00)
user28  pts/1        dialup    Thu Apr 24 00:35 - 00:36  (00:00)
user28  pts/1        dialup    Thu Apr 24 00:17 - 00:21  (00:03)
user28  pts/1        dialup    Wed Apr 23 23:40 - 23:49  (00:09)
```

Figure 2.20: *Excerpt from data produced by the last command. User names, machine names, etc. were sanitized.*

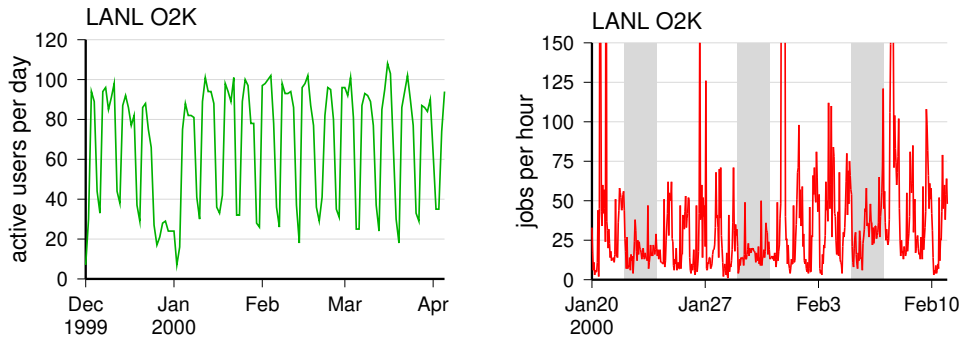


Figure 2.21: *The workload during holidays, weekends, and nights is less intense, and probably also statistically different from the workload during prime working hours. In the right-hand graph, weekends are shaded.*

into folders. While most of the folders represent genuine classification of messages by the users, many users also have an "all documents" folder that includes copies of all the messages in the other folders. These should, of course, be discounted when tabulating various metrics of email activity, such as the number of messages sent and received, how many were addressed to each recipient, and how they were classified into folders [407].

2.3.2 Multiclass Workloads

Even without errors and abnormal events, it is often necessary to filter the collected data. One example where filtering is needed occurs when the workload is actually composed of multiple classes of workloads that have been merged together (e.g., [717, 763, 249, 58]). If we want to model just one of these classes, we need to filter out the rest.

A straightforward example comes from hardware address tracing, which involves recording all the addresses that the CPU places on the memory bus. Although most of these are indeed memory addresses, some may be special addresses that are used to activate and control various devices. Thus if we are interested in memory behavior and caching, we should filter out the non-memory addresses (which are a different class of addresses).

A common type of multiclass workload is the combination of prime time and non-prime time workloads. Most users are active during normal work hours, say 9 to 5 on weekdays. This is when most of the workload is generated (Figure 2.21). During the night, on weekends, and on holidays, much less work is submitted. If we are interested in modeling the characteristics of prime time load, we should focus on prime time data, and filter out data that is generated at other times. Such filtering would also avoid administrative activity that is often performed at night or during weekends so as not to interfere with the system's intended work [129] (and see Figure 2.24).

As another example, consider the modeling of interactive user sessions. Data can be collected from a Unix server using the last command, which lists all user logins since

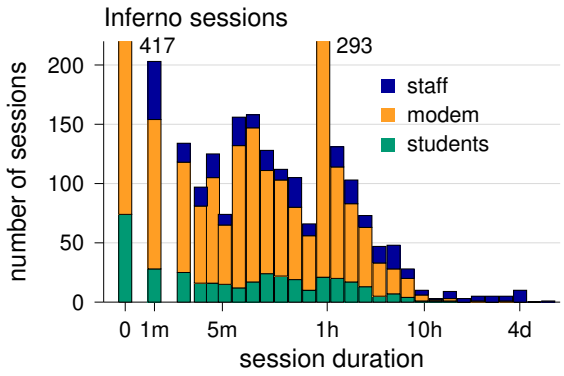


Figure 2.22: *The distribution of session durations for different classes of users. Only staff who have dedicated workstations generate sessions longer than 10 hours. (Data truncated at 220 sessions; there are more sessions of 0 or 1 hour.)*

the last reboot and the lengths of the ensuing sessions. Figure 2.22 shows the results of doing so on a shared server. The data contains many short and intermediate sessions, and a few extremely long ones. The longest sessions seem impossible, because they extend for several days. This mystery is solved by noting that the workload is actually composed of three classes: students working from student terminal labs, people accessing the server via remote dial-up networking, and staff working from their offices. The first two classes log off immediately when they complete their work and are responsible for the bulk of the data. This is the data that truly characterizes interactive sessions. Members of the third class, that of university staff, simply open windows to the server on their desktop machines, and leave them open for days on end. This data is not representative of interactive work, and should be filtered out.

Incidentally, it may also be the case that this dataset should be cleaned because it contains non-representative items. In particular, the large number of 0-length and 1-hour modem sessions may reflect connection problems and automatic termination of nonactive sessions, respectively, rather than the real lengths of user sessions. Notably, the interactive student sessions do not have these modes.

Multiclass workloads also occur in Internet traffic and web access. A well-known problem is the identification of the source using IP addresses. Many addresses indeed correspond to a specific computer and thereby identify a unique source. But if network address translation (NAT) is used, a single IP address may actually represent an entire network [461]. Likewise, if DHCP is used by the Internet service provider to assign IP addresses dynamically, such addresses may be reused to represent different clients [458]. Moreover, even a single computer may actually represent multiple users. Thus, from a workload analysis point of view, IP addresses come in two classes: those that represent a single user and those that represent many users.

Of course, not all multiclass workloads need to be cleaned: sometimes we are in fact interested in the complete workload, with all its complexities. For example, file-sharing

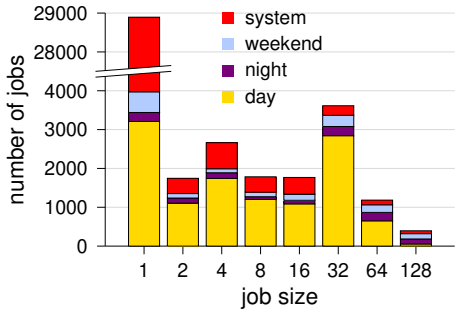


Figure 2.23: *Histogram of job sizes in the NASA Ames log, showing abnormal number of sequential system jobs.*

systems typically handle two very different types of files: audio clips and full-length movies [309]. Although it might be interesting to evaluate the performance of each type individually, this has to be done in the context of a system that supports both types. Likewise, Internet traffic includes both legitimate traffic and "background radiation" — the unproductive traffic generated by malicious activity (mainly automatic scanning of addresses and ports in an attempt to find vulnerable systems) and system misconfigurations [58]. Again, this traffic in itself is not very interesting, but an evaluation of service received by legitimate traffic must take it into account.

2.3.3 Anomalous Behavior and Robots

A special case of multiclass workloads is when one class is normal and another is anomalous [117]. Anomalous behavior is one that has a significant effect on the workload, but is not representative of general workloads. Observations reflecting such behavior should therefore be removed before starting the modeling. This is especially true if the source of the anomalous behavior is the system or its administrators, and not its users.

One rather extreme example of anomalous behavior was shown in the analysis of the NASA Ames iPSC/860 machine (repeated in Figure 2.23). Of the 42,264 jobs that appeared in that log, 24,025 were executions of the pwd Unix command on a single node. This anomaly was not due to users who forgot their working directory; rather, the system's administrators used these runs to verify that the system was up and responsive. In other words, a full 56.8% of the data in the log are actually bogus. But once identified, it is easy to remove.

Another example of a unique behavior is shown in Figure 2.24, which displays the daily arrival patterns at five large-scale parallel machines, averaged over the complete logs. While variations occur, all machines exhibit many more arrivals during work hours than during the night, as might be expected. The one exception is the SDSC Paragon, which also has very many arrivals between 3:30 and 4:00 AM — more than twice as much as the average during peak hours. Upon inspection, this seems to be the result of a set of 16 jobs that is run every day (Figure 2.25). The regularity in which these jobs

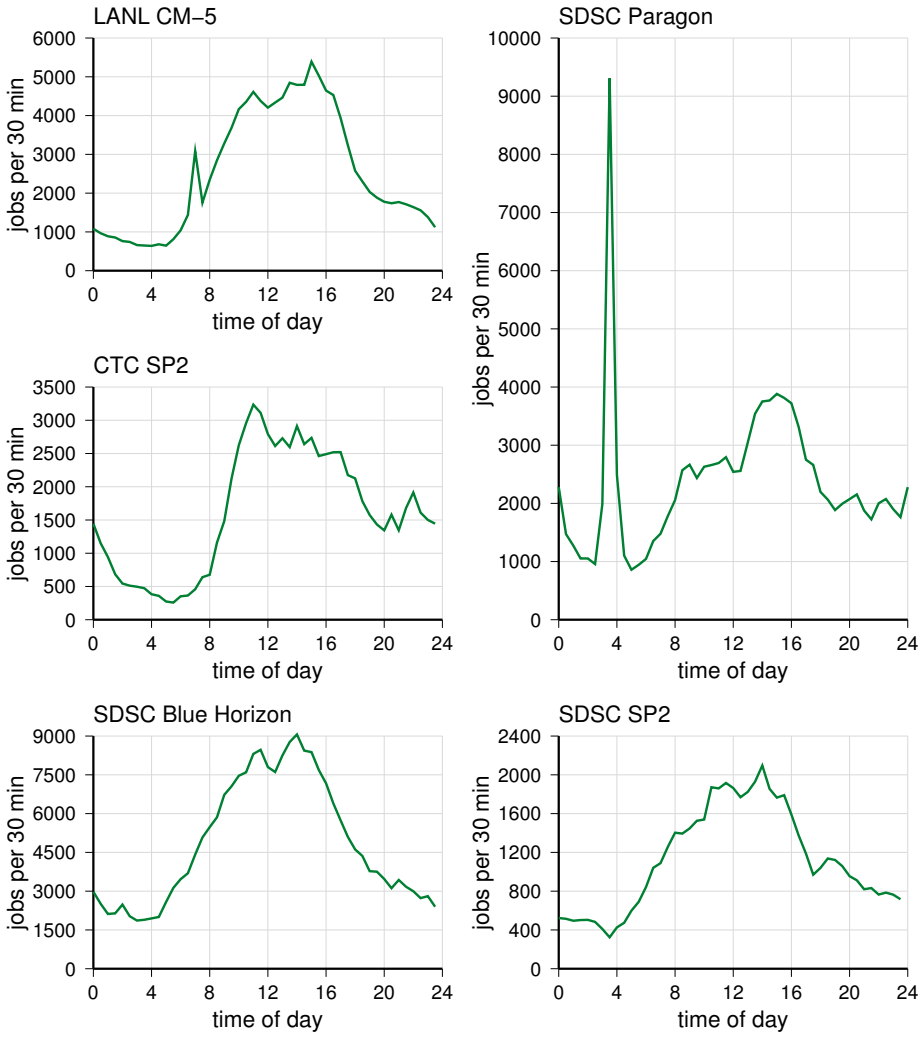


Figure 2.24: *Arrival patterns at five large-scale parallel machines. The large peak at 3:30 AM on the SDSC Paragon is an obvious anomaly.*

appear leads to the assumption that they are executed automatically at this time so as not to interfere with the work of real users. These jobs should therefore probably be removed from the log if normal user activity is sought.

A similar example was reported in an analysis of wireless network activity from Dartmouth College [414]. This showed an extreme spike of activity every Monday morning at 10 AM, that completely dominated wireless activity across the entire campus. This was traced to a course held at that time in the business school. Again, this spike — although real — was very atypical of normal usage, and should not be used to represent normal behavior.

The set of jobs that are executed at the same time every day is an example of a

```
user5    1    180 10/02/96 03:49:08
user5    1    248 10/02/96 03:49:29
user5    1    846 10/02/96 03:49:51
user5    1   3008 10/02/96 03:50:14
user5    1    715 10/02/96 03:50:37
user5    1     68 10/02/96 03:50:59
user5    1    455 10/02/96 03:51:22
user5    1    486 10/02/96 03:51:44
user5    1     50 10/02/96 03:52:05
user5    1     28 10/02/96 03:52:27
user5    1    101 10/02/96 03:52:50
user5    1   3524 10/02/96 03:53:13
user5    1     49 10/02/96 03:53:37
user5    1     81 10/02/96 03:54:00
user5    1    330 10/02/96 03:54:24
user5    1    826 10/02/96 03:54:57

user5    1     38 11/02/96 03:49:07
user12  64  22096 11/02/96 03:49:25
user5    1    252 11/02/96 03:49:29
user5    1    848 11/02/96 03:49:51
user2   16  22854 11/02/96 03:49:54
user5    1   3005 11/02/96 03:50:12
user5    1   1810 11/02/96 03:50:34
user5    1     66 11/02/96 03:50:56
user5    1    463 11/02/96 03:51:20
user5    1    639 11/02/96 03:51:43
user5    1     50 11/02/96 03:52:07
user5    1     26 11/02/96 03:52:32
user5    1    104 11/02/96 03:52:53
user5    1   3850 11/02/96 03:53:14
user5    1     48 11/02/96 03:53:35
user5    1     83 11/02/96 03:53:56
user5    1    336 11/02/96 03:54:18
user5    1   1157 11/02/96 03:54:44

user5    1     42 12/02/96 03:49:06
user5    1    311 12/02/96 03:49:27
user5    1   1076 12/02/96 03:49:48
user5    1   3037 12/02/96 03:50:09
user5    1   2603 12/02/96 03:50:33
user5    1     77 12/02/96 03:50:55
user5    1    468 12/02/96 03:51:18
user5    1    385 12/02/96 03:51:42
user5    1     59 12/02/96 03:52:05
user5    1     36 12/02/96 03:52:28
user5    1    116 12/02/96 03:52:51
user5    1   1733 12/02/96 03:53:12
user5    1     51 12/02/96 03:53:32
user5    1     88 12/02/96 03:53:54
user5    1    338 12/02/96 03:54:15
user5    1   1939 12/02/96 03:54:39
```

Figure 2.25: *Three excerpts of data from the SDSC Paragon log.*

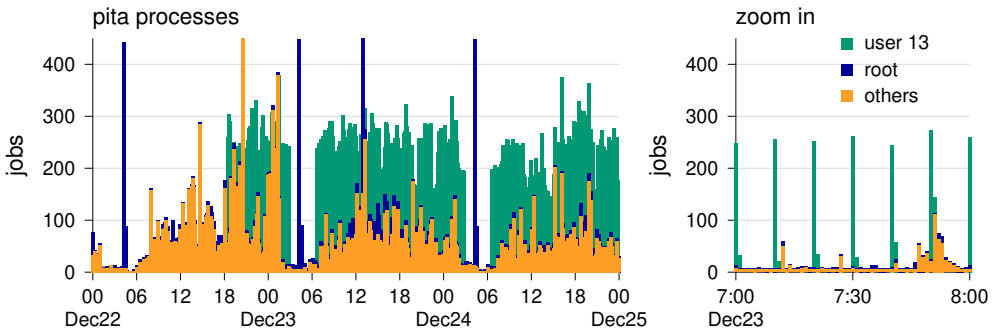


Figure 2.26: *Robots in a Unix server workload.*

robot: software that creates work automatically. A much more extreme example is the following one, from a Unix server (Figure 2.26). This workload is actually composed of three classes:

1. A set of more than 400 jobs submitted automatically by a root script each morning at 04:15 AM. These are all Unix `rm` commands, most probably issued by a maintenance script used to clean up temporary disk space.
2. Massive activity by user 13, starting in the evening hours of December 22. Upon closer inspection, this was found to consist of iterations among a half-dozen jobs, totaling about 200, that are submitted every 10 minutes. In subsequent days, this pattern started each day at 6:40 AM and continued until 2:50 AM the next day. Again, such a regular pattern is obviously generated by a script.
3. Normal behavior by other users, with its fluctuations and daily cycle.

With the advances made in using software agents, robot workloads are becoming prevalent in many areas. Examples of robot activity that may distort workload statistics include the following:

- Spam email, where many messages are repeated and sender addresses are spoofed to prevent the identification of the true sender.
- Spiders that crawl the web to create indexes for web search engines. Such crawls tend to be much more uniform than human browsing activity, which displays locality and limited attention [180].
- Web monitors that repeatedly ping a site to verify its accessibility from different locations. For example, one e-commerce site conducted a usage survey and were pleased to find significant activity at night, which was thought to represent non-U.S. clients. But on closer inspection this activity turned out to be largely due to a web monitoring service that they themselves had enlisted and forgotten to factor out [579].

- Repeated searches for the same item, possibly intended to make it seem more important and rank high in the "most popular search" listings. For example, a log of activity on the AltaVista search engine from 8 September 2002 included one source that submitted 22,580 queries, of which 16,502 were for "britney spears", and another 868 for "sony dvd player". Interestingly, another 4,305 were for "dogs" and 615 for "halibut". Many of these queries came in interleaved sequences, where the queries of each subsequence come at intervals of exactly five minutes.
- Computer viruses, worms, and denial-of-service attacks, which create unique Internet traffic patterns. It should be noted, though, that this sort of activity is so common on Internet backbone links that it should probably not be considered to be an anomaly [82].

In fact, the very fact that robots typically exhibit unique behavioral patterns can be used to identify them. For example, it has been suggested that robots visiting web search engines may be identified as those users that submit more than 100 queries a day [375, 372]. However, it is questionable whether a single sharp threshold is suitable for all datasets. Note, also, that in some cases robots try to disguise themselves as humans, and therefore explicitly try to mimic human behavior.

2.3.4 Workload Flurries and Flash Crowds

A more difficult situation is depicted in Figure 2.27 [249], which shows patterns of activity across the whole log for four large-scale parallel machines. In the two years of activity on the LANL CM-5 parallel supercomputer, for example, there were three bursts of activity that were 5–10 times higher than normal and lasted for several weeks. Each of these bursts can be attributed to one or two specific users[2]. It is thus relatively easy to filter out this anomalous behavior and to retain only the normal behavior [249, 696]. However, it is hard to justify an outright cancellation of these three peaks of activity. After all, they were created by three different users at different times, and account for a non-negligible fraction of the total workload. Moreover, similar flurries of activity occur in other logs. Therefore the phenomenon itself seems to be recurring, even if individual flurries are all different from each other.

The problem is that such flurries of activity by individual users can have a significant effect on the overall workload statistics. By definition, the existence of a flurry implies the submittal of a very large number of jobs over a relatively short span of time. This causes their interarrival times to be shorter than those observed under normal conditions. Because the flurries may account for a sizable fraction of the total workload, they affect the observed distribution of interarrival times for the entire log (Figure 2.28). The effect is especially large in the LANL log, where the flurries account for a large fraction of the activity, but is negligible in the CTC log, where there is only one rather small flurry.

[2] It is actually not completely clear in this case whether the first two of these are genuine users. One is root and the other may be Thinking Machines personnel.

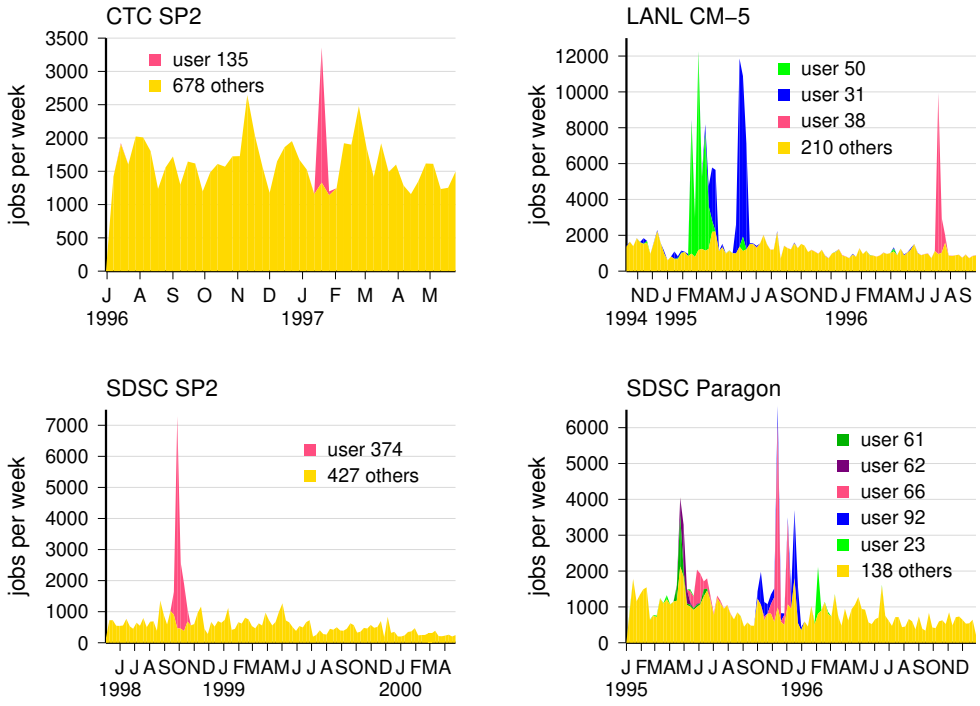


Figure 2.27: *Arrivals per week in long logs. Large flurries of activity can typically be attributed to one or two users.*

Flurries also affect other workload attributes, because the jobs in a flurry tend to be much more homogeneous than those in the entire workload. An example is given in Figure 2.29, focusing again on the LANL CM-5. This shows the effect of flurries on the distributions of four different workload attributes: the interarrival times, the job runtimes, the job sizes, and the average memory usage per node. In all the cases, the distributions with the flurries tend to have modes that reflect the jobs that compose the flurry itself.

More importantly, it is clear from the figure that the differences between the workload statistics observed in 1995 and those in 1996 stem from the flurries. When the flurries are removed, the distributions for the two years are rather close to each other. It is the flurries, which are each unique, that cause the workloads in the two years to look different. In other words, the workload is actually composed of two components: the "normal" workload, which is rather stable and representative, and the flurries, which are anomalous and unrepresentative.

Although flurries as identified here are limited in time, long-term abnormal activity by users may also occur. An example is shown in Figure 2.30. In this case, a single user completely dominates the machine's use for a whole year, and again for several months two years later. All told, this user's activity accounts for 57% of all the activity recorded in the log. In addition, as shown in the scatterplot, the combinations of job size and

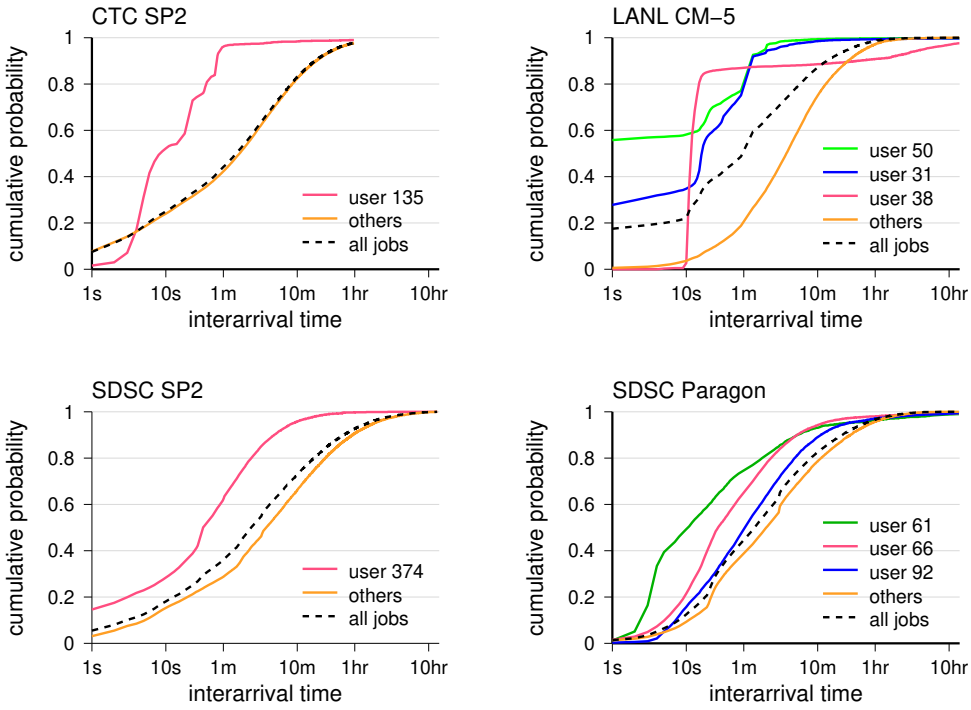


Figure 2.28: *Bursts of activity by hyper-active users bias the distribution of interarrival times.*

runtime that characterize this work are distinct and not like the combinations exhibited by other users.

Abnormal workloads are not limited to the workloads on parallel supercomputers such as those shown in these examples. Figure 2.31 shows five years worth of data from access point B of the WIDE backbone (a transpacific link). On each day, a sample of 15 minutes was taken at 2 PM. Analyzing the protocols used by the observed packets shows considerable variability in protocol usage. Some of this variability reflects changing applications, such as the use of the Napster file-sharing service in 2001–2002. There is also a prominent example of a worm attack in late 2003, when the Welchia worm used ICMP packets to scan for hosts. But there are also many unexplained short-lived anomalies, in which there is a surge of using UDP or some other protocol that is not one of those that are commonly observed.

Another well-known example comes from web workloads, where surges of activity called *flash crowds* may occur. The trigger is usually some external event that leads to a convergence of traffic on an unsuspecting server. In many cases these are breaking news stories, but scheduled events such as sporting events can also lead to similar traffic patterns. The characteristics of flash crowds are discussed in Section 9.4.3.

A possible way to handle phenomena like flurries and flash crowds is not to erase the anomalous behavior, but rather to model it separately. In other words, the log can be

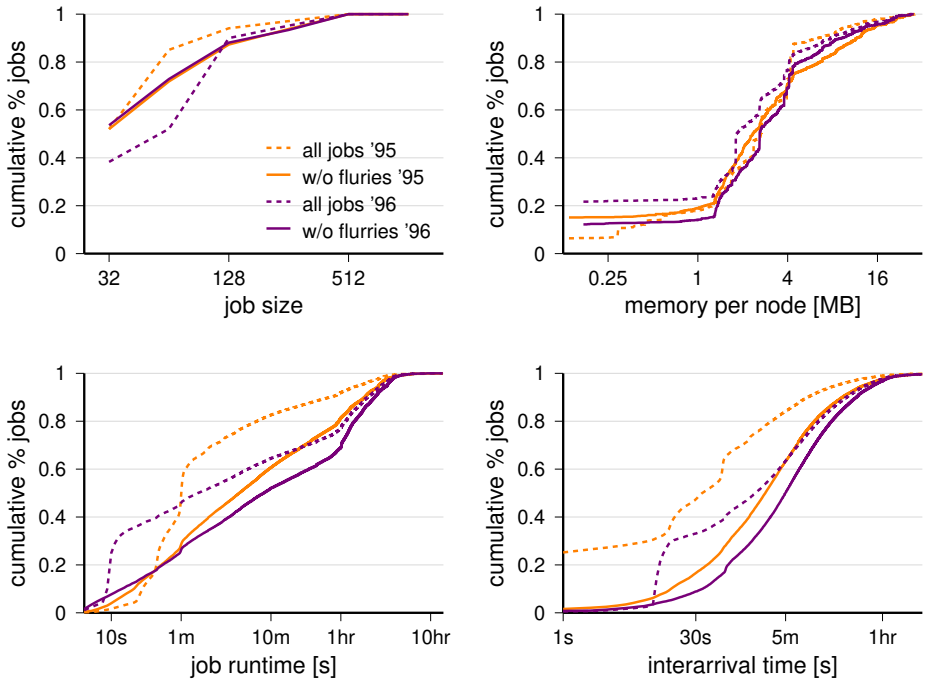


Figure 2.29: *Flurries of activity have a strong effect on workload statistics. On the LANL CM-5 the main difference between the 1995 and 1996 portions of the log is due to flurries.*

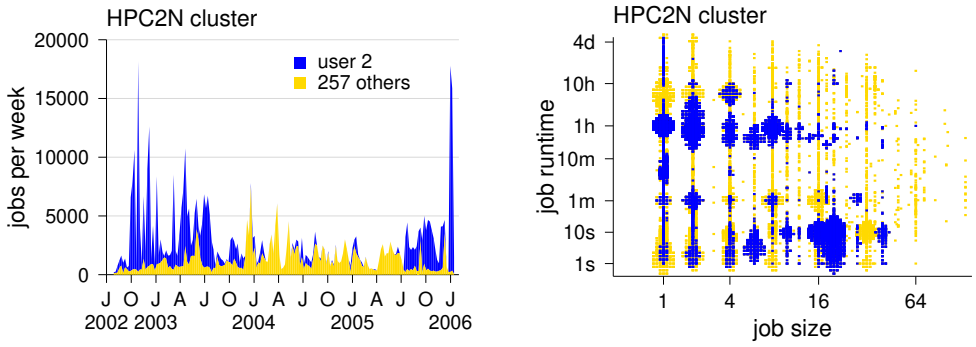


Figure 2.30: *Abnormal activity by a single user that accounts for 57% of the entire log.*

partitioned into two disjoint classes: the normal workload and the anomalous workload. Each of these is then modeled in isolation [249, 696, 76]. The models can then be used separately or combined, according to need. Flurries, in particular, can be integrated into user-based workload models, as discussed in Chapter 8.

It should be noted that flurries can be either intentional or the result of a bug (Figure

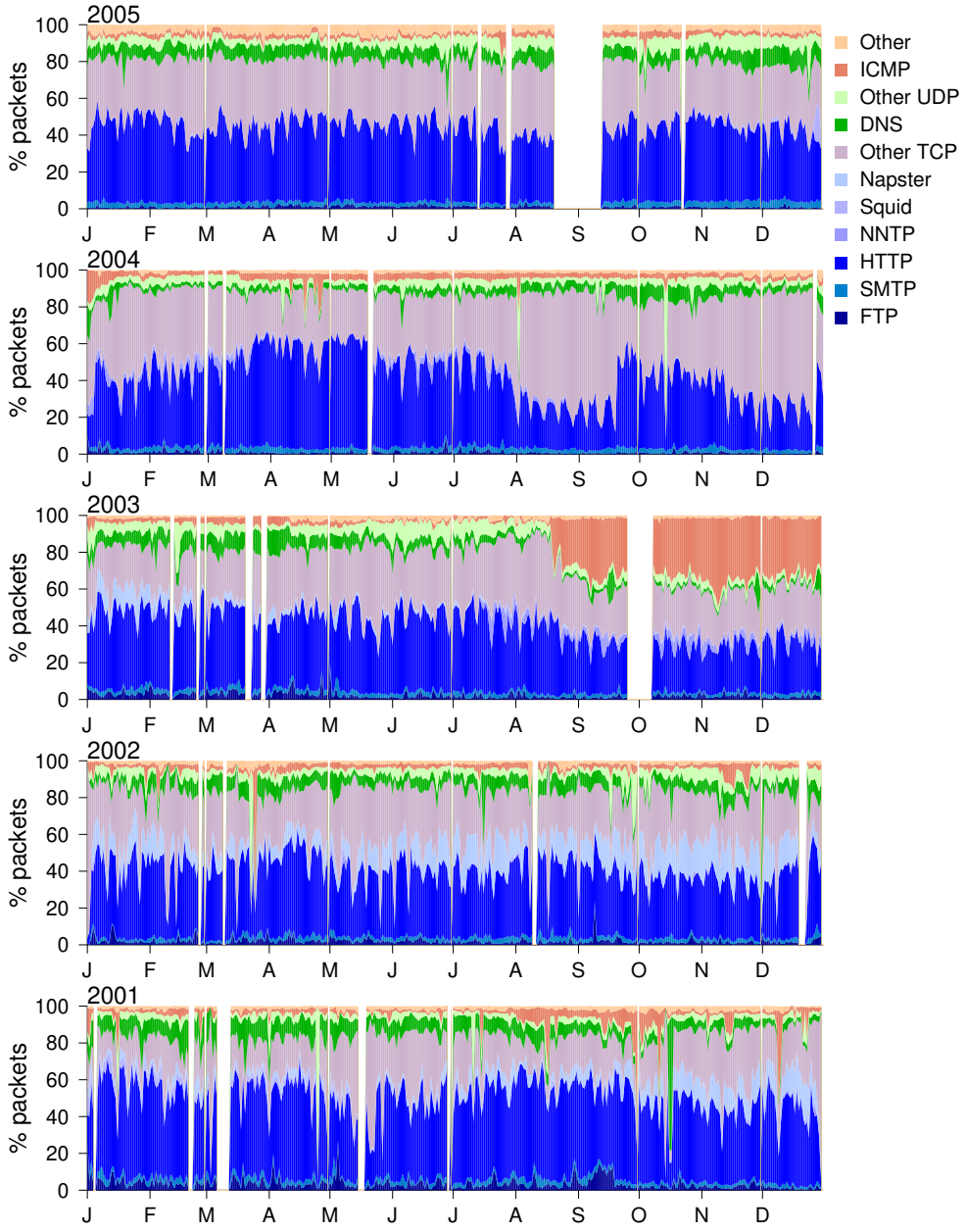


Figure 2.31: *Five years of Internet traffic data at daily resolution shows considerable variability and many unique events.*

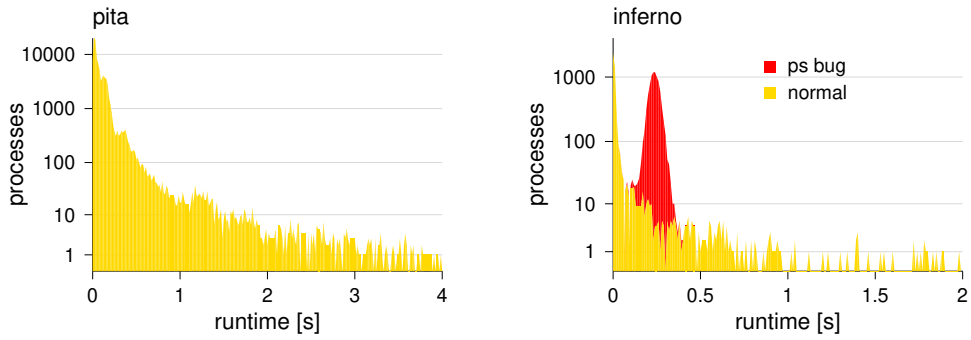


Figure 2.32: *The inferno dataset of Unix processes includes a huge number of processes that run the Unix* ps *command, the result of a bug in implementing an operating systems course exercise. The pita dataset is more typical.*

2.32). But separate modeling of flurries is suitable in either case. If buggy flurries are a common occurrence, they should be included in workload models. But they should probably not be included in the mainstream workload model for which the system is optimized.

Separate modeling is attractive because it can also be used for other situations, e.g. attacks [96]. Given a log of activity on a server, large outbreaks may indicate automated attacks. Identifying them and separating them from the general workload model then achieves two objectives: it prevents optimization of the server to handle attack loads, and it enables the development of tools that will identify and counteract attacks [665].

2.3.5 Identifying Noise and Anomalies

Given a data log, we would obviously want to be able to identify abnormal situations such as those described above. All these examples were identified by hand (or, rather, by eye): the data was presented graphically in different ways, and strange phenomena became apparent. Once identified they could be removed by writing a script to filter out the undesired data.

But can this identification be done automatically? It may be possible to devise rules to identify known forms of deviations in specific circumstances. For example, the following criteria may be used to identify robots in web server logs [180, 281, 658]:

- Well-behaved robots often identify themselves in the data they provide for the user agent field of the log (for real users, this field identifies the browser being used).
- Robots may also implicitly identify themselves by retrieving the file robots.txt, which specifies the rules of conduct that robots are requested to follow on this site. In addition, robots tend not to provide data in the referrer field.
- Web robots, especially those that crawl the web in the service of search engines, are typically interested mainly in text. They tend to avoid downloading embed-

ded images, because such images consume significant bandwidth and are hard to classify.

- Robots may also have unique temporal activity patterns. On the one hand, they may issue requests at a rapid rate that would be impossible for a human user. On the other hand, they may spread out their activity and access the site at, say, precise intervals of five minutes in order to reduce their impact.

A more general procedure that was advocated by Cirne and Berman is to use clustering as a means to distinguish between "normal" and "abnormal" data [137]. The methodology is to partition the workload log into days, and then to characterize each day by a vector of length n (specifically, this was applied to the modeling of the daily arrival cycle, and the vector contained the coefficients of a polynomial describing it). These vectors are then clustered into two clusters in R^n. If the clustering procedure distinguishes a single day and puts it in a cluster by itself, this day is removed, and the procedure is repeated with the data that is left. Note, however, that this specific approach has its risks: first, abnormal behavior may span more than a single day, as the above examples show; moreover, removing days may taint other data (e.g., when interarrival times are considered). Still, clustering may in general be useful for separating out rare abnormal data.

Another technique that has been proposed for removing anomalies from Internet traffic data (or, rather, for negating their effect on evaluation results) is the following [176]. First, hash the data into a number of groups based on some select workload attribute, for example a flow's source IP address or destination port. Take care that all the packets of each flow are mapped together. Now analyze each of these groups independently as you would analyze the full data. The crux of the method is to then use the median of the results derived for the different groups as representing the whole data. Assuming that anomalies map to less than half of the groups, the median results will indeed represent clean data with no anomalies. The number of groups to use depends on the workload length and the number of anomalies that may be expected. For short traces of Internet traffic, a small number such as eight groups suffices.

One point that deserves to be emphasized is that outliers that are simply much larger than other values are *not* necessarily anomalous. One sometimes sees data analyses in which an upper bound on "good" data is postulated, and all higher values are discarded. This may be justified only if the data is known to come from an appropriate distribution. For example, if the data is known to come from a normal distribution, where the tails decay exponentially, values that are several standard deviations from the mean are not expected to appear under realistic conditions [686]. More generally, the bound may be set based on Chauvenet's criterion, which is to remove those samples whose probability is less than $\frac{1}{2n}$ (where n is the number of samples).

A more sophisticated statistical consideration is to single out values that are both far from the mean and far from their neighbors [119]. Thus a sample X_i is characterized by

$$Y_i = |(X_{i+1} - X_i)(X_i - \bar{X})|$$

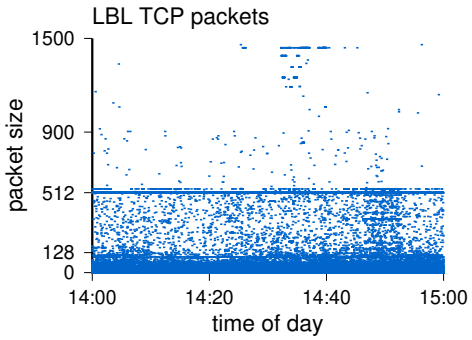


Figure 2.33: *The sizes of TCP packets on LBL's Internet connection in 1994 exhibit a strong concentration in the range of 0–512 bytes, with very few larger packets of up to 1460 bytes.*

The Y_i values are then normalized by dividing by their standard deviation. Values that are larger than a threshold depending on the number of samples are flagged as suspect.

In general, however, it is very dangerous to remove outliers based on such statistical considerations. One reason is that in many cases these high values are very significant — especially in positive and highly skewed distributions, which are common in computer system workloads. We discuss this issue at length in Chapter 5, which deals with heavy tails. But similar problems may also occur in distributions with a very limited support. For example, Figure 2.33 shows the sizes of 677,846 TCP packets on LBL's Internet link during one hour on 28 January 1994. The vast majority of the packets are up to 512 bytes long, but some reach a maximal size of 1460 bytes. The average packet size is 138.86 and the standard deviation 219.90, so maximally sized packets are a whopping six standard deviations from the mean; they are even farther away from the median, which is only 2 bytes. But removing these larger packets on such statistical grounds is obviously unjustified.

The sad truth is that outliers can take many different forms, and we do not have good mechanisms to reliably detect and classify them. A possible conceptual definition of outliers is that they are data points that are anomalous and therefore an uninteresting distraction in a given modeling context, but may potentially have a significant effect on the derived model. But in computer workloads, the anomalies that can change the model are often very interesting and important for the analyst, and therefore should not be regarded as noise [31, 117]. A careful manual inspection of the data, based on domain-specific understanding, is still the best approach.

2.4 Educated Guessing

Using available logs or the active collection of information can only be done if the target systems actually exist. But what if we want to characterize the workload on the En-

Type	*Examples*	*Good guess*	*Bad guess*
location	access to memory addresses access to web servers	spatial locality temporal locality	uniform access
size	runtimes of processes sizes of files	heavy-tailed modal	exponential uniform
arrivals	requests arrive at server	daily cycle self-similar	Poisson process
popularity	pages delivered by server	Zipf distribution	uniform

Table 2.2: *Summary of guidelines for educated guessing. The meanings of the different guesses are discussed at length in subsequent chapters.*

terprise's "beam me up" mechanism? Although this specific device may still be some distance from implementation, the rapid progress of technology continuously leads to situations in which we need to evaluate systems that do not have working counterparts. For example, the Internet protocols were developed long before any network carried significant traffic, let alone the vast amounts of data that the Internet carries today. Development of services for mobile devices faces the same uncertainty, not to mention cases where the devices themselves are completely new (e.g., [183]).

We therefore need to make assumptions about the workloads that systems will encounter without the benefit of data from similar systems [55]. One common approach is to base the assumptions on mathematical convenience — assume a workload that is easy to analyze. However, with the accumulation of knowledge about different types of workloads, we can start to identify patterns that seem to be repeated in many different contexts. These are general invariants that characterize not only different workloads of the same type, but also different types of workloads. It is therefore relatively safe to assume that these patterns will also appear in the workloads of future systems.

The best guess depends on the type of feature that is being considered. One type of feature is related to locations that are visited one after the other — memory addresses, web servers, and so on. It is well known that such sequences most often display locality of reference, a pattern that was identified in the earliest computers and that lies at the basis of all forms of caching. It is a good guess that future workloads will also display locality. At the very least, departure from this practice should be justified. We will discuss locality in Section 6.2.

Another type of feature relates to size. For example, process sizes are measured by the seconds of runtime and file sizes are measured by bytes of disk space. In many cases, real-life distributions turn out to be skewed: there are many small items and a few big ones. Moreover, the big ones are sometimes VERY BIG. Distributions that include such large values are called heavy-tailed and are discussed in Chapter 5. Note that the typical distribution observed is continuous, and one does not see a clear distinction between the small and large items. In other words, the distribution is not bimodal.

A major issue in dynamic workloads is the arrival process. A very common as-

sumption is that work arrives uniformly over time, in a Poisson process. This has the memoryless property, and is very convenient for analysis (these concepts are explained in Section 3.2.1). However, in many cases it is not justified. The arrivals at many systems turn out to be self-similar, as described in Chapter 7. Nevertheless, when load is high and results from the interleaving of multiple sources, arrivals may appear independent as in a Poisson process [105].

Finally, it is sometimes necessary to make assumptions about the relative popularity of different items, such as web servers or pages within a single server. The distribution of popularity turns out to also be very skewed, and often follows a Zipf distribution.

The above guidelines for educated guessing are summarized in Table 2.2. While they are not universal truths, they deserve to be considered carefully when other information is not available.

Note that guessing patterns based on past experience is better than just wild guessing, but it is nevertheless still guessing. It is therefore important to accompany guessing with sensitivity checking. This means that we need to check the sensitivity of performance evaluation results against the details of the guess. This is done by modifying the guess and checking the effect of the modification on the results. If the effect is large, we need to be more careful with our guessing, or perhaps admit that we cannot provide a single conclusive answer.

2.5 Sharing Data

Data acquisition is a major problem for workload modeling. Some data exists but is not known or accessible to those who need it. In other cases data needs to be collected explicitly, which poses both technical and administrative challenges. Identifying anomalies that should be removed is often a fortuitous affair.

The conclusion is that data should be made available and shared. Doing so will promote the construction and use of good models, as well as the accumulation of information. The easiest way to do so is to contribute data to an existing site [534, 259]. As an alternative you can maintain your own archive, but that should be done only if there is no established archive for your data, and you are willing to commit to maintain it for many years to come.

Importantly, sharing the raw data is not enough. It is equally important to share metadata, meaning information about the shared data. This includes both context information (from what system was the data captured? when was it captured?) and experience with using it. For example, it is important to share information about problems in the data, and any data cleaning that was performed [250].

Existing workload websites include the following. Most of them were set up to share data collected and owned by the site managers. Some, like the Parallel Workloads Archive, the Internet Traffic Archive (which regrettably seems to be dormant), and DatCat are intended to serve as a repository for data from multiple sources. A recurring problem is that the sites do not survive past a single funding cycle or a change in in-

terest of their initiators. As a community, we do not yet have the culture of supporting centralized data repositories and maintaining them indefinitely.

Active — Sites that seem to continue to include new data last I checked

- The Parallel Workloads Archive, containing accounting logs from large-scale parallel supercomputers since 1993:
  URL http://www.cs.huji.ac.il/labs/parallel/workload/
- The SNIA IOTTA repository, with extensive I/O traces:
  URL http://iotta.snia.org/
- The Cooperative Association for Internet Data Analysis (CAIDA) site:
  URL http://www.caida.org/
- RIPE Network Coordination Center Data Repository with Internet routing and traffic data:
  URL https://labs.ripe.net/datarepository/
- The MAWI backbone traffic archive with packet traces from transpacific links:
  URL http://mawi.wide.ad.jp/mawi/
- The CRAWDAD archive of wireless data:
  URL http://crawdad.org/
- Wikipedia hourly data on page views since December 2007:
  URL http://dumps.wikimedia.org/other/pagecounts-raw/

Accessible — Sites from which data can be accessed, but there has been no activity for some years

- DatCat, the Internet data measurement catalog, which includes some of the other data sources listed here (e.g. the CAIDA and CRAWDAD datasets) and also some others:
  URL http://www.datcat.org/
- The Internet Traffic Archive.:
  URL http://www.acm.org/sigcomm/ITA/
- The Grid Workloads Archive, containing accounting logs from large-scale grid systems:
  URL http://gwa.ewi.tudelft.nl/pmwiki/
- The LBNL/ICSI Enterprise Tracing Project, with Intranet rather than Internet traffic data:
  URL http://www.icir.org/enterprise-tracing/
- Video Frame Size Traces:
  URL http://www-tkn.ee.tu-berlin.de/research/trace/trace.html
- Email corpus from Enron corporation:
  URL http://www.cs.cmu.edu/%7Eenron/

Gone — Sites that used to exist but seem to be gone last I checked

- The Tracefile Testbed, with logs of MPI calls and other events in parallel applications:
  URL http://www.nacse.org/perfdb/
- The BYU Performance Evaluation Laboratory Trace Collection Center, with address, instruction, and disk I/O traces:
  URL http://traces.byu.edu/
- The New Mexico State University Trace Database, with address traces for processor architecture studies:
  URL http://tracebase.nmsu.edu/tracebase.html
- The NLANR Internet traces:
  URL http://moat.nlanr.net/

2.5.1 Data Formats

When preparing data, an important consideration is the format. Based on experience with the Parallel Workloads Archive [534], the following suggestions can be made.

- When collecting data, it is best to use normal signed integers and double-precision floats. The saved bits of unsigned values and lower precision are not worth the trouble of subsequent conversion errors and irretrievably lost resolution.
- In the interest of portability, it is best to store the data in plain ASCII files, rather than machine-specific binary formats or application-specific database formats. A possible exception is when huge data volumes are being recorded [27]; however, one needs to consider whether such large volumes are indeed needed, and in any case compression can be used.
- Strive for uniformity, e.g. expressing the data for each item as a fixed set of space-separated (or comma-separated) fields. If a standard format does not exist for your domain, define one and stick to it. De facto standard formats may exist if the data is collected by a standard tool, such as tcpdump. When designing a format, consider the following:
  - Different sources may produce different datasets, so you need to strike a balance between using the intersection of all sources and their union.
  - Some data items are "core" and universally useful, whereas others are only needed for specialized tasks.
  - Defining a format with lots of fields risks bloated data files that are devoted mainly to saying "no data for this field".
  - Whatever data you decide to discard will be lost forever.
  - The format should be readable by humans and easy to parse by computer.
- The simplest structure is to use uniform records in which each record (a row in the file) contains all the fields. A file with such rows is equivalent to a single table from a relational database. But in some cases it may make sense to use

multiple files with different structures connected by specific key fields — similar to having multiple tables in a database. One example is to have one table with data about jobs, and another with data about the infrastructure (especially when the configuration changed, parts were down, etc.). This complementary data may be very important when analyzing the jobs' data. Another example is having one table with the general data about each job (submit time, runtime, etc.), and another about the specifics of allocated resources. The key tying the two tables to each other is the job ID. Each job has one entry in the first table, but multiple entries in the second, allowing for a listing of all the different resources it used.

- Three examples of easy to parse formats are the following:
  - Fields separated by whitespace. This has the advantage of also being easy to look at for human users, especially if the fields are aligned. A potential problem occurs if a field is a list of items; do *not* group them by surrounding the field with quotation marks as in `"item1 item2 item3"`, because this will be impossible to parse by splitting on whitespace. It is therefore better to concatenate them with commas (and no spaces!) as in item1,item2,item3. However, this does not work for multiword strings.
  - Separate the fields with commas, leading to the so-called "comma-separated values" format (CSV, sometimes denoted by a .csv suffix). This supports fields that are multiword strings with spaces, but is less convenient for human users.
  - Separate fields by tabs and allow spaces within a field. The problem with this option is that it may be hard for humans to spot field boundaries.

- Another possible format is XML. This has the advantage of being a standard format, so many software packages that parse XML are available. In addition, it supports nested fields and attributes. However, it tends to create bloated files that are inconvenient for humans to read. Alternatives that are more human-friendly are JSON and YAML. These too have many packages that generate and parse them.

  Note that the XML standard format pertains only to the way things are written, not to the actual structure. For example, a user's name may be expressed by a simple tag such as

  ```
  <username uid="13">John Smith</username>
  ```

  or by a more complicated nested structure like

  ```
  <user>
    <firstname>John</firstname>
    <lastname>Smith</lastname>
    <userid>13</userid>
  </user>
  ```

This is an application-specific choice, and not defined by the standard. Thus applications that use the data must know in advance the intended semantics and the set of tags and attributes that are used.

- Use the appropriate resolution for each field. For example, always using two decimal places for numerical data is wrong, because fields with integral values (e.g. number of processors) will always have a meaningless suffix of ".00", whereas fields with very small values might lose all their data or suffer from rounding problems. Recall also that computers work in binary. If you see data fields with decimal values that all end with 5 or 25 or 75, this most probably indicates that too few bits are being used. In a related vein, appropriate units should be used. For example, consider whether to record memory usage in bytes, kilobytes, megabytes, or gigabytes.
- Pay special attention to timestamps. For jobs and other items related to user activity a resolution of seconds is sufficient. For networking and architecture a resolution of microseconds or even nanoseconds may be needed. Two common formats for date and time are
  - Seconds since some specific point in time, which is convenient for calculating the difference between timestamps. One common choice is to start counting at the moment that recording of the log started, so the initial timestamp is 0. Another is to use Unix time, meaning seconds since the epoch of 00:00:00 on 1 January 1970 in Greenwich. Each second can then be labeled with a date and time according to its local time zone, which should also be noted in the log. Unix time is typically considered to be a 32-bit signed value, so it will overflow in January 2038. However, it is expected that by that time 64-bit values will be used.
  - Wallclock date and time at the location where the log was recorded, which is more convenient for human consumption. This should be accompanied by a time-zone offset (i.e., the difference from UTC for this timestamp) to enable reliable calculation of differences between timestamps. The offset can change at different times of the year, depending on whether daylight savings time is in effect or not.
- Strive for consistent semantics. A bad example comes from the AOL search logs, where entries denote either queries or clicks on results. The format first identifies the query using a source, timestamp, and query string. In case of a click, the result rank and clicked URL are added. The unfortunate result is that clicks are associated with the timestamp of when the query was submitted, rather than with a timestamp of when the click occurred.
- Files should be self-documenting. Each file should start with header comments that specify where the data comes from and the time frame over which it was collected. Again, the format of the core header comments should be standardized.
- Files should have unique identifiers or names. These IDs should encode the source

and time frame in an abbreviated form. Version numbers should be used if the same dataset has several versions (e.g., a raw version and a cleaned version).

To illustrate these recommendations, Figure 2.34 contains an excerpt from the beginning of a workload file from the Parallel Workloads Archive. This is in the archive's "standard workload format" (SWF) [118, 534]. Comment lines start with a ";". Each job is described by a single line with the following 18 space-separated fields:

1. Job number: a counter field, starting from 1.
2. Submit time in seconds, relative to the start of the log.
3. Wait time in the queue in seconds.
4. Runtime (wallclock) in seconds. "Wait time" and "runtime" are used instead of the equivalent "start time" and "end time" because they are directly attributable to the scheduler and application, and are also suitable for models where only the runtime is relevant.
5. Number of allocated processors.
6. Average CPU time used per processor, both user and system, in seconds.
7. Average memory used per node in kilobytes.
8. Requested number of processors.
9. Requested runtime (or CPU time).
10. Requested memory (again kilobytes per processor).
11. Status: 1 if the job was completed, 0 if it failed, and 5 if canceled.
12. User ID: a number, between 1 and the number of different users.
13. Group ID: a number, between 1 and the number of different groups.
14. Executable (application): a number, between 1 and the number of different applications appearing in the log.
15. Queue: a number, between 1 and the number of different queues in the system.
16. Partition: a number, between 1 and the number of different partitions in the system.
17. Preceding job number, used in case this job depends on the termination of a previous job.
18. Think time from preceding job.

2.5.2 Data Volume

An important consequence of the selected format is its effect on the volume of data that needs to be stored. For example, XML is notorious for data bloat, because each datum is surrounded by tags specifying its meaning. Thus a parallel job using 32 PEs (processing elements) for 20 minutes may be represented by the record

```
; SWFversion: 2
; Computer: Intel iPSC/860
; Installation: NASA Ames Research Center
; Acknowledge: Bill Nitzberg
; Information: http://www.nas.nasa.gov/
;              http://www.cs.huji.ac.il/labs/parallel/workload/
; Conversion: Dror Feitelson (feit@cs.huji.ac.il) Nov 12 1999
; MaxJobs: 42264
; MaxRecords: 42264
; UnixStartTime: 749433603
; TimeZoneString: US/Pacific
; StartTime: Fri Oct  1 00:00:03 PST 1993
; EndTime: Fri Dec 31 23:03:45 PST 1993
; MaxNodes: 128
; Queues: queue=1 just means batch, while queue=0 is a direct job
; Note: There is no information on wait times - the given submit
;       times are actually start times
; Note: group 1 is normal users
;       group 2 is system personnel
;
    1       0 0   1451 128 -1 -1 -1 -1 -1 -1   1 1   -1 1 -1 -1 -1
    2    1460 0   3726 128 -1 -1 -1 -1 -1 -1   1 1   -1 1 -1 -1 -1
    3    5198 0   1067 128 -1 -1 -1 -1 -1 -1   1 1   -1 1 -1 -1 -1
    4    6269 0  10927 128 -1 -1 -1 -1 -1 -1   2 1   -1 1 -1 -1 -1
    5   17201 0   2927 128 -1 -1 -1 -1 -1 -1   1 1   -1 1 -1 -1 -1
    6   20205 0      3   1 -1 -1 -1 -1 -1 -1   3 2    1 0 -1 -1 -1
    7   20582 0      3   1 -1 -1 -1 -1 -1 -1   3 2    1 0 -1 -1 -1
    8   20654 0      8   1 -1 -1 -1 -1 -1 -1   3 2    1 0 -1 -1 -1
    9   20996 0     17   1 -1 -1 -1 -1 -1 -1   3 2    1 0 -1 -1 -1
   10   21014 0      2   1 -1 -1 -1 -1 -1 -1   3 2    1 0 -1 -1 -1
   11   21043 0     19   1 -1 -1 -1 -1 -1 -1   3 2    1 0 -1 -1 -1
   12   21097 0     20   1 -1 -1 -1 -1 -1 -1   3 2    1 0 -1 -1 -1
   13   21142 0     14   1 -1 -1 -1 -1 -1 -1   3 2    1 0 -1 -1 -1
   14   21206 0      2   1 -1 -1 -1 -1 -1 -1   3 2    1 0 -1 -1 -1
   15   21360 0     14   1 -1 -1 -1 -1 -1 -1   3 2    1 0 -1 -1 -1
   16   21405 0     15   1 -1 -1 -1 -1 -1 -1   3 2    1 0 -1 -1 -1
   17   21449 0     16   1 -1 -1 -1 -1 -1 -1   3 2    1 0 -1 -1 -1
   18   21496 0     15   1 -1 -1 -1 -1 -1 -1   3 2    1 0 -1 -1 -1
   19   21568 0      2   1 -1 -1 -1 -1 -1 -1   3 2    1 0 -1 -1 -1
   20   21655 0      5   1 -1 -1 -1 -1 -1 -1   3 2    1 0 -1 -1 -1
   21   22008 0      3   1 -1 -1 -1 -1 -1 -1   3 2    1 0 -1 -1 -1
   22   22083 0      2   1 -1 -1 -1 -1 -1 -1   3 2    1 0 -1 -1 -1
   23   22418 0     16   1 -1 -1 -1 -1 -1 -1   3 2    1 0 -1 -1 -1
   24   22463 0      3   1 -1 -1 -1 -1 -1 -1   3 2    1 0 -1 -1 -1
   25   22468 0     14   1 -1 -1 -1 -1 -1 -1   3 2    1 0 -1 -1 -1
```

Figure 2.34: *Excerpt of data from the NASA Ames iPSC/860 log as converted to the standard workload format of the Parallel Workloads Archive. Compare with the raw data format in Figure 2.1.*

```
<job>
  <PEs>32</PEs>
  <runtime unit="minute">20</runtime>
</job>
```

This makes XML unsuitable for really large-scale datasets.

The main argument against using a plain ASCII format as suggested earlier is also the concern about data volume [27]. Using a binary format can reduce the volume considerably. For example, using the binary two's complement representation, 4 bytes can convey values from -2^{31} up to $2^{31}-1$. But if we use an ASCII representation we will need 10 bytes to convey this full range of values.

A potentially more important consideration is how much data is really needed. The common approach is the more, the better. This sometimes leads to the collection of gigabytes of data. However, it is often hard to argue that all this data is indeed needed and meaningful.

For example, consider the issue of memory references. Commonly used benchmark programs run for many billions of cycles, generating many billions of memory references. A full benchmark suite can easily create more than a trillion references. It is ludicrous to suggest that each and every one of them is important when we want to evaluate a system's architecture; what we want is the general behavior (in this case, the locality properties), not the details. In fact, emphasizing the details may lead to overfitting and results that are specific to the available data and do not generalize to other situations.

Thus in situations in which a lot of data is available a good alternative to keeping all of it is to just keep a sample. But this should be done subject to considerations regarding the internal structure of the data, as described on page 34.

2.5.3 Privacy

Note that the standard workload format from the Parallel Workloads Archive maintains no explicit user information, in the interest of privacy. It not only substitutes user IDs with numbers, but also does the same for groups and applications. The numbers are assigned in order of first appearance in the log. This process of hiding sensitive data is sometimes called *data sanitization*. In addition to preserving privacy, it also helps prevent the dissemination of commercially sensitive information [195, 564].

Data sanitization may be harder to apply in other situations, however. For example, Internet packets may contain sensitive user information, so packet traces typically discard the packet contents — leaving only the header. But one still needs to anonymize the source and destination IP addresses. These can, of course, be replaced by arbitrary numbers, but such a practice results in the loss of all the topological information embedded in the original addresses (e.g., different addresses that belong to the same subnet). The alternative is to use a prefix-preserving mapping that preserves the topological information [227, 560]. However, exposing the topological data may in some cases compromise anonymity.

In fact, comprehensive anonymization is not easy to achieve [533]. For example, hosts can also be compromised by port numbers (if they support a unique service) or even by packet timestamps (by fingerprinting a host's clock drift [409]).

Perhaps the most sensitive user information is contained in web search query logs. These logs may include sensitive information such as Social Security numbers and even credit card numbers, as well as potentially damaging or embarrassing information about search subjects [12, 386, 147].

An especially notorious example was the release of search logs by AOL in August 2006, which included some 20 million queries submitted by about 650,000 users over a period of three months. The sources of the queries were dutifully anonymized, being replaced by random numbers. However, the query contents were enough to identify the users in some cases, and also to expose their most personal interests and anxieties [53]. The resulting uproar caused AOL to withdraw the data within a few days, but copies continue to be available on the Internet[3]. As a result, web search vendors are now extremely wary about releasing their data.

Sanitizing web search data is more difficult than sanitizing communications data. It is not enough to disguise the source of the queries (e.g., by anonymizing IP addresses): it is also necessary to sever the association between the (anonymized) source and the queries themselves, because the query contents may give away personal information. The following approaches have been suggested or used [12, 147]:

- Shorten apparent user sessions by changing the (anonymized) user ID every so often. Thus each user ID will be associated with only a relatively short sequence of queries, and it will be harder to accumulate data about the user. However, this obviously hurts the ability to perform user studies.
- Remove identifying queries, e.g., all queries that are unique or submitted by only a handful of users.
- Hash all query words. As a result some statistical information may be retained, but it will not be tied to identifiable terms.
- Erase all names that appear in queries. However, it may be desirable to retain the names of celebrities, politicians, and the like. This can probably be achieved by only removing rare names or name combinations.
- Erase all long numerical sequences in order to eliminate the risk of leaking Social Security numbers and credit card numbers.
- Erase all queries altogether. This obviously eliminates the risk of leaking private data, but may greatly reduce the usability of the log for research.

In short, there is no simple solution, and there is always the danger of unintended exposure. But this risk does not justify the blocking of all data publication. Rather, a privacy policy should be decided on, which will balance the usefulness of the data with the risk of exposure [533, 147]. In certain cases — notably the recording of network

[3]This dataset is also used in this book in Section 9.4.6, but only for aggregate statistics.

communications — there may also be specific laws that spell out exactly what is allowed or not [523].

3

Statistical Distributions

An observed workload is a set of observations of workload items: processes that ran on a Unix workstation, requests fielded by a server, and so on. Each workload item is characterized by certain attributes. For example, a Unix process is characterized by its runtime, by how much memory it used, by which system calls it issued, and so on. Different workload items have different values for these attributes: one process may run for 7 ms, whereas another runs for 23 ms and a third for 5 whole seconds. The premise of statistical workload modeling is that these values can be regarded as samples from an underlying distribution. Thus if we find the distribution, we have a good model for how the workload looks. The model enables us to create synthetic workloads that mimic the original observed workload: we simply need to sample from the correct distributions.

This chapter reviews what distributions are and presents in detail those that are most useful for workload modeling. The focus is on developing an understanding of the concepts and of problems that occur in practice. The approach is to start with the data and see how we can describe it. The treatment is by no means rigorous, preferring to build intuition and understanding over plowing through minutiae of mathematical formalism. For a more abstract mathematical treatment and for background in probability see probability textbooks such as [501, 691, 581, 305, 40].

Notation Box: Random Variables and Distributions

We will be using the following notational conventions:

X, Y, Z	random variables
$X_1, X_2, \ldots X_i$	a series of random variables
x_1, x_2, y, z	random variates (actual sampled values)
$\bar{X}$	the (arithmetic) mean of random variables or samples
$\mathbb{V}\mathrm{ar}(X)$	the variance of random variables or samples
$\mathbb{S}(X)$	the standard deviation of random variables or samples
$X_{(p)}$	the p order statistic of random variables or samples
$\Pr(\cdot)$	the probability of something
$\mathbb{E}[X]$	the expected value of a random variable
$f(x)$	a probability density function (pdf), continuous distribution

$p(x)$	a probability mass function (pmf), discrete distribution
$F(x)$	a cumulative distribution function (CDF)
$\bar{F}(x)$	a survival function (or complementary CDF, CCDF)
μ	the mean of a distribution
σ	the standard deviation of a distribution
μ'_r	the rth moment of a distribution
μ_r	the rth central moment of a distribution
$\hat{a}$	an estimate of a parameter a

In brief, a *random variable* is a measurement that is indeterminate. In our context this usually reflects the fact that we are measuring one particular workload item out of an entire population of items. For example, "the runtime of a process" is a random variable because we do not single out a specific process. The *result* of such a measurement, which necessarily specifies a particular item (e.g., the process submitted by Joe at 8:37 AM, which ran for 13 ms), is called a *random variate*.

A series of random variables represents an ordered set of measurements — for example, the runtime of the first process, that of the second process, and so on. This is called a *stochastic process*, because the index often denotes time.

Random variables are assumed to come from a distribution (e.g., the distribution of process runtimes). Probability and statistics textbooks typically focus on the theory of distributions. Thus they use attributes such as the mean, or the "expected value" of the distribution, which is defined as

$$\mu = \mathbb{E}[X] = \sum_{i=1}^{\infty} x_i \, p(x_i)$$

where we have assumed there are an infinite number of possible discrete values denoted by x_i, and $p(x_i)$ denotes the probability to see value x_i.

In this book our focus is less on theory and more on looking at real data. Such data may be considered a sample from the distribution. A specific sample is actually a set of variates $x_1, x_2, \ldots x_n$. (Note that these are not exactly the same x_is as in the above formula for μ. In the formula the x_is denote all the possible values in the distribution. Here they denote a set of samples from the distribution, so they are a subset with repetitions of the x_is from the formula.) The average of these variates can serve as an estimate for the mean of the distribution

$$\hat{\mu} = \bar{\vec{x}} = \frac{1}{n} \sum_{i=1}^{n} x_i$$

In general, the more samples we have the better such estimates become, at least for "well-behaved" distributions.

Naturally, estimates such as $\bar{\vec{x}}$ depend on the specific samples $x_1, \ldots x_n$ that were observed. If another set of n samples is collected, they will be different, and so will their average. When we want to discuss averages *in general* we therefore prefer to talk about random variables representing the samples, and use them to define another random variable, denoted $\bar{X}$, which is their average

$$\bar{X} = \frac{1}{n} \sum_{i=1}^{n} X_i$$

This random variable then serves as the estimate (that is, $\hat{\mu} = \bar{X}$). In the following we use such random variable notation extensively when discussing the analysis of samples.

Other notations and concepts will be explained as they are introduced.

End Box

3.1 Describing a Distribution

Ultimately, it is desirable to describe a distribution using mathematical notation. For example, we might describe the distribution of possible outcomes when tossing a coin by a two-valued function, indicating that the probability of either heads or tails is one half:

$$\Pr(\text{heads}) = 0.5$$
$$\Pr(\text{tails}) = 0.5$$

The nice thing about coins, at least fair ones, is that we know in advance what the distribution should be. But in practically all real-life situations this is not the case. What we have to work with are observed values, which are assumed to be samples from some unknown distribution. For example, if we toss a coin 10 times and see heads 6 times and tails 4 times, we may jump to the conclusion that the coin is biased and that it is actually described by this distribution

$$\Pr(\text{heads}) = 0.6$$
$$\Pr(\text{tails}) = 0.4$$

However, an outcome of 6 heads out of 10 tosses is also possible with a fair coin — in fact, it may be expected to occur 20.5% of the time (and in an additional 17.2% of the time the outcome will be even more heads). So we cannot really be sure that the coin is biased. This is what statistics is all about: given a certain set of observations (e.g., a set of coin flips), test whether they support a given hypothesis (e.g., that the coin is biased). Hypotheses that we can be confident about — typically taken to mean that the results supporting them could occur by chance only 5% of the time or less — are then said to be "statistically significant". In the previous example, the evidence of 6 heads out of 10 tosses is not enough to support the hypothesis that the coin is biased.

Things become easier if we have very many samples. For example, if we toss a fair coin 1000 times instead of just 10, the 5% mark is at about 525 heads: the probability of seeing 526 heads or more is only 5%. The probability of seeing more than 550 heads is less than 0.1%. So if we see 600 heads, we can be very sure that the coin is indeed biased.

Moreover, if we have many samples, we can use the empirical results as an approximation (or rather, an "estimation") of the underlying distribution. For example, if 536 of the 1000 tosses were heads, we can characterize the coin by this distribution

$$\hat{\Pr}(\text{heads}) = 0.536$$
$$\hat{\Pr}(\text{tails}) = 0.464$$

which becomes increasingly accurate as more samples are added.

When characterizing a workload, very many different values are typically possible. For example, the runtime of a process can be as short as a fraction of a second or as

long as three weeks. If we have many samples, such as the runtimes of many processes, we have what is called an *empirical distribution*. Luckily, we usually have a lot of workload data, with many thousands or even millions of samples. Thus we can consider the empirical distribution as a very good approximation of the real distribution. In this book we often make do with empirical distributions. The remainder of this section deals with ways to describe and characterize an empirical distribution.

An important difference between the empirical distribution and the "real" one is that the real distribution may include values we have not seen in our sample. For example, in skewed distributions extremely high values are possible, but they are rare. Our sample may not include such very large values, but in real life the workload will occasionally include them, so our empirical distribution is not really representative. The problem is how to generalize it to a distribution that represents all possible runtime values. The common solution is to find a mathematical distribution that is similar to the data, so it is reasonable to assume that the empirical data came from this distribution. In Section 3.2 we introduce a number of useful mathematical distributions that are candidates for fitting the empirical data. How to actually fit the data is discussed in Chapter 4.

Note that distributions can be continuous, as in the case of process runtimes, or discrete, as for file sizes or packet sizes. In practice all workload distributions are actually discrete, because of the limited resolution with which we can make measurements. But we will often treat them as continuous anyway, because they have so many possible values.

3.1.1 Histograms, pdfs, and CDFs

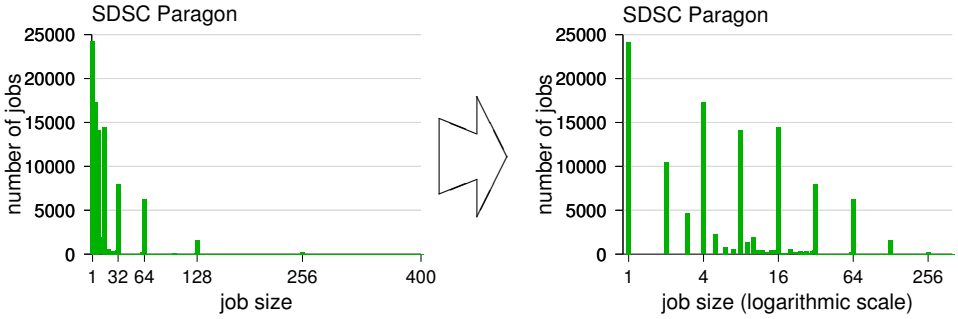
Histograms

The most direct way to represent an empirical distribution is with a histogram. This is simply a graph of the frequency with which different values occur. It is especially useful when the number of observed values is relatively small, and their range is limited. An example is given in Figure 3.1, which shows the distribution of job sizes in parallel machines. The X axis is the job sizes, and the height of the bars (the Y axis) indicates the number of times that each size value was observed in the dataset. We will sometimes use a line graph to show a histogram, but using bars is much more common.

Practice Box: Dealing with Large Ranges

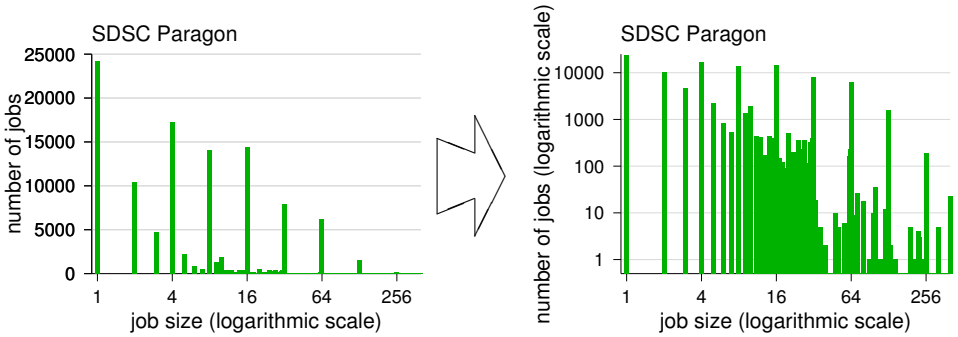
A basic problem when rendering distributions is how to deal with large ranges — both ranges of values and ranges of counts (or probabilities). This is immediately evident when looking at the histograms of Figure 3.1, where most of the ink is concentrated in the bottom left corner and near the axes.

In many cases, the values of interest are not uniformly spread over a large range. Rather, they tend to be concentrated at the bottom of the range (recall that we typically deal with non-negative values, so zero bounds the range from below). This enables us to use a logarithmic scale, in which a value of x is drawn at a distance of $\log x$ from the origin. For example, the SDSC Paragon histogram will change its appearance like this when using a logarithmic scale:



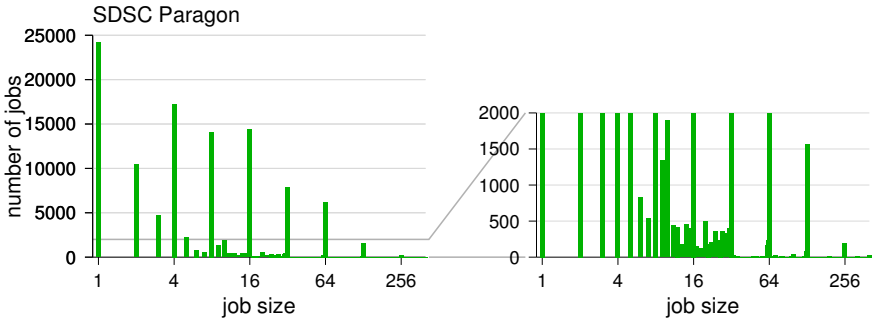
Doing so indeed spreads out the values nicely, and even emphasizes the popular powers of two as a side benefit.

Logarithmic scaling can also be applied to the counts (or probabilities) shown on the Y axis, when there are many small values and few large ones. Using the SDSC Paragon job-size distribution again as a test case, we get



These histograms expose the existence of many small values that were all but indistinguishable on the coarse linear scale needed to show the few big values. However, note that logarithmic scales are rather problematic in the sense that they are not intuitive to most humans. The grid lines in the graph come at constant intervals, but they do not represent constant increments; rather, they represent *orders of magnitude*. For example, 4-node jobs are 10 times more frequent than 128-node jobs, and a hundred times more frequent than 256-node jobs. While this data exists in the plot, these large differences are belittled rather than being emphasized.

An alternative is to use a second plot to zoom in on the small values. In effect, this presents the same data twice, at different scales:



This may be somewhat unusual, but is quite understandable.

End Box

The Probability Mass Function (pmf) and Probability Density Function (pdf)

Histograms show the distribution of data samples. But the exact same graph can also be used to describe a mathematical distribution. Let us first consider a discrete distribution, in which only a set of discrete values are possible. This can be described by a *probability mass function* (pmf), defined intuitively as the probability to see each value x:

$$p(x) = \Pr(X = x) \tag{3.1}$$

When describing an empirical distribution of n samples, this can be estimated using

$$\hat{p}(x) = \frac{|\{X_i \mid X_i = x\}|}{n}$$

i.e., the fraction of samples that had the value x. This is the same as the histogram, except that the Y axis is labeled differently.

Plotting all the individual values is not a useful approach when there are very many different possible values. For example, when we look at the distribution of runtimes of parallel jobs, we see that above a certain point all values occur only once, with large gaps between them. Small values may occur many times, but this is actually a result of quantization error, where measurements do not have sufficient resolution to distinguish between nearby values. At the limit of considering values as continuous, the probability for any specific value becomes zero. Therefore, in the context of continuous distributions, we use the *probability density function* (pdf). This captures the notion that the probability of seeing a sample in a certain range of values is proportional to the width of this range:

$$\Pr(x \leq X < x + \delta x) = f(x)\delta x$$

The pdf $f(x)$ is therefore not a probability, but the *density* of the probability; it is only turned into a probability by multiplying by a range.

Indeed, one can also consider ranges of values when drawing a histogram, and tabulate the frequency (or probability) of samples falling in each range. In this context, the ranges are usually called *bins*. The problem then is that the shape of the distribution we

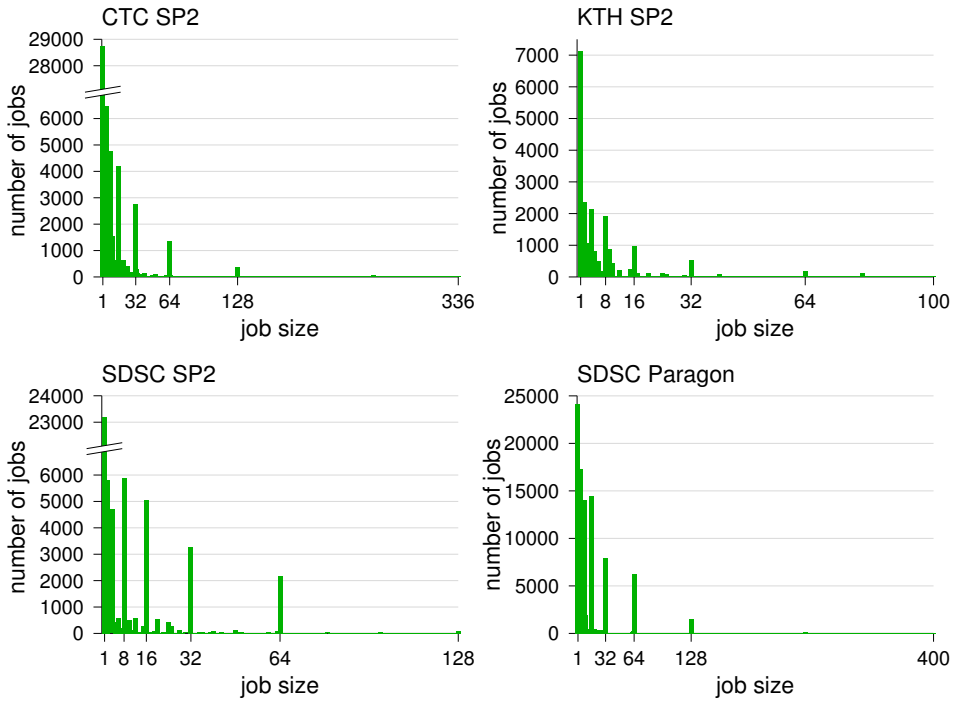


Figure 3.1: *Histograms showing the distribution of job sizes in parallel machines.*

get may depend on the number and sizes of the bins we select. At one extreme each sample is in a bin by itself. At the other extreme all the samples are in a single bin. In the middle, sets of nearby samples may either be grouped together, thus losing information about the fine structure of the distribution, or spread over multiple distinct bins, thus failing to observe that in fact they cluster in several groups. It is hard to find a scale that brings out all the features that are of interest (for ideas, see Greenwald [301]). Moreover, there may be no such single scale that is applicable to the whole distribution.

An example is given in Figure 3.2, which shows job arrivals for different times of day to the SDSC Paragon parallel machine. At a resolution of 10 seconds, it is rare that multiple jobs will arrive in the same bin, and this happens at random at all times of day. At a resolution of 10 minutes, on the other hand, we see a prominent arrival event that happens at around 5:50 AM — a set of 16 jobs that are submitted automatically within a few minutes at the same time each day. At coarser resolutions this peak of activity becomes diluted, and the most prominent feature of the arrival process becomes the concentration of job arrivals during normal work hours.

Practice Box: Histograms with Logarithmic Bins

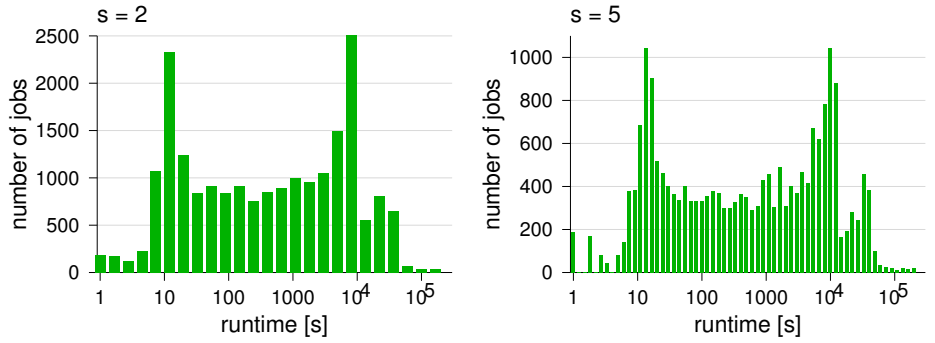
A special case occurs when the values are continuous, their range is very large, and the vast majority of the samples are small. In this situation, it makes sense to use logarithmi-

cally sized bins [482]. These are bins whose boundaries are not equally spaced; instead, successive boundaries have a constant *ratio*.

A simple choice is to use a ratio of 2, leading to bins that correspond to powers of two: the first bin will group values between 1 and 2, the next will group values between 2 and 4, the next between 4 and 8, and so on. An alternative that gives better resolution is to use the Fibonacci numbers as bin boundaries, leading to ratios that converge to 1.618. More generally, a formula can be used to map a sample value v to a bin b:

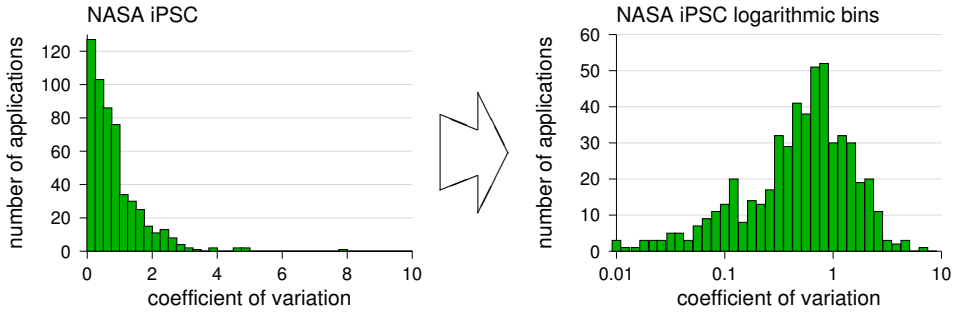
$$b = \lfloor s \cdot \log(v) + 0.5 \rfloor \tag{3.2}$$

where s is a scaling factor (the added 0.5 is used to turn truncation down to the next integer into rounding to the nearest integer). The larger s is, the more bins are used, and the better the resolution. This is demonstrated in the following two renditions of the same data from the KTH SP2:



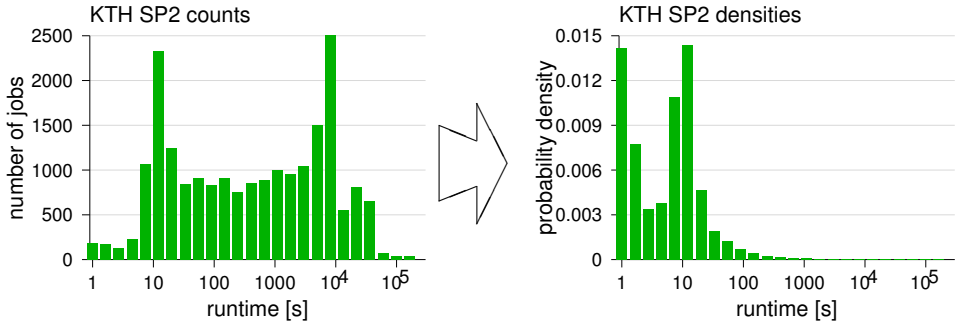
Effectively, s changes the base of the logarithmic transformation, and thus the ratio between bin boundaries.

A problem with logarithmic scaling may occur when the values in question may tend to zero. A logarithmic scale never reaches zero — it just keeps stretching indefinitely. Thus a histogram that actually has very many small values may end up looking as if it has just a few small values:



In such situations it may be better to zoom in using a linear scale with a fine resolution.

It is important to note that when histograms are drawn with logarithmic bins, their shape *does not* reflect the shape of the pdf (as opposed to using linear bins, where it does, but at reduced resolution). The reason is that the pdf is a *density*, so the number of items in a bin should be divided by the bin width. This can have a dramatic effect, significantly reducing the bars corresponding to the higher bins (which represent a wide range):



Depicting densities like this is important when one wants to give a mathematical description of the shape of the pdf. For just looking at the data, it may be better to use the original histogram, but keep in mind that the mass seen at the right end of the distribution is actually much more spread out.

End Box

To read more: Using histograms to estimate a distribution's density has received some attention in the literature [482, 71, sect. 4.3.1]. For example, Scott proposes a bin width of $\frac{3.49s}{\sqrt[3]{n}}$, where s is the estimated standard deviation and n the number of samples [604]. This is justified by an attempt to minimize the expected mean squared error. In this book, however, we often draw detailed histograms and are less concerned with fitting a mathematical function.

The Cumulative Distribution Function (CDF)

An alternative to the histogram is to represent a distribution by its *cumulative distribution function* (CDF). This is defined as the probability that a specific sample is smaller than or equal to a given value:

$$F(x) = \Pr(X \leq x) \tag{3.3}$$

Estimating this probability based on given samples is done by counting the fraction of samples that are smaller than each possible value:

$$\hat{F}(x) = F_n(x) = \frac{|\{X_i \mid X_i \leq x\}|}{n}$$

where $F_n(x)$ denotes the *empirical distribution of n samples*. Note that this is a step function, because there is a finite number of samples. So sorting the samples by size, such that $X_{(1)} \leq X_{(2)} \ldots \leq X_{(n)}$, we can also write this as

$$\hat{F}(x) = F_n(x) = \begin{cases} 0 & \text{if } x < X_{(1)} \\ i/n & \text{if } X_{(i)} \leq x < X_{(i+1)} \\ 1 & \text{if } x \geq X_{(n)} \end{cases} \tag{3.4}$$

(but note that if the distribution is discrete, and we observe multiple samples of the same value, some steps may be larger than $\frac{1}{n}$). This book uses such empirical CDFs extensively, but usually they are just called the CDF without stressing the "empirical" qualifier.

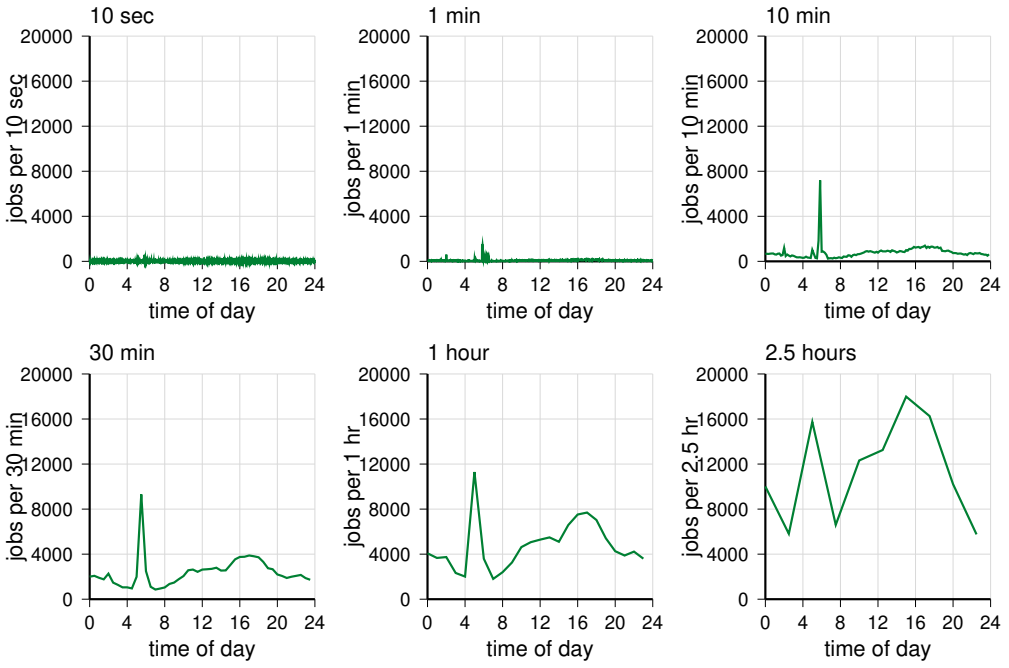


Figure 3.2: *Example of how the shape of a histogram changes as a function of the bin width. Job arrival data from the SDSC Paragon log, without cleaning.*

The CDF starts at zero below the smallest possible value (and for distributions on positive numbers, it is zero for $x < 0$). It is monotonically increasing and reaches one at the highest possible value (or at infinity). It increases rapidly in ranges that have a high probability and slowly in ranges that have a low probability. Figure 3.3 shows the CDFs that correspond to the histograms of Figure 3.2, with their different resolutions. Note that they are quite robust in shape, as opposed to the histograms that depend on the bin width.

The complement of the cumulative distribution function is called the survival function; this is the probability of observing a sample that is bigger than the given value:

$$\bar{F}(x) = 1 - F(x) = \Pr(X > x) \tag{3.5}$$

Naturally this is a monotonically decreasing function.

Using the CDF we can redefine the pdf in a way that explains the relationship between the two functions. For a continuous random variable, the CDF is still the probability of getting a value smaller than x, as per Equation (3.3). The pdf is defined as the derivative of the CDF:

$$f(x) = \frac{\mathrm{d}F(x)}{\mathrm{d}x} = \lim_{\delta x \to 0} \frac{F(x + \delta x) - F(x)}{\delta x} \tag{3.6}$$

This retains the interpretation of being a density — that when multiplied by a small interval, δx, gives the probability of being in that interval:

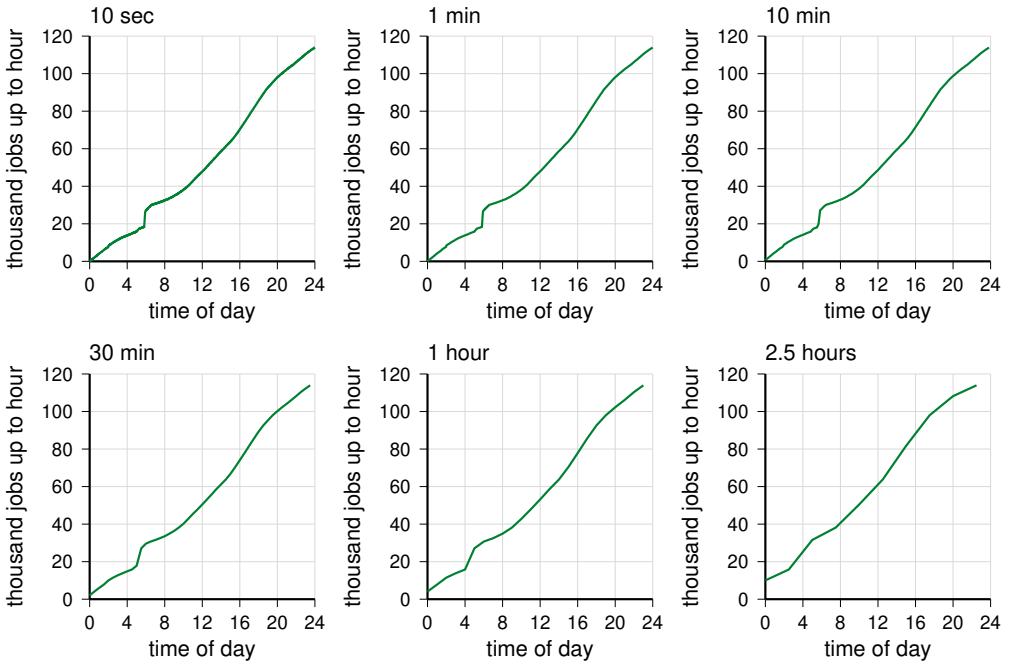


Figure 3.3: *CDFs of the histograms shown in Figure 3.2 are relatively robust to bin width.*

$$f(x)\,\delta x = \Pr(x < X \leq x + \delta x)$$

Conversely, the CDF is the integral of the pdf:

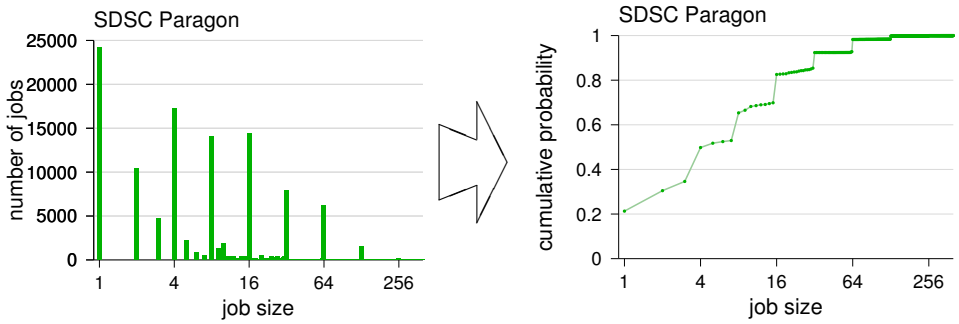
$$F(x) = \int_0^x f(t)\mathrm{d}t \tag{3.7}$$

where we assume that this is a distribution on positive values; in the general case, the lower boundary of the integration is $-\infty$.

Practice Box: Interpreting a CDF

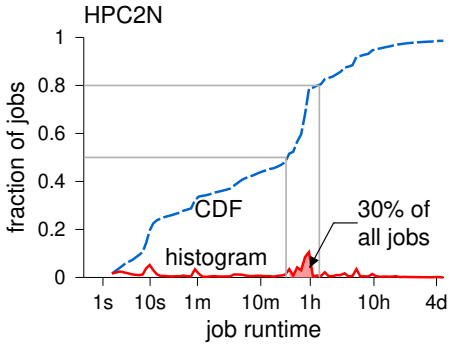
Although CDFs have several advantages over histograms, they are somewhat less intuitive. It takes some practice to read a CDF.

One point to keep in mind is that peaks in the histogram translate into steep slopes in the CDF. In particular, a modal distribution (in which only a few discrete values appear) turns into a stair shape. For example, the CDF of the job-size distribution looks like this:



Note that only really large modes are seen as steps. Smaller modes can only be seen in the histogram and tend to be lost in a CDF. In general, CDFs are not good at characterizing modal distributions.

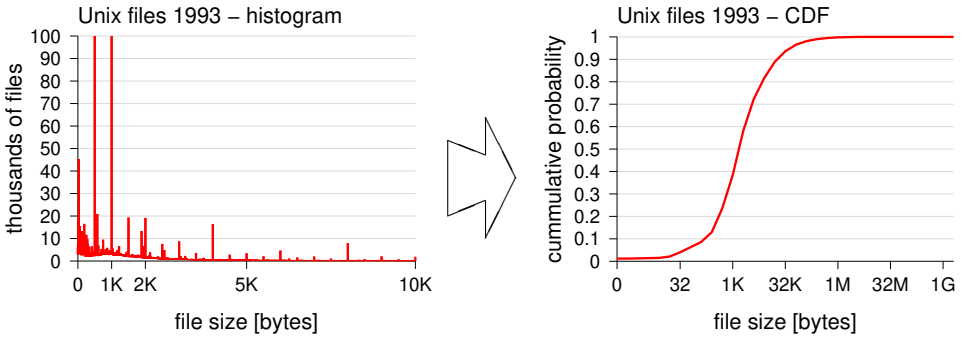
Even if the distribution is not strictly modal, steep areas in the CDF help identify where most of the mass is concentrated. Moreover, by noting the span of the CDF covered, it is easy to quantify exactly what fraction of the distribution is involved. This would be hard to do with the histogram — we would need to sum the individual values.



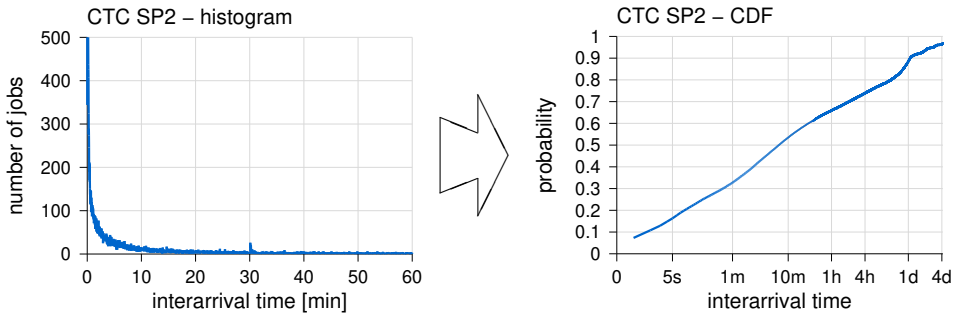
Another point to remember is that when one CDF is below and to the right of another, it means that the distribution has more large values. For example, the distribution of interarrival times of very active users (such as user 374 on the SDSC SP2) tends to have shorter interarrivals than the general population, so it is higher and to the left:

In fact, exposing such differences in tendency is one of the major strengths of CDFs as opposed to histograms, especially when the distribution is noisy and multimodal.

An important benefit of using the CDF when looking at skewed data is that it allows the size of the tail to be estimated easily. A histogram only enables us to see that there is a tail, but we cannot know how many samples it includes. With a CDF, we can make statements such as "50% of the files are smaller than 2 KB, and the top 5% are all larger than 32 KB":



In a related vein, the CDF allows one to easily distinguish between different types of tails. For example, compare the preceding graphs with the following ones. The tails of the histograms are impossible to distinguish, but this CDF is clearly uniform in log-space (i.e., after a logarithmic transformation), whereas the previous one is not.



Finally, there is the question of normalization. Empirical CDFs are typically normalized to the range 0 to 1, regardless of the number of samples involved. Thus a CDF made up of 13 samples covers the same range as a CDF made up of 72,000 samples, although it might not be as smooth. Histograms, in contrast, typically show counts. But it is also possible to normalize a histogram to show the *fraction* of samples in each bin. This is especially useful when one wants to compare distributions without being distracted by effects caused by different numbers of samples. One can also use "un-normalized CDFs" (actually, cumulative histograms) when the number of samples is in fact important.

End Box

Shapes of Distributions

A histogram provides a graphic illustration of the shape of the distribution. However, the observed shape depends on the resolution of the observation. An example is given in Figure 3.4[1]. This shows a distribution of memory usage, where the maximal possible value is 32 MB. A logarithmic scale is used, and samples are grouped into logarithmically sized bins. This is done again using Equation (3.2), and the resolution depends on s: when $s = 4$ only 23 bins are used, whereas $s = 12$ leads to 67 bins and far better

[1]This histogram does not approximate the pdf, because the counts have not been scaled by the range of each bin.

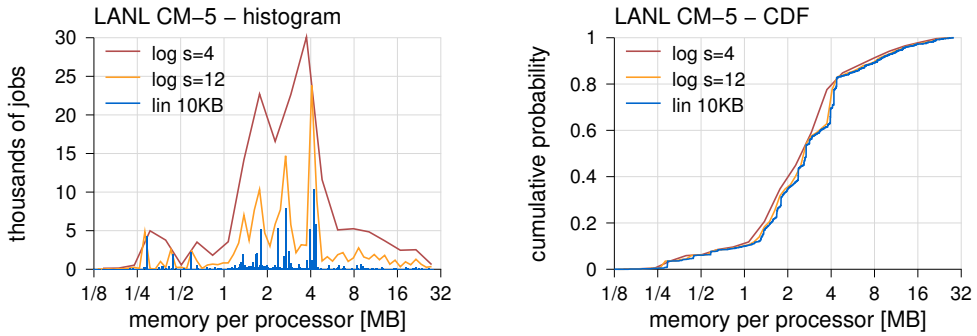


Figure 3.4: *A histogram (left) depends on the granularity of the observation, as seen by comparing logarithmic bins with a fine linear scale. A CDF (right) is more robust, but loses detail. Data is per-processor memory usage from the LANL CM-5 log.*

resolution. The interesting point is that multiple small peaks that are seen at a fine resolution may be united into one large peak when a coarse resolution is used, leading to a different functional shape. The shape of the CDF is much more robust to the bin size with which the distribution is observed. However, this comes at the price of losing most of the details (i.e., the number and relative sizes of the peaks).

Different distributions naturally have different shapes. A commonly encountered one is the so-called bell shape, like that of a normal distribution. Symmetrical normal distributions are extremely important in statistics, and also provide a good model for numerous datasets, such as the heights of individuals. But in computer workloads two other types of distributions are much more common.

One type is asymmetrical, or skewed distributions. In many cases, the values of interest are positive, so they are bounded from below by zero, but they are not bounded from above. Thus we can have many small values and a few very big values. Such distributions are said to possess a "right tail", because their pdf is skewed and extends a large distance to the right. For example, the distribution of process runtimes is skewed with a long tail. Specific examples are described in Section 3.2.

The other common feature is that distributions may be modal, meaning that certain discrete values are much more common than others. For example, the distribution of job sizes on parallel supercomputers emphasizes powers of two, as shown earlier. The distribution of packet sizes on a communications network tends to emphasize sizes used by the protocols used on the network. Modal distributions are discussed in Section 4.4.2.

3.1.2 Central Tendency

Histograms can show a whole distribution. But how can we summarize the distribution with a small amount of data?

System	*Arithmetic average*	*Geometric average*	*Median*
CTC SP2	9.79	3.48	2
KTH SP2	7.44	3.44	3
SDSC Paragon	17.00	7.50	8
SDSC SP2	11.94	5.09	4

Table 3.2: *Arithmetic average, geometric average, and median values for the distributions of job sizes shown in Figure 3.1.*

The Average

The most concise way to describe a distribution is by a single number. This number should convey the "center of mass" of the distribution, typically interpreted to be the mean of the distribution, or the "expected" value. It is expected because it is the weighted sum of all possible values, where the weights are the probabilities of seeing the different values. For a discrete distribution this is

$$\mu = \mathbb{E}[X] = \sum_{i=1}^{\infty} x_i \, p(x_i)$$

and for a continuous one

$$\mu = \mathbb{E}[X] = \int_0^{\infty} x \, f(x) \, \mathrm{d}x$$

where we have assumed that the distribution is on positive values (otherwise the integration should start from $-\infty$).

When we are dealing with a finite number of samples, the mean is estimated by their average value, defined as

$$\hat{\mu} = \bar{X} = \frac{1}{n} \sum_{i=1}^{n} X_i \tag{3.8}$$

According to the law of large numbers, the average converges to the true mean of the underlying distribution as more samples are added. The averages of the job size histograms of Figure 3.1 are shown in Table 3.2.

Details Box: Different Types of Means

The formula given in Equation (3.8) is called the *arithmetic mean*. There are other types of means as well.

One alternative is the *geometric mean*. This is defined by the formula

$$\bar{X}^* = \sqrt[n]{\prod_{i=1}^{n} X_i} \tag{3.9}$$

Note, however, that a naive calculation based on this equation is likely to overflow for a large number of samples. A better approach is therefore to use the equivalent form:

$$\bar{X}^* = e^{\left(\frac{1}{n}\sum_{i=1}^{n}\log X_i\right)}$$

The main reason to prefer the geometric mean is that it gives "equal weights" to all inputs. Consider a case in which the values being averaged are very different, with many small values and several large values. In this situation, the arithmetic mean is dominated by the large values. Changing the largest value by, say, 10%, will have a noticeable effect on the arithmetic mean, whereas changing the smallest value by the same factor of 10% will have a negligible effect. On the other hand, changing *any* value by 10% will have the same effect on the geometric mean: it will change by a factor of $\sqrt[n]{1.1}$. In addition, if the sample values are ratios X_i/Y_i, their geometric mean has the attractive property that the average of the ratios is equal to the ratio of the averages:

$$\sqrt[n]{\prod_{i=1}^{n}\frac{X_i}{Y_i}} = \frac{\sqrt[n]{\prod_{i=1}^{n}X_i}}{\sqrt[n]{\prod_{i=1}^{n}Y_i}}$$

So why is the geometric mean so seldom used? One reason is that it may be non-monotonic relative to the more intuitive arithmetic mean. This means that there are cases where the sum (and hence arithmetic average) of a set of numbers is bigger than that of another set, but the geometric mean is smaller. A simple example is the pair of numbers 10 and 10, for which both the arithmetic and geometric means are 10. Compare this with the pair 1 and 49, which have an arithmetic mean of 25, but a geometric mean of only 7.

Nevertheless, as we see later, the geometric mean may be quite useful when characterizing skewed distributions, because it is less responsive to large values. This is demonstrated for the job-size distributions in Table 3.2.

Another alternative is the *harmonic mean*, defined by

$$\bar{X}^h = \frac{1}{\frac{1}{n}\sum_{i=1}^{n}\frac{1}{X_i}} \tag{3.10}$$

This is useful when the x_is represent rates rather than sizes. Consider driving to work in the morning, where the first half of the trip (say 10 km) takes 10 minutes at 60 km/h, but then traffic stalls and the next 10 km take 30 minutes at 20 km/h. The distance you covered during those 40 minutes is 20 km, so your average speed was 30 km/h. This is indeed the harmonic mean of 60 and 20. But taking the arithmetic average gives a much more optimistic (and wrong) 40 km/h.

Actually, this example only worked because the two segments were of the same length — 10 km. The correct generalization is the weighted harmonic mean, in which the different rates are weighted according to how much of the total they represent, rather than having equal weights of $\frac{1}{n}$. Thus if $X_i = Y_i/Z_i$, the weight of X_i will be proportional to Y_i. The weighted harmonic mean is then equivalent to the quotient of the sums of the numerators and denominators

$$\bar{X}^h = \frac{1}{\sum_{i=1}^{n}\frac{Y_i}{\sum_{j=1}^{n}Y_j}\frac{Z_i}{Y_i}} = \frac{\sum_{j=1}^{n}Y_j}{\sum_{i=1}^{n}Z_i}$$

In general, the geometric mean is smaller than the arithmetic mean, and the harmonic mean is smaller yet. This is easy to see by calculating the three means of 6 and 8.

To read more: Selecting the right type of mean is especially important when summarizing benchmark results. This has been discussed by Fleming and Wallace [263], Smith [638], and Giladi and Ahituv [284]. Lilja devotes an entire chapter to means [444, chap. 3]; see also Jain [367, chap. 12].

End Box

A common problem with the mean is that it may not really be representative of the complete dataset. In particular, the mean value is not necessarily the most common value (the mode). For example, nearly everyone has more than the average number of legs. This is so because the average is slightly less than two, due to a small number of people with amputations. Likewise, the vast majority of people have less than the average income, due to a small number of individuals who make much much much more and thus pull the average upward.

Skewed distributions with some very large values may resemble a "reasonable" distribution with some outliers. When the large values are indeed outliers — that is, suspect values that may not reflect real data — one wants to eliminate their effect on statistics such as the mean. This can be done by using a trimmed mean, where, say, the top and bottom 20% of the samples are removed before calculating the average [212]. However, this approach is not recommended in general for workload data, where skewed distributions are the norm. In this situation, removing the top samples risks losing very important information.

The Median

An alternative to the mean is the *median*, the value that divides the set into two equal parts: half of the samples are smaller than this value, and half are larger[2]. In symmetrical distributions, such as the normal distribution, the average and median are equal (but of course, in any given set of samples, there may be a small difference between them). But the distributions describing workloads are most often skewed, with a small probability of very high values. In such cases the arithmetic average tends to be much larger than the median (Table 3.2). It is therefore often thought that the median is a more representative value of where many samples are concentrated.

Note that a careful choice of words was employed in the last sentence. In a skewed distribution the median may be representative of where many samples are concentrated, but this does *not* imply that it is also representative of where much of the mass of the distribution is concentrated. On the contrary, much of the mass may be found in the few very large samples from the tail of the distribution. When this happens the distribution is called a heavy-tailed distribution. Such distributions are discussed in Chapter 5.

Practice Box: Calculating the Median

The mean of a set of samples is easy to calculate online. One simply accumulates the sum of all the samples and their count, and then divides the sum by the count.

[2] "Half" is not strictly accurate if the number of samples is odd, but we usually have lots of samples and ignore this.

Finding the median is harder. For starters, one has to store all the samples. When all are available, they can be sorted, and the middle sample is then identified; all other order statistics can likewise be found. However, this comes at the cost of $O(n)$ storage and $O(n \log n)$ time (for n samples). But a linear time algorithm (running in $O(n)$ time) is also possible [150, sect. 9.2]. The idea is to partition the data recursively into parts that are smaller and larger than a selected pivot element, as is done in Quicksort. The difference is that instead of following both branches of the recursion, we only need to follow one of them: the one that includes the median. The correct branch is identified based on the count of values in each side of the partition.

An online algorithm, in which we do not store all the samples, was proposed by Jain and Chlamtac [368]. This is based on approximating the shape of the CDF using five "markers". Assuming we want to find the p percentile (for the median $p = 0.5$), the markers are placed at the minimal observation, the maximal observation, the p percentile, and halfway between the p percentile and the extremes: at $p/2$ and at $(1 + p)/2$. As more and more samples are seen, the minimal and maximal markers are updated in the obvious manner. The middle markers are updated using a piecewise parabolic approximation of the shape of the CDF. The approximation of the p percentile is read off this approximation of the CDF.

While the above algorithm is very efficient, we do not have a bound on how far off it might be from the real value of the desired percentile. An algorithm that does indeed provide such a bound was proposed by Manku et al. [470]. The idea is to keep b arrays of k sample values each. Initially these arrays are simply filled with samples. But when we run out of space, we take a set of full buffers and collapse them into one by sorting and selecting k equally spaced samples. This is done using the buffers that contain data that has been collapsed in this way the minimal number of times. At the end, we have a set of equally spaced samples that allow us to approximate various percentiles. The parameters b and k are chosen so as to guarantee that we have enough samples to meet the desired bound on the accuracy of the percentiles.

End Box

Minimizing the Distance from Samples

Another way to compare the mean and the median is the following. We are looking for a number that describes the "middle" of a distribution [205]. Denote this by M. Given a set of samples $x_1, x_2, \ldots x_n$ we will say that M is a good representative of the middle of the distribution if it is not too far from any of the samples. This can be formalized by treating M as a variable, and then finding the value that minimizes the sum of its distances from the n values that were sampled. But we don't want positive and negative differences to cancel out, so we will use the differences squared, to ensure they are all positive. Our metric is then

$$D = \sum_{i=1}^{n} (M - x_i)^2$$

and we want to find the M that minimizes this expression. By differentiating and equating to zero, we find that

$$\frac{\partial D}{\partial M} = 2 \sum_{i=1}^{n} (M - x_i) = 0$$

and that M is the average of the sampled values: $M = \frac{1}{n}\sum_{i=1}^{n} x_i$.

But there is also another way to guarantee that positive and negative differences do not cancel out: we could simply take their absolute values. This leads to formalizing the sum of differences as

$$D = \sum_{i=1}^{n} |M - x_i|$$

Now when we differentiate and equate to zero (in order to find the M that minimizes this expression) we get

$$\frac{\partial D}{\partial M} = \sum_{i=1}^{n} \operatorname{sgn}(M - x_i) = 0$$

where sgn is the sign function

$$\operatorname{sgn}(x) = \begin{cases} 1 & \text{if } x > 0 \\ 0 & \text{if } x = 0 \\ -1 & \text{if } x < 0 \end{cases}$$

To achieve a sum of zero, M must be exactly in the middle of the set of values, with half of them bigger and half smaller. In other words, M is the median.

Thus the average minimizes the sum of squared deviations, whereas the median minimizes the sum of absolute deviations.

3.1.3 Dispersion

A more general problem is that the mean or median only gives a single characterization, which cannot reflect the full distribution of possible values. For example, one can drown in a lake with an average depth of 10 centimeters, because some parts of it are much deeper.

The Variance and Standard Deviation

If we allow two numbers, we can also characterize the dispersion of the distribution in addition to its center — i.e., how much the values differ from each other. The way to do this is to look at the distances of the different values from the center (e.g., from the mean). But we can't just take the differences, because some values are larger than the mean (positive difference) whereas others are smaller (negative difference), and they will cancel out. As noted earlier, a simple way to make them all positive is to square them. Thus the metric for dispersion is the expected square of the distance of the different values from the average, known as the *variance*:

$$\sigma^2 = \mathbb{E}[(X - \mu)^2] = \sum_{i=1}^{\infty} (x_i - \mu)^2 \, p(x_i)$$

(for the discrete case). Given n samples, this is estimated by

$$\hat{\sigma^2} = \mathbb{V}\mathrm{ar}(X) = \frac{1}{n-1}\sum_{i=1}^{n}(X_i - \bar{X})^2 \tag{3.11}$$

where the estimate $\bar{X}$ replaces the real mean μ, and the normalization is by $n-1$ rather than n.

The variance has the disadvantage of reflecting the squared distances. To better reflect the actual distances one therefore often uses the square root of the variance, known as the *standard deviation*:

$$\hat{\sigma} = \mathbb{S}(X) = \sqrt{\mathbb{V}\mathrm{ar}(X)} = \sqrt{\frac{1}{n-1}\sum_{i=1}^{n}(X_i - \bar{X})^2} \tag{3.12}$$

which is the reason that the variance is denoted by σ^2. A related metric is the coefficient of variation (CV), which is defined as the quotient of the standard deviation divided by the mean:

$$CV = \frac{\mathbb{S}(X)}{\bar{X}} \tag{3.13}$$

This metric normalizes the units, essentially measuring the standard deviation in units of the mean: a standard deviation of 1000 around an average of 17,000,000 is, of course, much smaller (relatively speaking) than a standard deviation of 1000 around an average of 1300.

Practice Box: Calculating the Variance

A common misconception is that calculating the variance requires two passes on the data — one to calculate the average, and the other to calculate the variance based on each value's difference from the average. In fact, one pass is enough, using the identity

$$\frac{1}{n}\sum_{i=1}^{n}(X_i - \bar{X})^2 = \frac{1}{n}\sum_{i=1}^{n}X_i^2 - \bar{X}^2$$

which is easy to verify by opening the parentheses. The trick is to keep two running sums: one of the individual values and another of their squares. The procedure is as follows:

1. Initially $sum = 0$ and $sumsq = 0$.
2. Scan the sampled data and update the running sums:

$$sum = sum + X_i$$
$$sumsq = sumsq + X_i * X_i$$

3. Calculate the average $\bar{X} = sum/n$, where n is the number of samples.
4. The variance is then given by

$$\mathbb{V}\mathrm{ar}(X) = \frac{1}{n}sumsq - \bar{X}^2 \tag{3.14}$$

Note that to estimate the variance from samples the normalization factor should be $1/(n-1)$ rather than $1/n$. This compensates for using $\bar{X}$ rather than the true mean of the distribution in the formula. The samples naturally tend to be closer to the sample average $\bar{X}$ than

to the true mean, so using the distances to $\bar{X}$ will lead to an underestimate; we compensate by dividing by a smaller number. In the one-pass calculation, this can be achieved by using

$$\mathbb{V}\mathrm{ar}(X) = \frac{1}{n-1} sumsq - \frac{n}{n-1} \bar{X}^2$$

For large n the difference is, of course, negligible.

This procedure may run into trouble if the values are extremely large (leading to rounding errors) or if the variance is much smaller than the mean. In particular, if the two terms in Equation (3.14) are very close to each other, so that they share many digits, subtracting one from the other will lead to a large loss of accuracy. An alternative is to maintain the variance online [728]. Define $V_n = \sum_{i=1}^{n} (X_i - \bar{X})^2$, that is, the variance without the normalization factor. This can then be updated as each new X value is obtained by the formula

$$V_n = V_{n-1} + \frac{n-1}{n} (X_n - M_{n-1})^2$$

where $M_n = \frac{1}{n} \sum_{i=1}^{n} X_i$ is the mean calculated from the samples, which is also updated using

$$M_n = \frac{n-1}{n} M_{n-1} + \frac{1}{n} X_n$$

To read more: A discussion of these and additional, related algorithms is given in the Wikipedia article on "Algorithms for Calculating Variance".

End Box

Figure 3.5 shows the result of calculating the standard deviation of one of the parallel job-size distributions from Figure 3.1, based on the average values given in Table 3.2 and the standard deviation values given in Table 3.3. Due to the skewed nature of the distribution, using the standard deviation leads to invading the realm of negative numbers.

Indeed, the variance is an example of a mathematical convenience that may have unintended consequences. The reason that the variance is defined as the average of the distances from the mean *squared* is to ensure that the values are all positive and do not cancel out. Squaring is also a differentiable function, and the variance can be calculated analytically for many useful mathematical distributions. But squaring also has the side effect of giving larger weight to higher values: the contribution of each value to the sum is the value squared, which can be seen as the value weighted by itself. If the distribution is skewed, a small number of large values can lead to the impression that it has a very wide dispersion, although most values are actually quite close to each other. Therefore other metrics for dispersion may be more suitable in this case.

Another dangerous misconception is that about 68% of the mass of a distribution may be found within a range of one standard deviation above and below the mean. This is true for the normal distribution, but is not true in general. In particular, it can be way off the mark for skewed distributions.

The common approach to handling a few large values is to assume they are outliers and eliminate them. In particular, the suggested metric for dispersion is the Winsorized variance [212]. This is analogous to the trimmed mean, with one important difference:

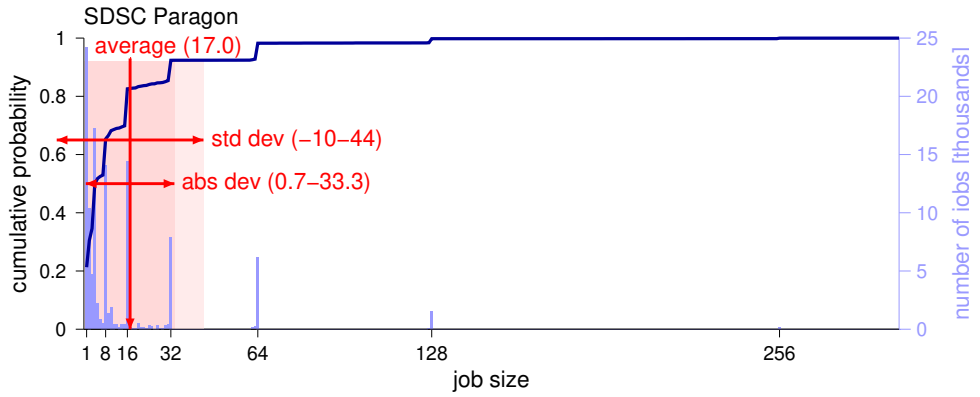


Figure 3.5: *The standard and absolute deviations as metrics for dispersion around the mean, applied to a skewed positive distribution that is also largely modal (the sizes of parallel jobs, shown using superimposed histogram and CDF). Note that the distribution actually continues up to a maximal value of 400.*

instead of eliminating the top and bottom 20% of the samples (or some other fraction), it replaces them with the extremal values that are left. For example, if our original samples are 0, 0, 1, 2, 3, 3, 5, 8, 15, 73, the Winsorized series will be 1, 1, 1, 2, 3, 3, 5, 8, 8, 8. The Winsorized variance is then the variance of this Winsorized dataset. But this approach is not recommended for workload data, in which skewed distributions are common, and the large samples may in fact convey important information.

The Absolute Deviation

The problem with the standard deviation stems from the practice of using squared values to make them all positive, so that deviations in one direction do not cancel out with deviations in the other direction. An alternative is to use the absolute values of the deviations, leading to the average absolute deviation:

$$AbsDev(X) = \frac{1}{n}\sum_{i=1}^{n}|X_i - \bar{X}| \tag{3.15}$$

(Note that in this case it is of course improper to take a square root.) An alternative definition uses the deviations from the median, rather than from the average. Comparing this with the standard deviation for distributions such as those shown in Figure 3.1 leads to the results shown in Table 3.3 and Figure 3.5 — the average absolute deviations are significantly smaller. Although this does not guarantee that the range specified by the average absolute deviation around the mean does not include negative values, it reduces that danger.

Figure 3.6: *Quantifying dispersion using the geometric mean and multiplicative standard deviation.*

The Multiplicative Standard Deviation

The standard deviation and absolute deviation imply a symmetrical additive range around the mean: they characterize the distribution as concentrated in the range $\bar{X} \pm \mathbb{S}(X)$. This is, of course, appropriate for symmetrical distributions such as the normal distribution. But it is inappropriate for skewed distributions that extend a long way to the right. This leads to the idea of using a *multiplicative* standard deviation (also called the geometric standard deviation) [445], where the range is given as $\bar{X}^* \times\!/ \mathbb{S}(X)^*$. The symbol $\times\!/$ means "multiply-divide", in analogy to the "plus-minus" used with the conventional standard deviation. Thus the lower end of the range is defined by the average divided by the multiplicative standard deviation, while the higher end is the average multiplied by it:

$$\bar{X}^* \times\!/ \mathbb{S}(X)^* = [\bar{X}^* / \mathbb{S}(X)^* \,..\, \bar{X}^* \times \mathbb{S}(X)^*]$$

To implement this idea we need to define $\bar{X}^*$ and $\mathbb{S}(X)^*$. Given the multiplicative nature of the definition, it is natural to use the geometric mean $\bar{X}^*$ as the centerpoint. As for the multiplicative standard deviation, it should reflect the average quotient of the different values and this centerpoint (that is, the average of $\frac{X_i}{\bar{X}^*}$), where again the geometric average is meant. But given that the values are distributed around the centerpoint, some of these quotients will be smaller than 1, while others will be larger than 1. Thus multiplying them by each other to find the geometric average may cancel them out. The solution is to do the averaging in log-space, ensuring that all the logs are positive by using the common trick of squaring them. Then the square root of the result is used before the inverse transformation. The final result of all this is

$$\mathbb{S}(X)^* = e^{\sqrt{\frac{1}{n}\sum_{i=1}^{n}\left(\log \frac{X_i}{\bar{X}^*}\right)^2}} \tag{3.16}$$

Figure 3.6 shows the result of applying this to one of the parallel job-size distribu-

System	*Std dev*	*Abs dev*	*Mult std dev*	*SIQR*	Q_1–Q_3	95^{th}%
CTC SP2	±20.98	±10.76	×/3.84	±4.0	1.0–9.0	33.0
KTH SP2	±12.49	±7.14	×/3.21	±3.5	1.0–8.0	32.0
SDSC Paragon	±27.00	±16.30	×/3.62	±6.5	3.0–16.0	64.0
SDSC SP2	±17.14	±11.70	×/3.78	±7.5	1.0–16.0	64.0

Table 3.3: *Comparison of metrics for dispersion, as applied to the distributions of parallel job sizes of Figure 3.1: the conventional and multiplicative standard deviation, the absolute deviation, and percentile-related metrics such as the SIQR.*

tions from Figure 3.1, based on the geometric average values given in Table 3.2 and the multiplicative standard deviation values given in Table 3.3. Quite obviously, the multiplicative standard deviation does a better job at characterizing the distribution than the conventional standard deviation shown in Figure 3.5. Interestingly, as shown in Table 3.3, the multiplicative standard deviation also indicates that the four job-size distributions have similar characteristics, which was not the case when using the conventional standard deviation.

The (S)IQR and Quartiles

Yet another approach is to use the interquartile range, or IQR. Quartiles are the values that divide the distribution into four equal parts: the first quartile (often denoted by Q_1) is that value that one-quarter of the distribution is below it, the second quartile is the median, and the third quartile (Q_3) is the value that three-quarters of the distribution is below it. The Interquartile range is the difference between the first and third quartiles (i.e., $Q_3 - Q_1$). Alternatively one can simply note these two quartiles, with the understanding that the middle half of the distribution is contained in the range from Q_1 to Q_3 (Figure 3.7). In either case, this is robust against rare large values that may distort the variance.

Another possibility is to calculate the average of the distances from Q_1 and Q_3 to the median. This is called the semi interquartile range, or SIQR, because it is half of the IQR. The problem with this metric is that it implies a symmetry around the median. With skewed positive distributions this measure is typically misleading, so it is better to directly specify the range from the first to the third quartiles (as is done in the box plots described later). The size of the resulting range is the same as when using ± the SIQR, but it will typically be shifted to the right and therefore be asymmetrical around the median.

Finally, taking an even more extreme approach, we may specifically take the nature of positive skewed distributions into account. Being positive, the bottom end of such distributions is always zero. Therefore the high values seen also specify the entire range of the distribution. Because the very highest values are by definition rare, it is better to use a somewhat lower point, such as the 90th or 95th percentile, as representing the "top" of the distribution and also its dispersion [276]. An alternative that is also useful

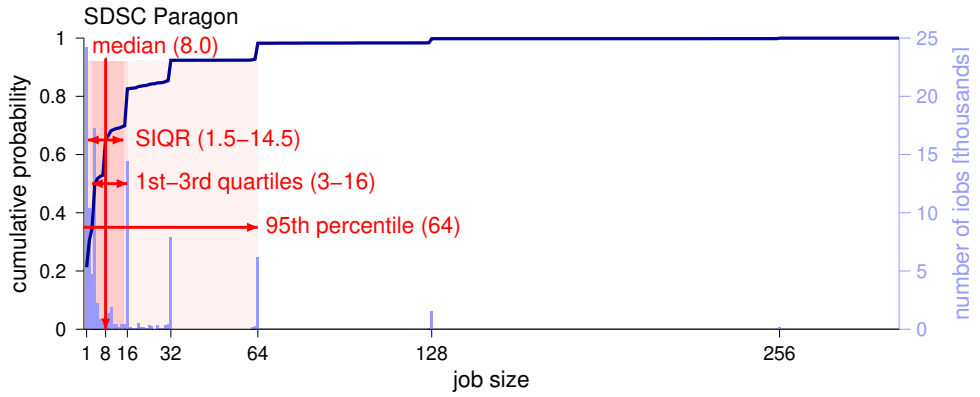


Figure 3.7: *Percentile-based metrics for dispersion.*

for less skewed distributions is to use a certain range of percentiles, for example the 90% interval from the 5th percentile to the 95th percentile.

3.1.4 Moments and Order Statistics

By now you might be confused, or you might begin instead to see a pattern. In each case, we have two basic approaches — one based on moments and the other on order statistics.

Moments (or, rather, "sample moments", because we base the discussion on data samples and not on theoretical expressions using the pdf) are defined as the average of the observations raised to some power. The first moment is the simple average of Equation (3.8). The rth moment is

$$\mu'_r = \frac{1}{n}\sum_{i=1}^{n} X_i^r \tag{3.17}$$

The variance can be calculated from the first and second moments:

$$\mathbb{V}\mathrm{ar}(X) = \mu'_2 - \left(\mu'_1\right)^2$$

Skewness, kurtosis, etc. are related to higher moments. Some distributions can be completely specified in terms of their moments; more on that later.

A related set of values are the *central* moments. These are the moments of the centered data, where X_is are taken relative to the mean (they are therefore also called "moments about the mean" as opposed to the moments defined above which are "moments about the origin") and are denoted without the $'$:

$$\mu_r = \frac{1}{n}\sum_{i=1}^{n}(X_i - \bar{X})^r \tag{3.18}$$

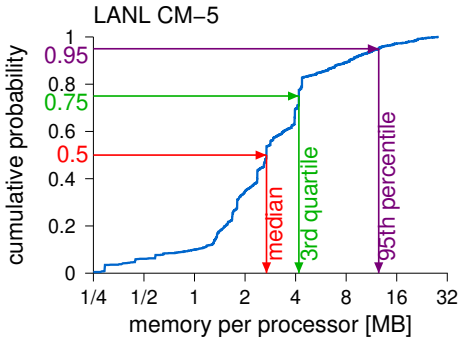


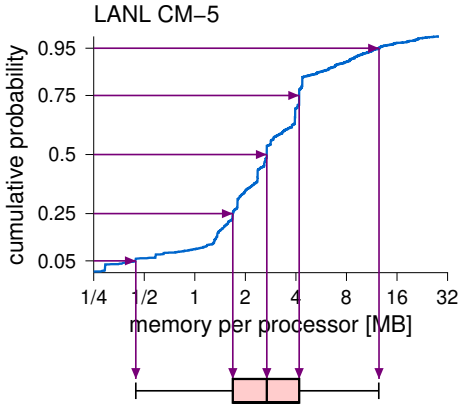
Figure 3.8: *Example of finding percentiles from the CDF.*

The reason for using central moments is that when we are studying fluctuations, it is more useful to have data that assumes both positive and negative values, and has a 0 mean.

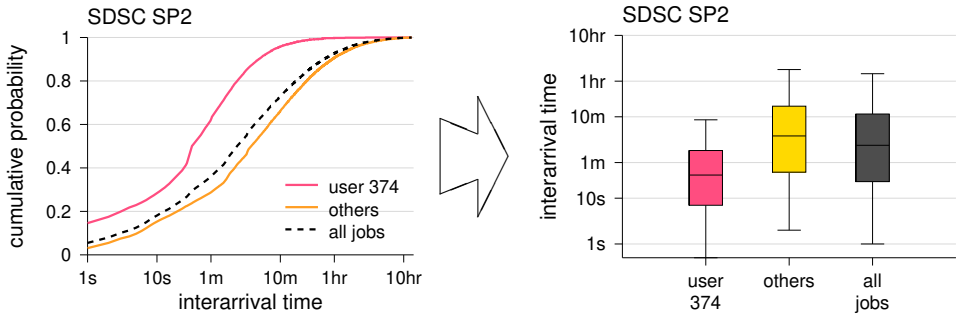
Order statistics are percentiles of the distribution. We already encountered three of them: the median is the 50th percentile, and the first and third quartiles are the 25th and 75th percentiles. In general, the p percentile is the value such that p percent of the samples are smaller than this value. Percentiles are easy to find from a distribution's CDF (Figure 3.8), and in fact serve to describe the shape of the CDF.

Practice Box: Comparing Distributions with Box Plots

Like the CDF, order statistics are also useful for comparing distributions. This can be done at a glance using box plots. Such plots describe the "body" of the distribution using a box, with whiskers that describe the tails of the distribution. Specifically, the box typically extends from the 25th percentile to the 75th percentile (i.e., the range from the first to the third quartile), and the whiskers extend to the 5th and 95th percentiles. A special mark indicates the median:



Although this is a very cursory representation of the distribution, it may be more convenient than the full CDF when making a quick comparison:



End Box

So when does one use moments or percentiles? Statistical equations and theory typically use moments, because of their mathematical properties. We'll encounter them again when we discuss distribution fitting using the method of moments in Section 4.2.2. However, with skewed distributions in which we often see some very large samples it is hard to estimate high moments, because the calculations become dominated by the largest samples we happen to see. Order statistics such as the median and other percentiles are much more stable, and therefore can be estimated with a higher degree of confidence.

3.1.5 Focus on Skew

The classic approach to describing distributions is to first use the central tendency, and possibly augment it with some measure of dispersion. This implicitly assumes a roughly symmetric distribution.

As we saw earlier, many of the distributions encountered in workload examples are not symmetric. They are skewed. This means that the right tail is much longer than the left tail. In fact, in many cases there is no left tail, and the relevant values are bounded by 0. (In principle distributions can also be skewed to the left, but in workloads this does not happen.)

Skewed distributions have, of course, been recognized for many years. An indication that a distribution is skewed is that the median is significantly different from the mean. The conventional metric for skewness is

$$\gamma_1 = \frac{\mu_3}{\sigma^3} = \frac{\frac{1}{n}\sum_{i=1}^{n}(X_i - \bar{X})^3}{\left(\frac{1}{n}\sum_{i=1}^{n}(X_i - \bar{X})^2\right)^{3/2}} \tag{3.19}$$

This may be interpreted as the weighted average of the signed distances from the mean, where large distances get much higher weights: the weights are equal to the distance squared. This is then normalized by the standard deviation raised to the appropriate

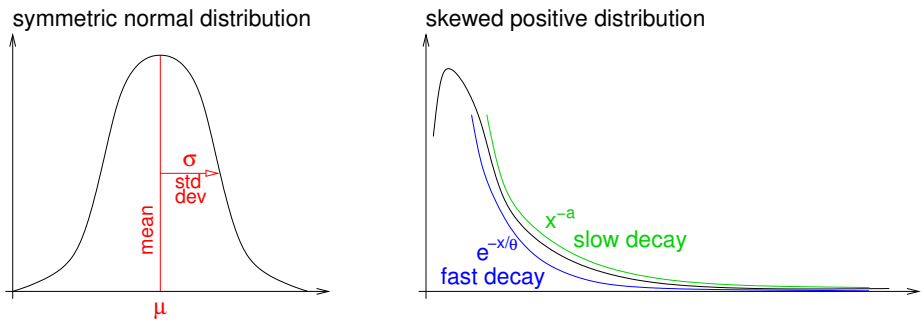


Figure 3.9: *Paradigm shift in describing skewed positive distributions: use the shape of the tail rather than the center of mass and dispersion.*

power. Positive values of γ_1 indicate that the distribution is skewed to the right, and negative values indicate that it is skewed to the left.

The definition of skewness still retains the notion of a central mode, and characterizes the asymmetry according to distances from the mean. An alternative approach suggested by the Faloutsos brothers is to forgo the conventional metrics of central tendency and dispersion, and to focus exclusively on the shape of the tail [226]. For example, as we will see later in Chapter 5, an important class of skewed distributions have a heavy tail, which is defined as a tail that decays polynomially. The exponent of the decay can then be used to characterize the distribution as a whole (Figure 3.9). In particular, such a polynomially decaying tail is completely different from an exponentially decaying tail.

Other alternatives also exist. For example, the reason that heavy tails are so important is that a significant fraction of the mass of a heavy-tailed distribution is spread along the tail, as opposed to the more normal distributions where most of the mass is concentrated around the center. The fact that so much mass is concentrated in rare large samples leads to the mass-count disparity phenomenon, which can be quantified using the joint ratio. This is a generalization of the Pareto principle (e.g., that 80% of the effect come from 20% of the events). Thus the joint ratio may also be used to characterize a distribution in a way that is more meaningful than a central tendency and a dispersion.

The joint ratio can also be combined with novel metrics that replace central tendency and dispersion. An example is given in Figure 3.10, which shows data about the distribution of file sizes in Unix systems from 1993. This distribution can be characterized by the following three main attributes:

- The joint ratio is 11/89, meaning that 11% of the files account for 89% of the disk space, and vice versa.
- The joint ratio occurs at a file size of 16 KB. This serves as a location parameter similar to the geometric average. It strikes a balance between the typical file sizes, which are around 0.5–10 KB, and the file sizes that account for most of the disk space, which are around 100 KB to 10 MB.
- The median-median distance ("m-m dist" in the figure) is a factor of 490. This serves as a metric for dispersion similar to the multiplicative standard deviation.

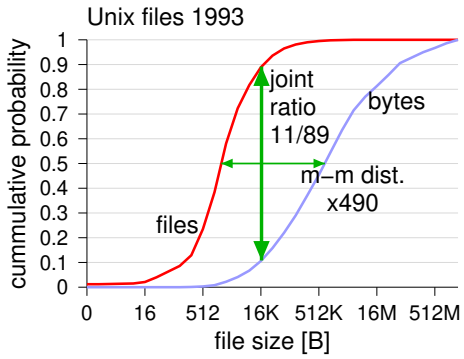


Figure 3.10: *Characterizing skewed distributions using metrics for mass-count disparity.*

It gives the ratio between where most of the mass is (around 750 KB) and where most of the files are (around 1.5 KB).

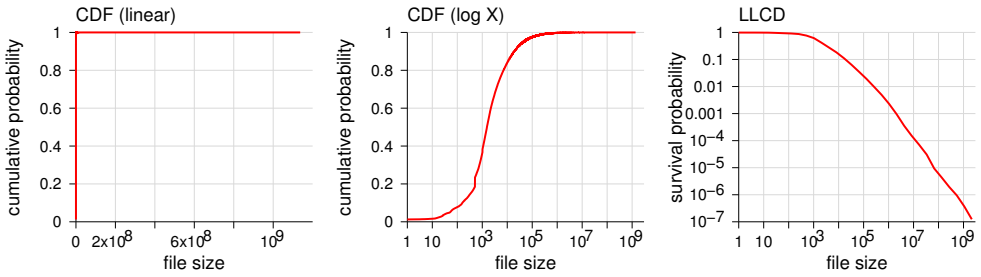
Mass count disparity and the associated metrics are discussed in detail in Section 5.2.2.

Practice Box: Looking at Skewed Data

Highly skewed data presents special challenges when one simply wants to look at it. Using a simple histogram or CDF often doesn't work, because all the ink is found to be concentrated along the axes.

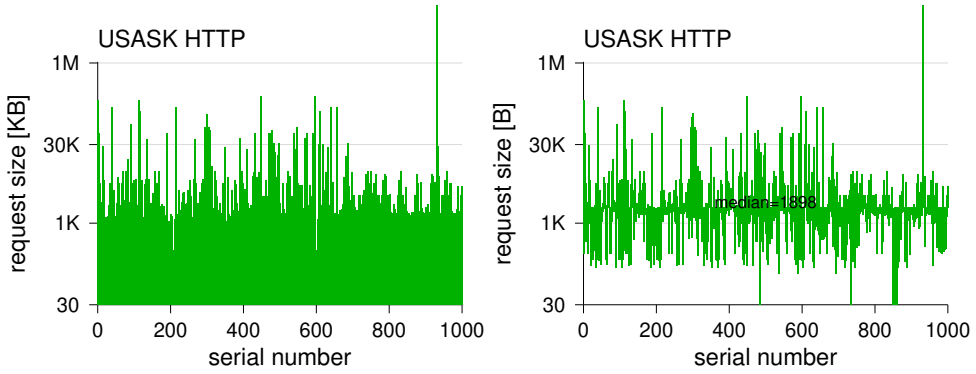
One solution is to draw the histogram or CDF with logarithmic axes. This was demonstrated above in the boxes on pages 81 and 84.

Still, at high values the histogram bins are all close to 0, and the CDF is close to 1. It may therefore be useful to use some additional mechanism to focus on the small differences that remain. One type of graph that does so is the log-log complementary distribution plot (LLCD). This shows the difference between the CDF and 1 on log-log axes. For example, the following graphs show the CDF of Unix file sizes from 1993 and the corresponding LLCD:



The Y axis of the LLCD is marked "survival probability". This is because the LLCD shows the complementary distribution (if the CDF is denoted by $F(x)$ then the LLCD shows $\bar{F}(x) = 1 - F(x)$). Thus it actually shows the probability of observing a value larger than x. LLCDs are explained in depth in Section 5.3, and LLCDs that appear to be a straight line as in this case are an indication of a heavy tail — the subject of Chapter 5.

When looking at a time series of skewed data, even in log scale, the high values tend to hide the small ones. An example is shown below, based on the request sizes from the University of Saskatchewan HTTP log of 1995. A possible solution is to show the values relative to the median rather than relative to 0 [315]. By doing so high values above the median are shown separately from low values below the median, and it is easy to see clustering of both types of values. The median is used rather than the average because it is less sensitive to extreme values.



End Box

3.2 Some Specific Distributions

Distributions of workload parameters typically have the following properties:

- The values are positive. There is no such thing as a negative runtime or negative amount of memory.
- The distribution is skewed: there are many small values and few large values.

In this section we introduce several mathematical distributions that have these properties and can therefore serve as good models.

The mathematical expressions used typically have various parameters. These parameters can be classified as the following:

- Location parameters, which specify where along the numbers line the distribution is situated (e.g., the location of the mean or the largest mode). In many cases the distribution does not have a built-in location parameter, but one can always be added. For example, if a distribution $f(x)$ has its mode at x_0, then we can create a shifted distribution $f(x-c)$ that has its mode at x_0+c.
- Scale parameters, which specify how far the distribution is spread out (e.g., the standard deviation). Again, if the distribution does not have such a parameter, it can be added: given a distribution $f(x)$, the scaled distribution $f(x/c)$ is stretched by a factor of c.

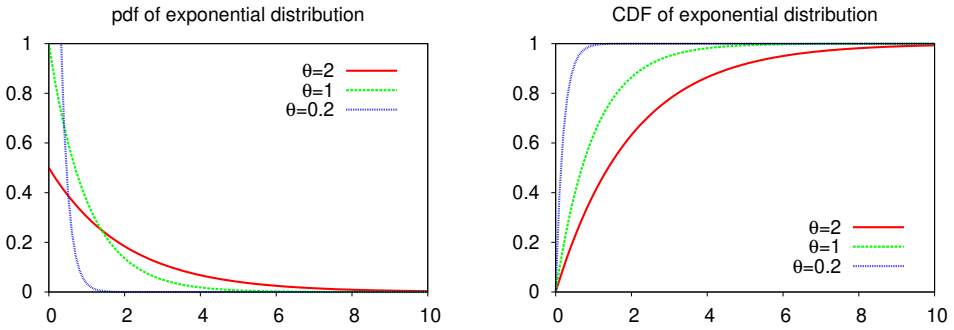


Figure 3.11: *The exponential distribution. The smaller θ is, the faster the tail decays.*

- Shape parameters, which specify the shape of the distribution. In our case these typically determine whether the distribution has a mode or not, or how heavy the right tail is.

To read more: We only give a small sampling of distributions here. There are, in fact, many more distributions, some of which also have the properties we are looking for. Additional information regarding statistical distributions and their properties can be found in books on performance evaluation, such as Jain [367, chap. 29] or Law and Kelton [427, chap. 8]. There are also complete books devoted to distributions, such as Evans et al. [221]. A similar collection available online is the compendium by McLaughlin [481].

3.2.1 The Exponential Distribution

Definition

The exponential distribution (or, rather, the negative exponential distribution) is defined by the pdf

$$f(x) = \frac{1}{\theta}\, e^{-x/\theta} \qquad x \geq 0 \tag{3.20}$$

and the CDF

$$F(x) = 1 - e^{-x/\theta} \qquad x \geq 0 \tag{3.21}$$

These functions are shown in Figure 3.11. θ is a scale parameter that determines how quickly the probability decays; in effect, x is measured in units of θ. θ is also the mean of the distribution, and satisfies $\theta > 0$.

An alternative form of the definition uses $\lambda = 1/\theta$ as the parameter. The pdf is then $f(x) = \lambda\, e^{-\lambda x}$ and the CDF is $F(x) = 1 - e^{-\lambda x}$. λ is interpreted as a rate parameter: it measures how many things happen per unit of x. In the following, we will use the two forms interchangeably as convenient.

Properties

The exponential distribution has several important properties. Let us focus on three related ones: that the interarrival times of a Poisson process are exponentially distributed,

that the exponential distribution is memoryless, and that interleaving Poisson processes leads to a new Poisson process.

In a nutshell, a *Poisson process* is one in which events occur uniformly and independently. More formally, consider a period of time T during which events occur at an average rate of λ events per time unit. We say that this is a Poisson process if the period T can be partitioned into very many equal small intervals such that the following properties hold:

1. There is no more than a single event in each interval. This excludes bursts of several events occurring at exactly the same time. Of course, many intervals will have no events in them.
2. The probability of having an event is the same for all intervals. In other words, the events are uniformly distributed in time.
3. The existence of an event in a specific interval is independent of whatever happens in other intervals.

Note that we have (on average) λT events in a period of T time units. This implies that as we divide T into more and more intervals of smaller and smaller size, the probability of seeing an event in any given interval must shrink. In fact, this probability is proportional to the lengths of the intervals.

What is the distribution of the time intervals between successive events? The probability that the time until the next event exceeds t is the same as the probability that there will be no events in t time units. Let us define the random variable $N(t)$ to be the number of events that occur in t time units. Given that the average rate is known to be λ, we know that the expected value of $N(t)$ is λt. We want to know the distribution of $N(t)$, and specifically, the probability that $N(t) = 0$. To find this, we divide our period of t time units into n small intervals. The probability of an event occurring in each interval is then $p = \lambda t/n$. The probability that a total of k events occur is

$$\begin{aligned}
\Pr(N(t) = k) &= \binom{n}{k} p^k (1-p)^{n-k} \\
&= \frac{n!}{k!(n-k)!} \, \frac{(\lambda t)^k}{n^k} \left(1 - \frac{\lambda t}{n}\right)^{n-k} \\
&= \frac{(n-k+1)\cdots n}{k!} \, \frac{(\lambda t)^k}{n^k} \left(1 - \frac{\lambda t}{n}\right)^{n} \left(\frac{n}{n-\lambda t}\right)^{k} \\
&= \frac{(n-k+1)\cdots n}{(n-\lambda t)^k} \, \frac{(\lambda t)^k}{k!} \left(1 - \frac{\lambda t}{n}\right)^{n}
\end{aligned}$$

As n tends to infinity the first factor tends to 1, and the last one tends to $e^{-\lambda t}$. We therefore find that $N(t)$ has a Poisson distribution:

$$\Pr(N(t) = k) = \frac{(\lambda t)^k}{k!} \, e^{-\lambda t} \qquad \text{for } k = 0, 1, 2, \ldots$$

Equating k to 0, we find that the probability for zero events in t time units is $\Pr(N(t) = 0) = e^{-\lambda t}$.

Let us call the random variable denoting the time until the next event X. We just found that $\Pr(X > t) = e^{-\lambda t}$. This means that $\Pr(X \leq t) = 1 - e^{-\lambda t}$. But this is exactly the CDF of the exponential distribution. Thus X is exponentially distributed with parameter $\theta = 1/\lambda$. In terms of workload modeling, this means that if we create events with exponentially distributed interarrival times, we get a Poisson process.

Note that item 2 in the list of properties defining a Poisson process implies that events are spread uniformly over the period T. We have therefore also just shown that the intervals between uniform samples are exponentially distributed.

The second important property of the exponential distribution is that it is *memoryless*. This is easily explained by reference to Poisson processes. Consider a situation in which you are waiting for the next event in a Poisson process. Time is divided into numerous little intervals, and the probability that an event occurs in any of them is uniform and independent of what happened in other (previous) intervals. Just after an event occurs, you expect the wait time until the next event to be $1/\lambda$, the mean of the exponential inter-event times. But because of the independent and uniform probability for events at each time interval, this stays the same as time goes by! For example, if λ is three events per minute, you initially expect to have to wait 20 seconds. After 10 seconds have gone by with no event, you still expect to wait another 20 seconds. If no event has occurred for 5 minutes, you still expect another 20 seconds. The distribution of how much you will have to wait is independent of how much you have already waited — it forgets the past.

More formally, what we are doing is looking at the tail of the distribution, and asking about the distribution of the tail. Given an exponential random variable X, and a threshold value τ, we are interested in the distribution of X given that we know that $X > \tau$. This is given by the conditional probability $\Pr(X \leq \tau + x \mid X > \tau)$. For the exponential distribution we get

$$\begin{aligned}\Pr(X \leq \tau + x \mid X > \tau) &= \frac{\Pr(X \leq \tau + x) - \Pr(X \leq \tau)}{\Pr(X > \tau)} \\ &= \frac{1 - e^{-(\tau+x)/\theta} - (1 - e^{-\tau/\theta})}{e^{-\tau/\theta}} \\ &= 1 - e^{-x/\theta}\end{aligned}$$

So the distribution of the tail is the same as the distribution as a whole! The exponential distribution is the only one with this property.

The third related property is that interleaving multiple Poisson processes creates a new Poisson process. This is a direct result from the above derivations. For example, if we have three Poisson processes with rates λ_1, λ_2, and λ_3, we can still divide time into ever finer intervals, and claim that the three properties of a Poisson process listed above hold. The only difference from a single process is that now the probability of an event

in an interval T is $(\lambda_1 + \lambda_2 + \lambda_3)T$. This is a Poisson process with a rate that is the sum of the rates of the constituent processes.

Finally, we note that the exponential distribution is the continuous version of the discrete geometric distribution. In workload models based on Markov chains, the number of consecutive steps during which the model remains in the same state is geometrically distributed. When transition probabilities are low and very many time steps are considered, this tends to the exponential distribution.

Uses

A major use of the exponential distribution in workload modeling is to model interarrival times. As we saw, this leads to a Poisson process, in which arrivals are independent and uniformly distributed. This both seems like a reasonable assumption and leads to simple mathematical analysis, due to the memoryless property. However, analysis of real workloads often reveals that arrivals are not uniform and independent, but rather come in bursts at many time scales. This phenomenon, known as self-similarity, is covered in Chapter 7. Thus one should consider carefully whether to use exponential interarrival times. Doing so may be justified only if arrivals can be argued to be independent.

The exponential distribution has also been used to model service times. This has less justification, except for the fact that it simplifies the analysis and leads to simple closed-form solutions for many queueing problems (e.g., the M/M/1 queue and others [367, chap. 31]). However, analysis of real workloads often shows that service times are heavy-tailed, a phenomenon discussed in Chapter 5. It is therefore preferable to use other distributions, such as phase-type distributions, that have the right shape and are nevertheless amenable to mathematical analysis.

Note that in some cases using an exponential distribution to model service times (or lifetimes) may have a strong effect on performance evaluation results. In particular, the memoryless property that makes analysis easier also precludes systems that try to learn about their workload and classify it. This means that various optimizations become meaningless, and therefore cannot be evaluated. For example, consider process migration in a network of workstations. We would like to be able to identify long-living processes and preferentially migrate them, because they have the biggest effect on the load. If we assume that process runtimes are exponentially distributed, this becomes impossible. Luckily, the real distribution tends to be heavy-tailed, so long-lived processes can indeed be identified leading to large benefits from migration [435, 320]. Another example comes from garbage collection. Generational garbage collectors preferentially scan newly allocated objects, based on the premise that most objects have short lifetimes. But this strategy does not work if object lifetimes are exponentially distributed [45].

So when is the exponential distribution clearly justified? Take radioactive decay as a mechanistic example. Define $g(t)$ to be the relative decay during time t. This means that if we start with an amount of material x, we will have $xg(t)$ left after time t. If we wait an additional s time, we will have $(xg(t))g(s)$ left. But this should be the same as what is left if we wait $t+s$ time to begin with, namely $xg(t+s)$. So the function g must satisfy $g(t+s) = g(t)g(s)$. The only function with this property is the exponential:

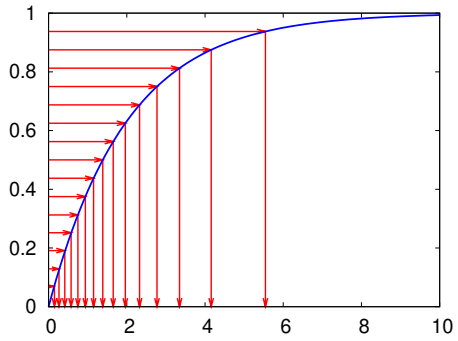


Figure 3.12: *Transforming a uniform distribution into another distribution using the inverse of the CDF.*

$g(t) = e^{-t/\theta}$. In computer systems, this reasoning leads to the use of the exponential distribution to model time to failure, under the assumption that there is no mechanical wear. In terms of workloads it is harder to find an analog of this situation. However, if you encounter a situation in which a value may be split into two terms, such that the probability of the sum is the same as the product of the probabilities of the terms, an exponential distribution should be used.

Generation

A major use of workload models is to create a synthetic workload. When the model specifies that certain distributions be used, it is then necessary to generate random variates from these distributions. Computers typically provide random number generators that create a random number from a uniform distribution on $[0, 1)$. This then has to be converted to the desired distribution.

The conversion is actually quite simple, because a distribution's CDF provides a one-to-one mapping to the uniform distribution. Define F^{-1} to be the inverse of the CDF F (i.e., $F^{-1}(F(x)) = x$). The claim is that if we select values u uniformly on $[0, 1)$, and calculate $x = F^{-1}(u)$ for each one, we will get xs that are distributed according to F. This is illustrated in Figure 3.12.

More formally, let us define a uniform random variable U, and apply the function F^{-1} as suggested above to create a random variable $X = F^{-1}(U)$. To find the distribution of X we need to evaluate $\Pr(X < x)$. Considering F^{-1} as just another function that operates on values, let us select a value u and denote its mapping as $x = F^{-1}(u)$. Then

$$\Pr(X < x) = \Pr(X < F^{-1}(u))$$

The function F, being a CDF, is monotonic, and so is F^{-1}. Therefore

$$\Pr(X < F^{-1}(u)) = \Pr(U < u)$$

But U is uniformly distributed on $[0, 1)$, so $\Pr(U < u) = u = F(x)$. In short, we found that $\Pr(X < x) = F(x)$, which means that the distribution of X is F, as desired.

In the case of the exponential distribution, the conversion is very simple. Writing $u = F(x) = 1 - e^{-x/\theta}$, we can invert F to obtain $x = -\theta \ln(u)$. (Note that if u is uniformly distributed on $[0, 1]$, so is $1 - u$.) We can therefore use the following two-step procedure:

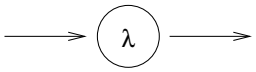
1. Select a value u uniformly from the interval $[0, 1]$. This is what random number generators typically give. (If your random number generator provides random integers from the set $\{0, 1, 2, \ldots, M\}$, simply divide them by M, the maximal possible value.)
2. Apply the transformation $x = -\theta \ln(u)$ to obtain x. Use this value.

By repeating this procedure many times we will obtain values that are exponentially distributed with parameter θ.

3.2.2 Phase-Type Distributions

Definition

Assuming that we have a server with exponentially distributed service times, we can generate exponential random variables using simulation. We simulate a client arriving at the server, record its service time, and repeat the process. Graphically, we denote such a server by a circle, with its average service rate written in it:



This idea can be generalized by postulating a network of interconnected servers, and recording the time required by the clients to traverse the network. The resulting distributions are called *phase-type* distributions, because they are composed of multiple exponential phases.

An equivalent definition that is sometimes used is the following. Consider a continuous-time Markov chain with one absorbing state, structured so that the absorbing state is guaranteed to be reached eventually. The time until absorption is then given by a phase-type distribution.

Different structures of the network of servers (or of the Markov chain) lead to distributions with different properties, and various structures have indeed been proposed, analyzed, and used (e.g. [341]). For example, the servers can have different rates, clients can enter "directly into the middle" of the network of servers, the network can have loops in it, and so on. However, several simple variants are usually sufficient to provide good approximations of empirical distributions. These are the hyper-exponential distribution, the Erlang distribution, and the hyper-Erlang distribution, which is actually a generalization of the previous two.

Uses

Phase-type distributions can be used in practically any situation, because they can be designed to be a close approximation of any empirical dataset. In particular, hyper-

exponential distributions can be designed to capture the details of the tail of a distribution, which is very important for reliable performance evaluations (see the discussion in Section 4.4.3).

Moreover, in the context of queueing networks, using phase-type distributions leads to models that can be solved analytically using the matrix-analytic method [571]. In these models, the "real" servers in the system being studied are represented by networks of exponential servers that together lead to the desired service-time distribution; the transitions between them are made by networks that create the desired interarrival distribution. This is an important extension over models that only use the exponential distribution.

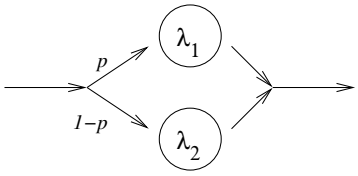
Generation

Generating random variables with a phase-type distribution is simple: we just simulate the desired network of servers, generating and summing exponential service times along the way as needed.

3.2.3 The Hyper-Exponential Distribution

Definition

The hyper-exponential distribution is obtained by selecting from a mixture of several exponential distributions. The simplest variant has only two stages:



This means that each client either gets a service time from an exponential distribution with rate λ_1, which happens with probability p, or else it gets a service time from an exponential distribution with rate λ_2 (with probability $1-p$). Naturally, λ_1 should be different from λ_2. In the general case (with k stages) the pdf is

$$f(x) = \sum_{i=1}^{k} p_i \, \lambda_i \, e^{-\lambda_i x} \tag{3.22}$$

and the CDF

$$F(x) = 1 - \sum_{i=1}^{k} p_i \, e^{-\lambda_i x} \tag{3.23}$$

where $\sum_{i=1}^{k} p_i = 1$ and $p_i > 0$ for $i = 1 \ldots k$.

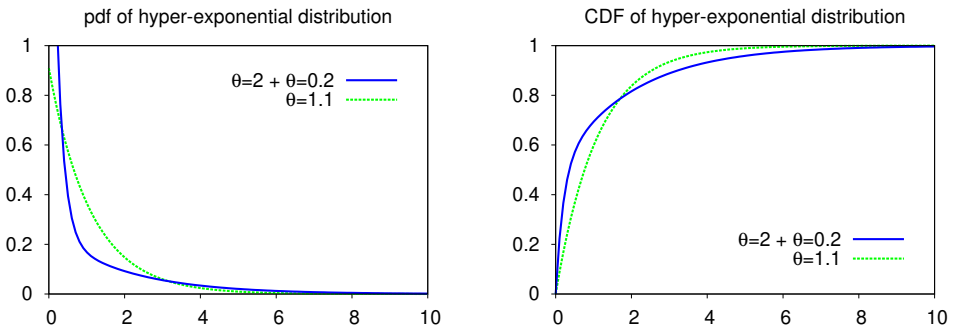


Figure 3.13: *Two-stage hyper-exponential distribution compared with an exponential distribution with the same mean.*

Properties and Uses

The main difference between the hyper-exponential distribution and the exponential distribution is that the variance of the hyper-exponential distribution is larger relative to its mean. This is often stated in terms of the CV. For the exponential distribution, both the mean and the standard deviation are equal to the parameter θ (or $1/\lambda$), so the CV is identically 1. For a hyper-exponential distribution, the CV is larger than 1. It is thus preferred over the exponential distribution if the empirical data has a large CV.

Because many distributions in workload modeling are highly skewed, the hyper-exponential distribution is quite popular in performance evaluation studies (e.g. [580, 146, 717]). In many cases it is used not to model the whole distribution, but just its tail. The hyper-exponential distribution is especially useful for such modeling because it is based on exponential phases, and is thus suitable for Markovian modeling. In addition, several techniques have been devised to match hyper-exponential distributions to empirical data. We review two such techniques in Section 4.4.3.

Figure 3.13 shows an example of a hyper-exponential distribution composed of two exponential distributions with equal probabilities (that is, $p = 0.5$). For comparison, an exponential distribution with the same mean is also shown. The hyper-exponential distribution achieves the larger variability with the same mean by emphasizing the extremes: it has a higher probability both of very small values and of very high values.

Details Box: The CV of the Hyper-Exponential Distribution

To show that the CV of a hyper-exponential distribution is larger than 1, we start by expressing the CV as a function of the moments:

$$CV = \frac{\sigma}{\mu} = \frac{\sqrt{\mu_2' - \mu^2}}{\mu}$$

By squaring this, we find that the condition for being larger than 1 is

$$\frac{\mu_2' - \mu^2}{\mu^2} > 1 \qquad \longleftrightarrow \qquad \mu_2' > 2\mu^2$$

We will show this for the two-stage hyper-exponential distribution. Using the pdf of the hyper-exponential we can calculate the mean

$$\mu = \int_0^\infty x\, f(x)\, \mathrm{d}x = \frac{p}{\lambda_1} + \frac{1-p}{\lambda_2}$$

and the second moment

$$\mu_2' = \int_0^\infty x^2\, f(x)\, \mathrm{d}x = \frac{2p}{\lambda_1^2} + \frac{2(1-p)}{\lambda_2^2}$$

We then have

$$\begin{aligned}
\mu_2' - 2\mu^2 &= \frac{2p}{\lambda_1^2} + \frac{2(1-p)}{\lambda_2^2} - 2\left(\frac{p}{\lambda_1} + \frac{1-p}{\lambda_2}\right)^2 \\
&= 2\left(\frac{p-p^2}{\lambda_1^2} + \frac{(1-p)-(1-p)^2}{\lambda_2^2} - \frac{2p(1-p)}{\lambda_1\lambda_2}\right) \\
&= 2p(1-p)\left(\frac{1}{\lambda_1^2} - \frac{2}{\lambda_1\lambda_2} + \frac{1}{\lambda_2^2}\right) \\
&= 2p(1-p)\left(\frac{1}{\lambda_1} - \frac{1}{\lambda_2}\right)^2 \\
&\geq 0
\end{aligned}$$

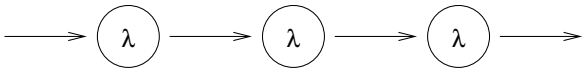
Therefore the CV is indeed always greater than or equal to 1. It is equal only in the degenerate case when $\lambda_1 = \lambda_2$ (or, equivalently, $p = 0$ or $p = 1$), and we are actually dealing with an exponential distribution.

End Box

3.2.4 The Erlang Distribution

Definition

The Erlang distribution is obtained by summing several exponential service times:



This means that each client passes through several servers in sequence. Note that all the λs are equal to each other. In the general case of k stages, the pdf is

$$f(x) = \frac{(\lambda x)^{k-1}}{\frac{1}{\lambda}(k-1)!}\, e^{-\lambda x} \tag{3.24}$$

and the CDF is

$$F(x) = 1 - e^{-\lambda x} \sum_{i=0}^{k-1} \frac{(\lambda x)^i}{i!} \tag{3.25}$$

The Erlang distribution is actually a special case of the hypo-exponential distribution, which has the same structure, but with different λs in the different stages.

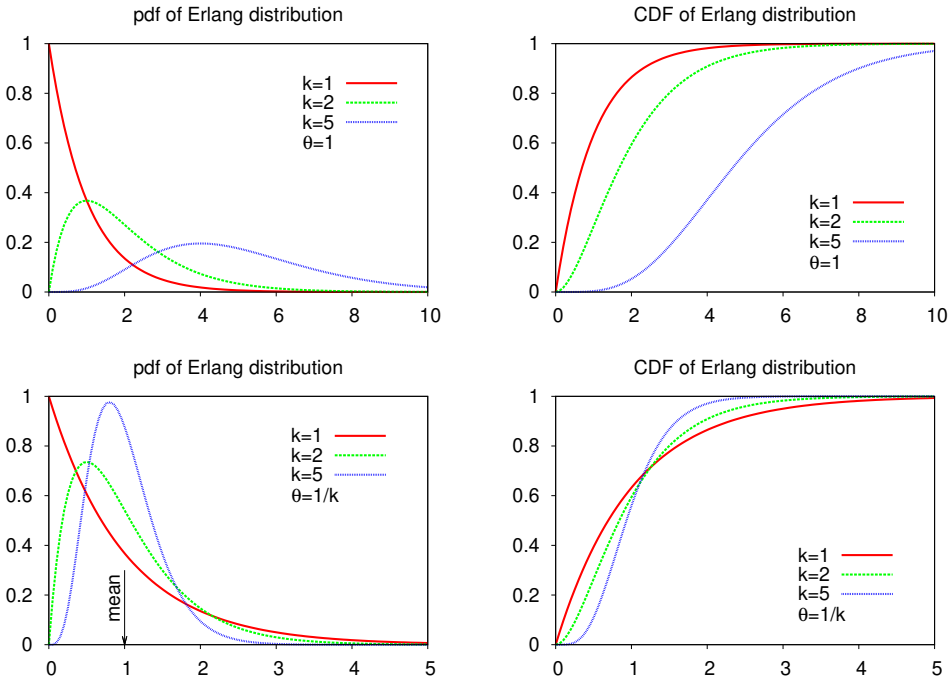


Figure 3.14: *The Erlang distribution. $k = 1$ is actually the exponential distribution. In the top graphs $\theta = 1$, and the mean grows with the number of stages. In the bottom ones $\theta = 1/k$, so the mean stays constant at 1. Additional stages then make the distribution more concentrated about the mean.*

Properties and Uses

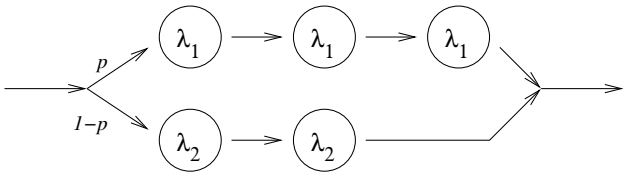
The main difference between the Erlang distribution and the exponential distribution is that the standard deviation of the Erlang distribution is smaller relative to its mean. Intuitively, when we sum several random values, the deviations from the mean tend to cancel out. Formally, this results from the fact that the standard deviation of the sum of two independent and identically distributed random variables is $\sqrt{2}$ the standard deviation of only one, whereas the mean of the sum is twice the mean of one. The quotient is therefore smaller by a factor of $1/\sqrt{2}$. As the CV of the exponential distribution is 1, the CV of the Erlang distribution is smaller than 1, and it becomes smaller as more stages are added (in fact, the CV of a k-stage Erlang distribution is $1/\sqrt{k}$). It is thus preferred over the exponential distribution if the empirical data has a small CV.

Figure 3.14 shows an example of Erlang distributions with an increasing number of stages. When the number of stages is larger than 1, the distribution has a mode, rather than being monotonically decreasing like the exponential and hyper-exponential distributions.

3.2.5 The Hyper-Erlang Distribution

Definition

The hyper-Erlang distribution is a mixture of several Erlang distributions, just as the hyper-exponential is a mixture of several exponentials:



Note that the number of stages in each Erlang distribution may be different from the others. If they are all the same, it is called a hyper-Erlang distribution of common order. The pdf of the general case is

$$f(x) = \sum_{i=1}^{k} p_i \frac{(\lambda_i x)^{k_i-1}}{\frac{1}{\lambda_i}(k_i-1)!} e^{-\lambda_i x} \tag{3.26}$$

and the CDF

$$F(x) = 1 - \sum_{i=1}^{k} p_i \left(e^{-\lambda_i x} \sum_{j=0}^{k_i-1} \frac{(\lambda_i x)^j}{j!} \right) \tag{3.27}$$

(note the notation: k is the number of Erlang stages in the hyper-Erlang distribution, and k_i is the number of exponential stages in the ith Erlang).

Properties and Uses

Because each Erlang distribution is more localized than an exponential distribution, the hyper-Erlang construction enables better control over the shape of the resulting distribution. In particular, it enables the construction of a multimodal distribution, in which the pdf has multiple ridges and valleys (the hyper-exponential distribution, in contradistinction, has a monotonically decreasing pdf). With too few components, the distribution tends to be modal even if this was not intended. Thus good control over the shape comes at the expense of having a large number of components, and therefore a large number of parameters.

An example is given in Figure 3.15, which shows two hyper-Erlang distributions, each composed of two Erlang distributions. In both cases, the mean of the first stage is 3 and the mean of the second stage is 18, and both occur with equal probabilities ($p = 0.5$). The overall mean is therefore 10.5. The difference lies in the number of stages used. When more stages are employed, the peaks become more concentrated. However, when the scale parameter of the exponentials is bigger, many more stages are required than when it is small.

Specific uses of a hyper-Erlang distribution for workload modeling include matching the first three moments of empirical distributions [370], and creating a distribution with

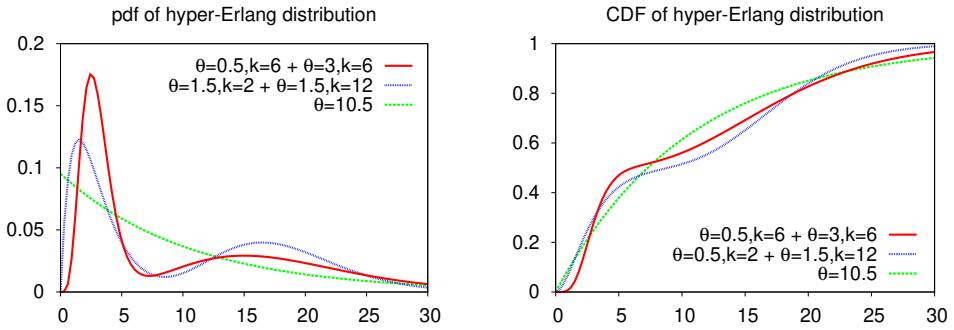


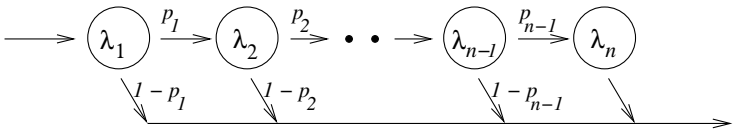
Figure 3.15: *Example of hyper-Erlang distributions compared with an exponential distribution with the same mean. The hyper-Erlangs have two stages, each with probability 0.5.*

a mode and a heavy tail (by actually combining an Erlang distribution and a hyper-exponential distribution) [568].

3.2.6 Other Phase-Type Distributions

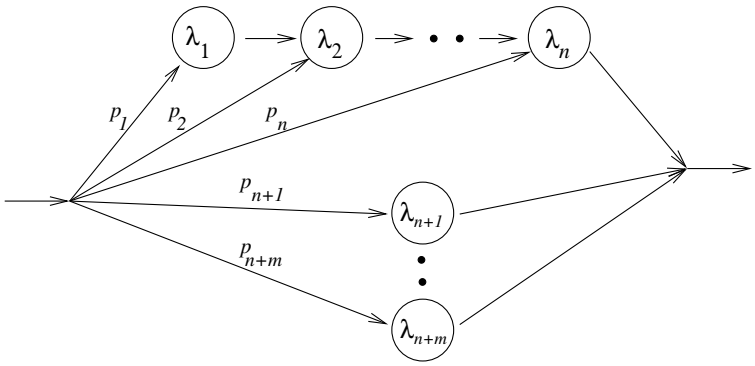
There are several additional structures of phase-type distributions that are often used. The reason for their frequent use is that it can be proved that any distribution can be approximated by these phase-type distributions arbitrarily closely.

The first is the Coxian distribution, proposed by Cox. This is a sequence of exponential stages with an "escape path" that allows subsequent stages to be skipped:



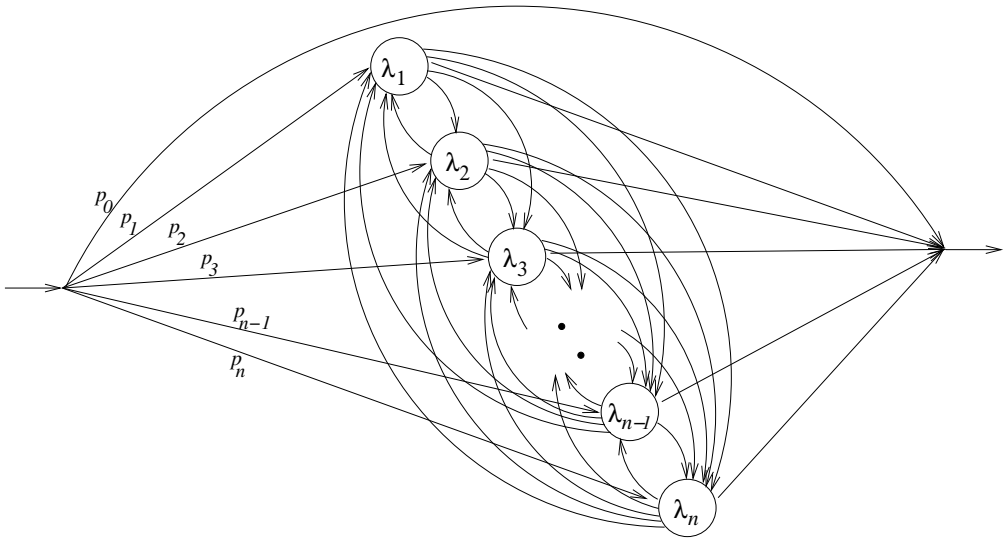
With n stages, this has $2n - 1$ parameters: the mean service times of the n servers, and the probabilities of using the escape path. A simplified version where all λs are equal has also been proposed [97]. Coxian distributions can be constructed so as to match the first three moments of the data using a minimal number of stages, thereby simplifying the model [527].

The opposite approach has also been used: instead of an escape path, use an entry path that allows you to "jump into" the sequence at a selected stage:



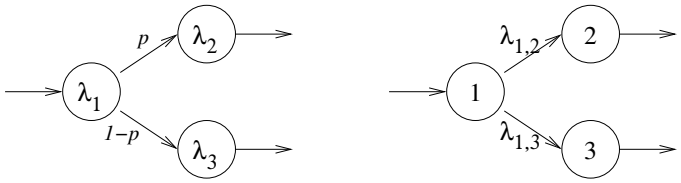
The n stages in the sequence are ordered according to their mean service rates; that is $\lambda_1 \leq \lambda_2 \leq \cdots \leq \lambda_n$. In addition, a set of m individual stages are added [342, 341]. Effectively, the sequence of the first n stages creates a mode that serves as the body of the distribution, and the additional individual stages are actually a hyper-exponential distribution used to construct a long tail for the distribution. This approach emphasizes matching the distribution's shape rather than its moments.

All the above are special cases of the general phase-type distribution, which includes all the possible transitions between all the stages. In each special case, only a select subset of transitions is used, and the rest have probability 0. Note that in the general case a direct transition that bypasses all the stages is also included, and such a bypass can be included in the special cases too. It adds the value 0 with probability p_0.



Details Box: Parameterizing Phase Type Distributions

In the preceding graphical representations of phase-type distributions, each exponential stage is represented by its service rate, and the transitions from one stage to subsequent ones are represented by probabilities. An alternative formulation is to number the stages and specify the rates of the transitions:



The two formulations are equivalent. The service rate of a stage is the sum of the transition rates out of it, and the transition rates are the overall rate multiplied by the relevant probabilities. Thus in the example given above we have

$$\begin{aligned}\lambda_1 &= \lambda_{1,2} + \lambda_{1,3} \\ \lambda_{1,2} &= p\,\lambda_1 \\ \lambda_{1,3} &= (1-p)\,\lambda_1\end{aligned}$$

To see this, consider the approach we used when discussing Poisson processes. Divide time into many very small intervals, so that each transition happens in a distinct interval (while of course in most intervals nothing happens). Denote by $p_{1,2}$ the probability of transitioning from state 1 to state 2, and by $p_{1,3}$ the probability of transitioning from state 1 to state 3. Then obviously the probability of transitioning out of state 1 is $p_{1,2} + p_{1,3}$. The relationships between the rates follow.

End Box

3.2.7 The Normal Distribution

Definition

The pdf of the normal distribution is

$$f(x) = \frac{1}{\sqrt{2\pi}\,\sigma}\, e^{-\frac{(x-\mu)^2}{2\sigma^2}} \qquad -\infty < x < \infty \tag{3.28}$$

where μ is the mean and σ the standard deviation. There is no closed form for the CDF.

Properties

The normal distribution is the omnipresent "bell curve" of popular science, so called in reference to the shape of its pdf (the CDF is a sigmoid). It is of central importance in probability and statistics, being the unique stable distribution that is the limiting distribution when summing random variables with a finite variance. This is essentially the central limit theorem.

In plain English, the above sentence means the following. Consider a sequence of random variables $X_1, X_2, \ldots, X_n$, which are independent and identically distributed. Now define a new random variable Y, which is their average:

$$Y = \frac{1}{n}\sum_{i=1}^{n} X_i$$

The central limit theorem states that, as n tends to infinity (that is, the number of Xs grows), and provided that the Xs come from a distribution with finite variance, Y will

come to have a normal distribution. Moreover, the normal distribution is the only distribution with this property. And because the normal distribution has a finite variance, a special case is that this also works if you sum normal random variables. This is why the distribution is called "stable".

Uses

The central limit theorem is very important and useful in statistics. Any effect that is the combination of multiple independent random effects can be expected to have a normal distribution. However, this hinges on the assumption that the summed effects have a finite variance. The distributions mentioned earlier, — the exponential and phase-type distributions — indeed have this property. However, some of the distributions described below do not. (Such heavy-tailed distributions are discussed at length in Chapter 5.) Since heavy-tailed distributions (or at least strongly skewed distributions) are ubiquitous in workloads, the central limit theorem may not apply. In addition, the symmetrical and short-tailed normal distribution is inappropriate for matching skewed data directly. As a result the normal distribution is rarely used in workload modeling.

Generation

Since there is no closed-form expression for the normal distribution's CDF, we cannot use the technique of inverting the CDF. The common way to generate standard normal variates (with mean 0 and standard deviation 1) is to do so in pairs [581, sect. 11.3.1]. Start with a pair of uniform variates, u_1 and u_2. Now use them to generate a pair of normal variates by computing

$$n_1 = \sqrt{-2 \ln u_1}\ \sin(2\pi u_2)$$
$$n_2 = \sqrt{-2 \ln u_1}\ \cos(2\pi u_2)$$

The justification for this somewhat strange formula is as follows. The sum of the squares of two normal random variables has distribution χ^2 with two degrees of freedom. The square root of this has an exponential distribution. Using polar coordinates, the formula selects a random point on the 2D plane with an exponentially distributed radius (distance from the origin) and a uniformly distributed angle. Converting to Cartesian coordinates then leads to two normal variates.

Given a standard normal variate n, a normal variate with mean μ and standard deviation σ is obtained by

$$x = \mu + \sigma n$$

3.2.8 The Lognormal Distribution

Definition

The lognormal distribution is a normal distribution in log-space — in other words, if we take the logarithm of the data values, they will have a normal distribution. This is evident from its pdf, which is based on the pdf of the normal distribution:

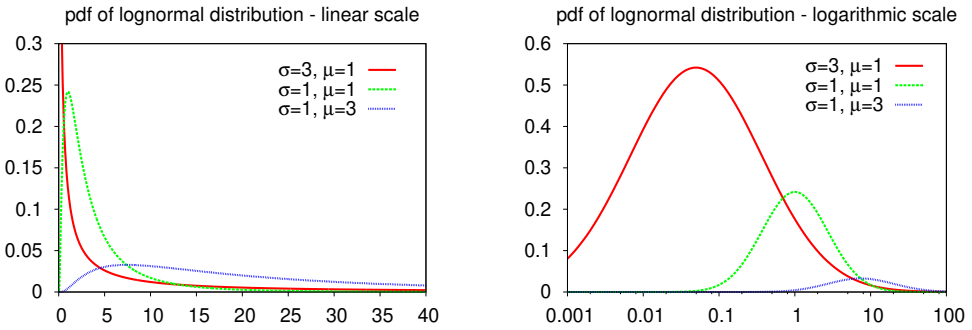


Figure 3.16: *Examples of the lognormal distribution with different parameters. The graph on the right uses a logarithmic scale, showing the normal bell shape.*

$$f(x) = \frac{1}{x\sigma\sqrt{2\pi}} e^{-\frac{(\ln x - \mu)^2}{2\sigma^2}} \qquad x \geq 0 \tag{3.29}$$

There is no closed form for the CDF.

Note that μ and σ are the mean and standard deviation in log-space — not the mean and standard deviation of the distribution. A good characterization of the distribution itself is obtained by using the geometric mean and the multiplicative standard deviation (as described on page 100) [445].

Properties

The shape of the lognormal distribution is modal and skewed, like that of the gamma and Weibull distributions (discussed later) when their shape parameters satisfy $\alpha > 1$. But when viewed in a logarithmic scale, it displays the bell shape of the normal distribution (Figure 3.16). This property has made it quite popular. It is standard practice to subject skewed data with a long tail to a logarithmic transformation and to examine it in log-space. If the result is modal and symmetric, like a normal distribution, it is natural to assume a lognormal model. Moreover, the parameters of the distribution have a simple intuitive meaning.

Two examples are shown in Figure 3.17. The LANL CM-5 interarrival times seem to be lognormally distributed, provided the data is cleaned by removing workload flurries (see Section 2.3.4). The file size data may also be roughly lognormal, at least for this data set, albeit there are a few large modes at small values that do not fit. Note that these histograms do not show the number of items falling within logarithmic bins, but rather the probability density: the count in each bin is divided by both the total number of observations and the bin width. As a result high values in the tail are strongly suppressed and cannot be seen in the graph. Nevertheless, these histograms do have a long right tail.

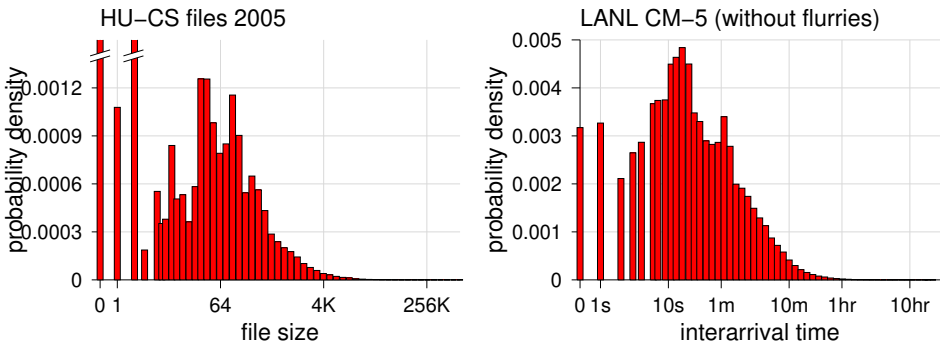


Figure 3.17: *Examples of logarithmically-transformed datasets that seem reasonable to model using a lognormal distribution.*

Uses

The lognormal distribution is similar to the exponential, gamma, and Weibull distributions in that it is positive and has a tail that extends to infinity. It is distinguished from them by virtue of matching the results of a specific mechanism of creating values. This is based on an analogy with the normal distribution.

The normal distribution is well known because of its role in the central limit theorem: if we take a large number of random variables from the same distribution and sum them up, we get a new random variable, and this new random variable is normally distributed (at least if the original distribution had a finite variance). The lognormal distribution has a similar role. If we take multiple random variables and *multiply* them by each other, the resulting random variable will have a lognormal distribution. This can easily be seen by taking the log of the product. The logarithmic transformation turns the product into a sum; the sum of the logs of the values will be normally distributed, so the original product is therefore the exponentiation of a normally distributed value and is itself lognormal.

At a more concrete level, the normal distribution may be regarded as the result of a sequence of decisions (or random effects), in which each decision adds or subtracts a certain value. The lognormal, in analogy, is the result of a similar sequence of decisions (or random effects), but here each such decision either multiplies or divides by a certain value [445].

The preceding argument has been used in a generative explanation of the distribution of file sizes to justify the claim that a lognormal model is better than a heavy-tailed one [188]. The idea is to initialize the model with a single file. Then, model the creation of additional files by selecting a file at random, multiplying its size by a factor that is also selected at random, and inserting a file with the new size. This is supposed to model editing a file, translating it (as when a compiler produces an executable from a program), or copying (the special case where the factor is 1). Thus file sizes are related

to each other via a sequence of multiplications, and can be expected to have a lognormal distribution.

Although this derivation is appealing, similar approaches may lead to alternative distributions such as the Pareto distribution described below. Details of these contending models, the generative processes that motivate them, and how to choose among them are discussed in Sections 5.4.2 and 5.4.3.

Generation

The generation of a lognormal variate is simple if you already have a standard normal variate n (that is, with mean 0 and standard deviation 1). If this is the case, compute

$$x = e^{\mu+\sigma n}$$

and use this value.

The common way to generate standard normal variates is to do so in pairs, as described above on page 122.

3.2.9 The Gamma Distribution

Definition

The gamma distribution is defined by the pdf

$$f(x) = \frac{1}{\beta\,\Gamma(\alpha)}\left(\frac{x}{\beta}\right)^{\alpha-1} e^{-x/\beta} \qquad x \geq 0 \tag{3.30}$$

where the parameters satisfy $\alpha, \beta > 0$ and the gamma function is defined as

$$\Gamma(\alpha) = \int_0^\infty x^{\alpha-1}e^{-x}\mathrm{d}x \tag{3.31}$$

This somewhat intimidating expression is actually quite straightforward. Note that the definition of the gamma function is an integral with exactly the same factors as the pdf (after scaling by a factor of β). Thus the $\Gamma(\alpha)$ in the denominator is just a normalization factor.

There is no closed-form expression for the CDF unless α is an integer. However, the CDF can be expressed using the gamma function as

$$F(x) = \frac{\displaystyle\int_0^x t^{\alpha-1}e^{-t}\mathrm{d}t}{\displaystyle\int_0^\infty t^{\alpha-1}e^{-t}\mathrm{d}t} \qquad x \geq 0$$

The denominator is the gamma function, and in the numerator the upper bound of the integration is x rather than ∞.

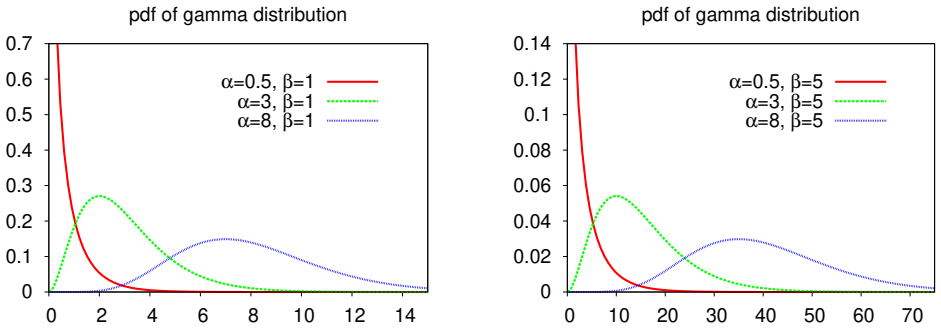


Figure 3.18: *Example of gamma distributions with different parameters.*

Note also the following interesting property of the gamma function, which can be derived by integration by parts:

$$\begin{aligned}\Gamma(\alpha+1) &= \int_0^\infty x^\alpha e^{-x}\mathrm{d}x \\ &= \frac{1}{\alpha+1}\int_0^\infty x^{\alpha+1}e^{-x}\mathrm{d}x \\ &= \frac{1}{\alpha+1}\Gamma(\alpha+2)\end{aligned}$$

So $\Gamma(\alpha+2) = (\alpha+1)\Gamma(\alpha+1)$. If α is an integer, the gamma function is therefore identical to the factorial: $\Gamma(\alpha+1) = \alpha(\alpha-1)(\alpha-2)\cdots = \alpha!$. For non-integral values, it provides a generalization of the factorial.

Properties

One reason for the interest in the gamma distribution is that it is very flexible. This means that different parameter values lead to distributions with different shapes. This flexibility is the result of having two competing terms in the pdf: a polynomial term $(x/\beta)^{\alpha-1}$ and an exponential term $e^{-x/\beta}$. In the long run, as $x \to \infty$, the exponential term always wins and $f(x) \to 0$. β is therefore called the scale parameter, and determines the spread of the distribution, or more precisely, how quickly the tail will decay. This is illustrated in Figure 3.18, where the only differences between the left and right graphs are the scale and the value of β.

α is called the shape parameter. When $\alpha \leq 1$ the pdf is a monotonically decreasing function. However, when $\alpha > 1$ the polynomial factor "wins" over a certain range, and the distribution has a hump. The peak then occurs at a value of $(\alpha-1)\beta$. For low αs it is skewed, but when α grows it becomes more symmetric. All this is illustrated in Figure 3.18.

Another reason for interest in the gamma distribution is that it is a generalization of the exponential and Erlang distributions. Specifically, when $\alpha = 1$ the gamma distribution is just an exponential distribution with parameter β, and when α is an integer k it is

equivalent to a k-stage Erlang distribution (again with parameter β). This is evident by comparing Figure 3.18 with Figure 3.14.

Uses

The gamma distribution is one of several versatile distributions that may be used to model workload parameters. It is positive, has a tail that extends to ∞, and may be adjusted to have a mode at some positive value, rather than being monotonically decreasing.

Generation

Generating random variates from a gamma distribution is rather involved. However, in the special case that α is an integer, the gamma distribution is actually an Erlang distribution (i.e., a sum of exponentials). In that case we can select α uniform random variates $u_1, \ldots, u_\alpha$, and use the value

$$x = -\beta \sum_{i=1}^{\alpha} \ln(u_i)$$

The Erlang distribution may sometimes be used as an approximation when α is not an integer. Law and Kelton provide an extended discussion of the general case [427, sect. 8.3.4].

3.2.10 The Weibull Distribution

Definition

The Weibull distribution is defined by the pdf

$$f(x) = \frac{\alpha}{\beta} \left(\frac{x}{\beta}\right)^{\alpha-1} e^{-(x/\beta)^\alpha} \qquad x \geq 0 \tag{3.32}$$

and the CDF

$$F(x) = 1 - e^{-(x/\beta)^\alpha} \qquad x \geq 0 \tag{3.33}$$

where $\alpha, \beta > 0$. Focusing on the CDF, it is seen to be closely related to that of the exponential distribution, the only difference being that the exponent is raised to the power α. This causes the X axis to be distorted in a nonlinear manner (which is why the Weibull distribution is sometimes also called a "stretched exponential"). When $\alpha < 1$ the axis is indeed stretched out, because it takes longer to arrive at high values. Conversely, when $\alpha > 1$, the axis is actually pulled in, and we arrive at high values faster.

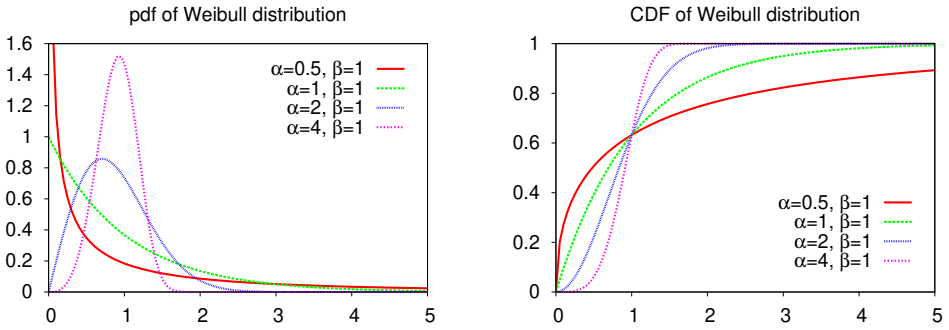


Figure 3.19: *Example of Weibull distributions with different parameter values. Note that the case $\alpha = 1$ is the exponential distribution.*

Properties and Uses

The Weibull distribution is similar to the gamma distribution in its flexibility and possible shapes, which result from a similar combination of a polynomial factor and an exponential factor. α and β have the same interpretation as shape and scale parameters. Large αs cause the mode to be more pronounced and symmetric. The location of the mode is $\left(1 - \frac{1}{\alpha}\right)^{1/\alpha} \beta$, and it exists only for $\alpha > 1$. For small αs, in contrast, the tail of the distribution becomes more pronounced. In fact, when $\alpha < 1$, the Weibull distribution is considered long-tailed, and as $\alpha \to 0$ we approach a power-law tail (long tails are discussed in Section 5.1.3). At the boundary, when $\alpha = 1$, the Weibull becomes a simple exponential.

Several examples are shown in Figure 3.19. Note that although the Weibull and gamma distributions have similar parameters, their shapes are actually quite different. In particular, for large αs the Weibull is much more concentrated in a narrow range.

Generation

By inverting the expression for the CDF of the Weibull distribution, one can see that random variates can be generated by creating a uniform random variate u and using the transformed value

$$x = \beta(-\ln(u))^{1/\alpha}$$

3.2.11 The Pareto Distribution

Definition

The Pareto distribution is defined by the pdf

$$f(x) = \frac{a\ k^a}{x^{a+1}} \qquad\qquad x \geq k \tag{3.34}$$

and the CDF

$$F(x) = 1 - \left(\frac{k}{x}\right)^a \qquad x \geq k \tag{3.35}$$

where k is a location parameter — it specifies the minimal value possible (that is, $x \geq k > 0$). a is a shape parameter that defines the tail of the distribution, commonly called the tail index, and satiafying $a > 0$. The smaller a is, the heavier the tail, meaning that there is a higher probability of sampling very large values.

Note that k is not a conventional location parameter; actually it is more of a scale parameter, because it divides x rather than being added to x. Thus changing the value of k does not only shift the distribution along the X axis, but also affects its shape (but, of course, if we look at the distribution on a logarithmic scale changing k does induce a shift). This allows us to also define a *shifted* Pareto distribution (also called a Pareto distribution of the second kind, or a Lomax distribution) with a pdf of

$$f(x) = \frac{a\ k^a}{(x+k)^{a+1}} \tag{3.36}$$

and the CDF

$$F(x) = 1 - \left(\frac{k}{x+k}\right)^a \tag{3.37}$$

Here we used a shift of k, but in general this can be an independent parameter. However, a shift of k has the nice property that now the distribution is defined for all $x \geq 0$.

The heavy tail of the Pareto distribution can cause significant difficulties for analysis and even for simulation. Therefore a *truncated* version is sometimes used. This postulates a maximal possible value t and truncates the distribution at that point. Naturally using this version requires a re-normalization. As the probability (of the original Pareto distribution) of being in the range up to t is $F(t) = 1 - (\frac{k}{t})^a$, the pdf of the truncated version is

$$f(x) = \begin{cases} \dfrac{a\ k^a}{\left[1 - \left(\frac{k}{t}\right)^a\right] x^{a+1}} & k \leq x \leq t \\ 0 & x < k, x > t \end{cases} \tag{3.38}$$

and the CDF is

$$F(x) = \frac{1 - \left(\frac{k}{x}\right)^a}{1 - \left(\frac{k}{t}\right)^a} \qquad k \leq x \leq t \tag{3.39}$$

and of course $F(x) = 0$ if $x < k$ and $F(x) = 1$ if $x > t$. This is called either a truncated or a bounded Pareto distribution.

The differences between these three versions are illustrated in Figure 3.20, where $a = 1$ and $k = 10$. In this graph we look at the distribution's survival function on log-log scales, which is called a "log-log complementary distribution" (LLCD; this is explained in detail in Section 5.3). The pure Pareto distribution has a straight LLCD, because

$$\log \bar{F}(x) = \log(x^{-a}) = -a \log x$$

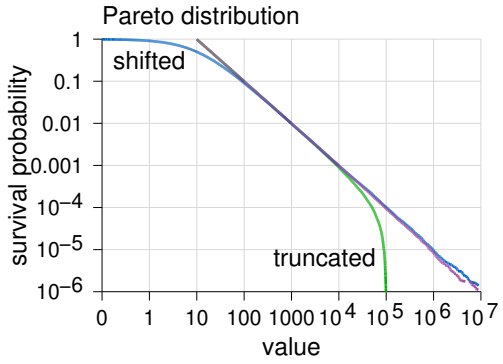


Figure 3.20: *LLCDs of variants of the Pareto distribution.*

The shifted version rounds off the top part of the plot, and makes it start from 0. The truncated version rounds off the tail of the distribution, and makes it converge asymptotically to the truncation point, which is 10^5 in this case.

Properties

The Pareto distribution has a power-law tail: the probability of seeing big values drops as x^{-a}, which is slower than the exponential drop of, say, the exponential distribution. Distributions with power-law tails are said to be "heavy-tailed". In fact, the Pareto distribution is the archetype of heavy-tailed distributions. These distributions have many important properties, and we devote all of Chapter 5 to discussing them.

In a nutshell, the main property of heavy-tailed distributions is that very large values have a non-negligible probability. In fact, a large part of the total mass of the distribution is found concentrated in these large items. This is a technical expression of the Pareto principle, also known as the 80/20 rule. It originated with the work of Vilfredo Pareto, an Italian economist, who found that 80% of the land in Italy was owned by only 20% of the population. He also found that the tail of the distribution of wealth followed a power law.

Uses

heavy-tailed distributions such as the Pareto distribution are ubiquitous in computer workloads. Examples include the distributions of process runtimes and of file sizes. In general, these are distributions of a "size" attribute of workload items. Such distributions are also common in the natural sciences, especially in geophysics (earthquake magnitudes, moon crater diameters, solar flares), and in human activity (city sizes, phone calls received, distribution of wealth, and even casualties of war) [514].

It is interesting to speculate on why power-law tails are so common. A generative explanation of how such distributions come about has been suggested for the case of wealth; essentially, it says that money begets money. Interestingly, this is very similar to

the multiplicative model that generates the lognormal distribution [497]. The idea is that if an individual i has wealth w_i, it may grow by a multiplicative factor f that comes from some distribution. But in addition, it is also limited by terms that relate to the average wealth $\bar{w}$. On the one hand, due to social security, nobody's wealth can fall below a certain fraction of $\bar{w}$. (Pareto himself put it more bluntly: he said that below a certain level people just die.) On the other hand, because of the finite size of the economy, nobody's wealth can grow without bounds. Formalizing all this leads to a model in which the w_is turn out to have a power-law distribution [462].

Similar models have been suggested for various parameters of computer systems, especially in the context of the world wide web. One such model postulates that the probability that a new page links to an existing one is proportional to the number of links the page already has [49, 9]. Another more abstract model considers the observed system state as resulting from sampling individual processes that each grow exponentially [353, 562] (or similarly, from an exponentially growing number of sources that each grows exponentially [207]).

A somewhat related approach links power-law tails with prioritization. Consider the waiting time of tasks in a priority queue. High-priority tasks will be picked immediately after entering the queue, so they will not have to wait very long. But low-priority tasks will be skipped over many times and may suffer from very long wait times. Again, with an appropriate formalization this leads to a power law [48].

The question of whether a Pareto distribution is the best model is a difficult one to answer. Other options are often viable, including the Weibull and lognormal distributions described earlier. Truncated Pareto distributions have also been used. Again, this discussion is deferred to Chapter 5, and specifically Sections 5.4.2 and 5.4.3. Section 5.4.3 also discusses derivations of the generative models mentioned above.

Generation

By inverting the CDF of the Pareto distribution, we find that a Pareto variate can be created from a uniform variate u using

$$x = \frac{k}{u^{1/a}}$$

To get a shifted Pareto variate, simply subtract k. To get a truncated one, discard variates that exceed the desired bound.

3.2.12 The Zipf Distribution

Definition

The Zipf distribution has its origins in linguistics. Start with a good book, and tabulate the number of times that each word appears in it. Now sort the words in order of popularity: the most common word first, the second most common next, and so on. Denoting the number of occurrences of the ith word by $c(i)$, you should find that $c(i) \propto 1/i$. At

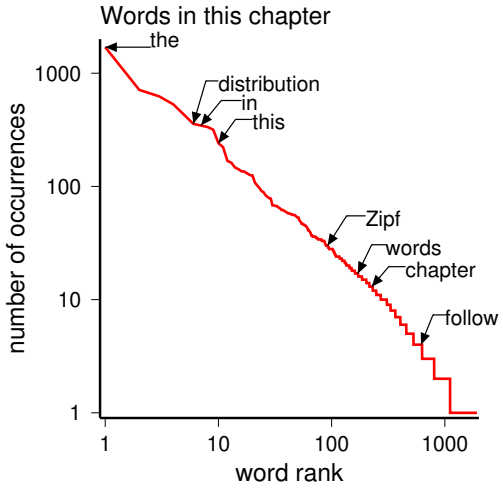


Figure 3.21: *The words in this chapter follow the Zipf distribution.*

least, that is what Zipf found[3] [764]. This result can be visualized by plotting $c(i)$ as a function of i on log-log axes (the so-called *rank-size plot*); Zipf's law implies that this should lead to a straight line with a slope of –1. As it turns out, it is also approximately true for the words in this book (at least when the sample is not too large, as in Figure 3.21).

The generalized form of the Zipf distribution (also called the "Zipf-like distribution") is defined by the pdf

$$\Pr(i) \propto \frac{1}{i^\theta} \tag{3.40}$$

where i is an index, not a value (so $\Pr(i)$ is the probability of observing the ith item, not the probability of observing a value of i), and θ is a shape parameter (e.g. [737]). Setting $\theta = 0$ leads to the uniform distribution. Setting $\theta = 1$ is the original Zipf distribution. The range $0 < \theta < 1$ is less skewed than the original Zipf distribution, whereas the range $\theta > 1$ is more skewed.

Another variant, attributed to Mandelbrot [59], adds a constant b in the denominator:

$$\Pr(i) \propto \frac{1}{i+b}$$

This has the effect of reducing the count of the high-ranking items (i.e., the $c(i)$ for low values of i) and may lead to a better fit to some datasets.

The obvious problem with regarding any of these variants as a distribution is one of normalization. For $\theta < 2$, the sum $\sum_{i=1}^{\infty} 1/i^\theta$ does not converge. Therefore it can only be defined on a finite set, with proper normalization. For $\theta = 1$ and n items, the expression becomes

[3] Although there has been active debate regarding the actual distribution of word frequencies [621, 59], Zipf's law is still a good approximation and has the advantage of being very simple.

$$\Pr(i) = \frac{1}{i \ln n}$$

Connection Box: Monkey Typing

An interesting observation is that Zipf's law appears not only in natural languages but also in random texts. This casts doubts on its importance in the context of linguistics [489, 441].

Consider an alphabet with M symbols and an additional space symbol. To generate a random text using this alphabet, a sequence of symbols is selected at random with uniform probabilities (this is sometimes called "monkey typing", based on the conjecture that monkeys are good random number generators). Words are defined to be subsequences of non-space symbols separated by the space symbol. The probability of seeing a word of length ℓ is therefore geometrically distributed:

$$\Pr(\ell) = \left(\frac{M}{M+1}\right)^{\ell} \frac{1}{M+1}$$

But the number of distinct words of length ℓ is M^{ℓ}, and they are equiprobable, so the probability of seeing any specific word of length ℓ is simply $(\frac{1}{M+1})^{\ell+1}$ — it falls off exponentially with ℓ.

If we now rank the words according to their probability, words of length ℓ will be preceded by all shorter words. There are M words of length 1, M^2 of length 2, and so on, for a total of $\sum_{i=1}^{\ell-1} M^i = \frac{M-M^{\ell}}{1-M}$. Roughly speaking, we find that as the probability of a word falls exponentially, its rank grows exponentially, and thus the probability is inversely proportional to the rank. Miller gives the exact derivation, using the average rank of all words with each length ℓ [489].

A problem with this derivation is that it assumes that all letters in the alphabet are equiprobable. Perline shows that if this is not the case, the distribution of randomly generated words is not Zipfian, but lognormal [544]. However, a very large number of samples will be needed to observe the difference from a Zipf distribution.

End Box

Properties

The Zipf distribution is actually a special case of the Pareto distribution [8, 85, 442]. Assume a set of items are ordered according to their popularity counts (i.e., according to how many times each was selected). Zipf's law is that the count c is inversely proportional to the rank i, and can be written as

$$c = C\, i^{-\theta} \tag{3.41}$$

where C is some constant that reflects the number of samples and the number of items to choose from. Being in rank i means that there are $i-1$ items with counts larger than c. Using X to denote a random variable giving the count of an item, we then have

$$\Pr(X > c) = i/n \tag{3.42}$$

where n is the total number of items. We can express i as a function of c by inverting the original expression (3.41), leading to $i \approx C\, c^{-1/\theta}$ (where C is a different constant now). Substituting this into (3.42) gives a power-law tail:

$$\Pr(X > c) = C \cdot c^{-1/\theta} \tag{3.43}$$

(C has changed again, but is still constant). Note that if $\theta \approx 1$, as in the original Zipf distribution, then $1/\theta \approx 1$ as well.

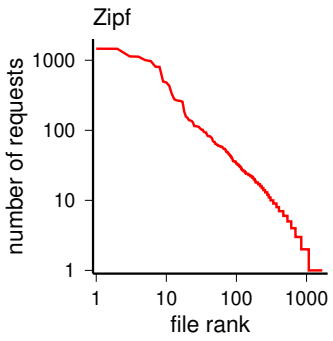
The equations also allow us to estimate the count that should be expected at rank i [226]. Assume there are n different items overall. Given the integral nature of the distribution, the count of the last one (and actually a large number of items toward the end) should be 1. Plugging this into Equation (3.41) leads to $1 = C\, n^{-\theta}$, so the constant is $C = n^{\theta}$. Equation (3.41) then becomes

$$c = n^{\theta}\, i^{-\theta} = \left(\frac{n}{i}\right)^{\theta}$$

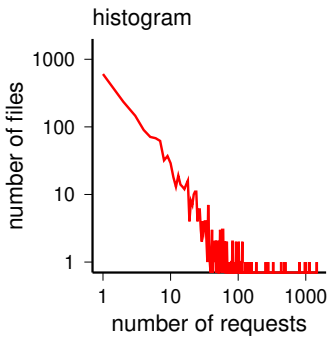
Details Box: Three Plots for Power Laws

There are three common ways to plot power-law or Zipfian data. All exhibit straight lines in log-log axes. They are illustrated here using data of HTTP requests for files from a server at SDSC.

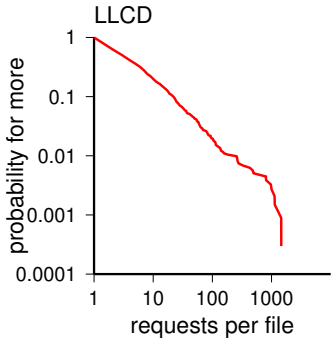
One plot is the rank-size plot introduced by Zipf. Here the X axis is the rank of an item, and the Y axis is its size (or number of requests or appearances, or probability). Hence the tail data appears on the top left. These top-ranked items all have different sizes, and according to Zipf's Law they fall on a straight line in this plot; by definition, the slope of this line is $-\theta$, where θ is the parameter of the Zipf distribution. Lower ranked items may have the same sizes (e.g., there are many items of size 2 or 3), leading to the characteristic step shape at the bottom right of the plot.



The simplest and most intuitive plot is the histogram: the X axis is the size of an item (or how many times a file is requested or a word appears), and the Y axis is a count of how many such items (or files or words) there are. There are very many items of size 1, fewer of size 2, and so on: these are actually the widths of the steps at the bottom right of the Zipf plot. With log-log axes, we get a straight line with a slope of $-(1/\theta + 1)$ (use Equation (3.43) to derive the CDF, and differentiate to get the pdf). But note that the tail of the distribution is hard to characterize, because there are many unique values that each appear only once (these are the distinct values at the top left of the Zipf plot and on the extreme right of the histogram). This can be solved by using logarithmic-sized bins when drawing the histogram [514] (Figure 5.18). However, such binning loses some data fidelity.



To better capture the tail, we can plot the survival function: the probability of seeing items bigger than a certain value (files that are requested more times, words that appear more times, etc.). Again, we use log-log axes. This is called a log-log complementary distribution plot, and is used in Chapter 5 as a tool to characterize heavy tails. The X axis is the size, and the Y axis is the probability of seeing larger sizes. This naturally drops off for large sizes, and according to Equation (3.43) we should get a straight line with a slope of $-1/\theta$. The tail data appears on the bottom right, which reflects the fact that this is actually the same as the Zipf rank-size plot flipped on its side.



End Box

An important property of this distribution is that the shape remains the same when two sets of items are merged. For example, this book can be considered as using two vocabularies: the conventional English vocabulary and the specific vocabulary of workload modeling (for example, the word "distribution" is not in general the 12th most common word in English; in fact, according to http://wordcount.org, its rank is 1637, right after "dinner"). Each vocabulary by itself follows the Zipf distribution, and so does the combined set of words used in the whole book. In workloads, we might see requests to many servers flowing through a network link. If the requests to each server follow the Zipf distribution, so will the combined stream. The only difference will be in the size of the support (the number of values for which the distribution is nonzero) and the normalization constant.

The derivation showing this is simple. Starting with Equation (3.42), consider the probability of being larger than a certain value x in each of two sets: a set of n_1 items in which x appears at rank i_1, and a set of n_2 items in which x appears at rank i_2. Taken together, we then have

$$\Pr(X > x) = \frac{i_1 + i_2}{n_1 + n_2}$$

But both i_1 and i_2 can be expressed as a function of x with appropriate constants C_1 and C_2 (which depend on n_1 and n_2, but are nevertheless constants). We can then extract the common factor $x^{-1/\theta}$ and the result is

$$\Pr(X > x) = \frac{C_1 + C_2}{n_1 + n_2} x^{-1/\theta}$$

Note, however, that this only works if the sets are Zipf distributed with the same θ.

Uses

The Zipf distribution turns out to describe the distribution of many varied human activities [442, 514]. As noted earlier, one example is the distribution of word frequencies in natural language [764]. Other examples include the sizes of cities [273] and the productivity of authors of scientific papers [668, 207].

In computer workloads, the Zipf distribution is often found to be a good model for popularity distributions. A good example is the popularity of files stored on a server. If the most popular file is accessed k times, the next most popular may be expected to be accessed about $k/2$ times, the third most popular $k/3$ times, and so on. Another example is the popularity of documents on a web server [57, 85, 572] or the websites themselves [9, 10]. A third is the popularity of online games [116]. However, there are exceptions. For example, it has been observed that the distribution of popularity on file sharing services such as KaZaA is not Zipfian, and, specifically, that the most popular items are downloaded much less than expected. This is attributed to the fact that the shared media files are immutable, and therefore only downloaded once by each user [309].

In addition, an important consideration is the size of the sample being considered (or perhaps equivalently, the time scale of the observation period). If the real popularity distribution has a heavy tail (that is, the tail of the distribution, but not necessarily the whole distribution, is Pareto), it may look like a Zipf distribution when only a small number of samples are considered [690]. The reason is that with few samples they will be dominated by the Pareto (and hence, Zipfian) tail. But when we look at very many samples we see the deviation from the Zipf distribution, because there are not enough different items in the underlying population. This is discussed in more detail later in relation to Figure 5.16.

Connection Box: The Value of a Network

Zipf's law has also been linked with the question of the value of a network. The common wisdom on this issue is summarized by Metcalfe's law, which states that the value of a network grows quadratically with the number of members; thus if the number of members is doubled, the value quadruples. This is based on the simplistic model where n members have about n^2 connections (each one connects to all the others), and all these connections have equal value.

An alternative is to postulate that connections are not all equally important. In particular, we can assign them a value according to Zipf's law. The most useful connection is given a value of 1. the next most useful has a value of $\frac{1}{2}$, the third $\frac{1}{3}$, and so on. The sum of all these values is then on the order of $\log n$, and the total value of the network is only $n \log n$, not n^2 [90].

An important outcome of this model is that most of the value is concentrated at the top. In fact, in a network with n nodes, half of the value belongs to the top $\sqrt{n}$ members [554]. In certain contexts, this means that the high-value items can be found and used efficiently. An example in point is the network of citations among scientific papers. In 1966 the Science Citation Index covered 1573 out of an estimated total of 26,000 journals, which was a mere 6% of the journals. But assuming that these were the top cited journals, they are expected to have contained 72% of the cited papers of that year [554]. Moreover, the payoff of making significant extensions to the set of covered journals would be expected to be rather small.

End Box

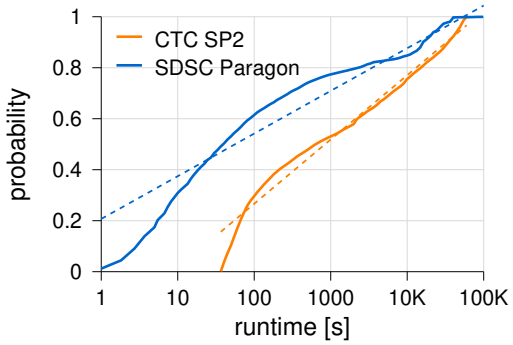


Figure 3.22: *Downey's log-uniform model (dashed lines) of job runtimes.*

Generation

If the range of i is finite and known, samples from a Zipf distribution can be easily generated by using a precomputed table. For example, this is applicable for files stored on a server, where we know that the number of files is n.

The value in the ith entry of the table is

$$Z[i] = \frac{\sum_{j=1}^{i} 1/j^{\theta}}{\sum_{j=1}^{n} 1/j^{\theta}}$$

that is, the relative size of the first i counts. Now generate a uniform random variate $u \in [0, 1]$. Find the index i such that $Z[i] \leq u \leq Z[i+1]$. This is the index of the item that should be selected. In effect, this is just like sampling from an empirical distribution.

3.2.13 Do It Yourself

The distributions described in this chapter are not a comprehensive list. Many more have been defined and used (e.g. [221]), and it is also possible to create new constructions based on the concepts described here.

For example, Downey has defined the log-uniform distribution to describe the runtimes of jobs on parallel supercomputers [186, 187]. This is inspired by the lognormal distribution: first perform a logarithmic transformation, and then see what the resulting distribution looks like. In this case, it looked like a uniform distribution on a certain well-defined range. The data and the log-uniform model are compared for two systems in Figure 3.22. At least for the CTC SP2, the model seems to be pretty good.

As another example, Lublin has defined what may be called a "hyper log-gamma" distribution [454]. The log-gamma part comes from fitting log-transformed data to a gamma distribution. The hyper part comes from actually fitting a mixture of two such log-gamma distributions, so as to capture shapes that are bimodal — such as the SDSC

Paragon data shown in Figure 3.22. A similar construction was used by Downey to fit distributions of file sizes, but using the lognormal distribution as the basis [188].

4

Fitting Distributions to Data

In this chapter we consider the problem of finding a distribution that fits given data. The data has a so-called *empirical distribution* — a list of all the observed values and how many times each one of them has occurred. The goal is to find a distribution function that is a good match to this data, meaning that if we sample it we will get a list of values similar to the observed list. Note, however, that we never expect to get a perfect match. One reason is that randomness is at work — two different sets of samples from the same distribution will nevertheless be different. More importantly, there is no reason to believe that the original data was indeed sampled from our model distribution. The running times of Unix processes are *not* samples from an exponential distribution, or a Pareto distribution, or any other distribution function that has a nice mathematical formulation.

But we can hope to find a distribution function that is a close enough match. The real meaning of "close enough" is that it will produce reliable results if used in a performance evaluation study. As this is impossible to assess in practice, we settle for statistical definitions. For example, we may require that the distribution's moments be close to those of the data, or that its shape be close to that of the empirical distribution. Thus this entire chapter is concerned with the basic methods of descriptive modeling.

To read more: Although we cover the basics of fitting distributions here, there is much more to this subject. An excellent reference, including the description of many different distributions, is the book by Law and Kelton [427], which also has the advantage of placing the discussion in the context of modeling and simulation. There are many statistics texts that discuss fitting distributions per se, e.g. DeGroot [169] and Montgomery and Runger [501]. Fitting distributions is a special case of fitting data in general; a very accessible review was written by Christopoulos and Lew [135], and a good book is by Berthold et al. [71].

4.1 Approaches to Fitting Distributions

There are two basic approaches to finding a distribution that fits a given dataset. One is to limit the search to a specific type of distribution. In this case, only the parameters of the distribution have to be found. If the type is not known, this can be generalized by trying

to match a sequence of different distributions, and then selecting the one that provided the best match. The alternative approach is to forgo the use of predefined distributions and simply construct a distribution function with the desired shape.

A good example of the first approach is provided by a Poisson arrival process. Consider the points in time in which users start new sessions with an interactive server. It is reasonable to believe that such arrivals are independent of each other, and are uniformly distributed over time. They therefore constitute a Poisson process. If this is true, then the interarrival times are exponentially distributed. We can therefore *assume* this distribution, and look for the parameter θ that best fits the data. This value turns out to be equal to the average of the interarrival times.

A good example of the second approach, in which we do not use a known distribution function, is provided by the distribution of job sizes in parallel supercomputers. This distribution is quite unique, with many small jobs, few large jobs, and a strong emphasis on powers of two. It is not similar to any commonly used mathematical distribution. Therefore we have no alternative but to hand-carve a distribution function that has the desired properties, or simply use the empirical distribution function.

The following sections discuss these approaches in detail. Section 4.2 provides the basis, and covers the estimation of distribution parameters so as to match given data samples. Section 4.3 extends this discussion to cases where a single distribution cannot capture the data and a mixture is needed. Section 4.4 deals with cases where we simply want to match the shape of the data (e.g., by using the empirical distribution directly). After that we turn to tests for goodness of fit.

4.2 Parameter Estimation for a Single Distribution

The most common approach for fitting distributions is to use the following procedure [427]:

1. Select a candidate distribution function.
2. Estimate the values of the distribution parameters based on the available data.
3. Check the goodness of fit.

We might expect that step 1 should be done based on the general shape of the distribution; for example, if the histogram of the data is bell shaped, we might try a normal distribution, whereas if it has a tail we would try an exponential one. However, there are actually very many possible distribution functions with rather similar shapes that are hard to distinguish from each other. The real considerations are therefore the availability of prior knowledge regarding the distribution, the desire to perform a comprehensive search, or practical considerations. These are all discussed in Section 4.2.1.

Step 2 may be based on calculating the moments of the data and using them as estimators for the moments of the distribution, which is described in Section 4.2.2. Regrettably, this method has the drawback of being very sensitive to outliers, especially if high moments are needed. An alternative is to look for maximum likelihood estimates, i.e.

parameter values that are the most likely to have led to the observed data, as described in Section 4.2.3.

Although step 2 finds the most reasonable parameter values for the given distribution function, this still does not necessarily mean that this specific distribution is indeed a good model of the data. We therefore need step 3, in which we assess the goodness of fit in absolute terms. This is done by comparing the distribution function with the empirical distribution of the data, as described in Section 4.5.

4.2.1 Justification

There are several possible justifications for trying to match a predefined distribution function.

The first and best justification is knowing that this is indeed the correct distribution. For example, if we have reason to believe that arrivals are a Poisson process, then the interarrival times must be exponentially distributed. If a certain workload attribute is known to be the sum of many contributions, which come from some distribution with finite variance, then this attribute can be assumed to be normally distributed. Likewise, if an attribute is the product of many contributions, it can be assumed to be lognormal. Of course such assumptions should be checked, but they nevertheless provide a good starting point.

If we do not know what the correct distribution is, it is nevertheless possible to use the procedure of fitting a given distribution as a subroutine in an automated algorithm to find the best match. In this case we have a large number of possible distributions, and we try to find the best parameters for each one. We then run goodness-of-fit tests on the resulting distributions. The distribution that achieves the best goodness-of-fit score is selected as the best model for the data. This approach is used by many distribution-fitting software packages.

Another possible justification is that the precise distribution does not matter. For example, some queueing analysis results depend only on the first few moments of the interarrival or service times. Thus the exact distribution that is used is not important, as long as it has the correct moments. For example, it can be shown that in an M/G/1 queue the average waiting time is [581, p. 383]

$$W = \frac{\lambda \mathbb{E}[S^2]}{2(1 - \lambda \mathbb{E}[S])}$$

where λ is the arrival rate and $\mathbb{E}[S]$ and $\mathbb{E}[S^2]$ are the first two moments of the service time distribution. Note that, although the arrivals are assumed to be Poisson, the service times can come from any distribution for which the first two moments are defined. This enables one to choose from a family of distributions with different characteristics. Often, a hyper-exponential distribution is used if the coefficient of variation is larger than 1, and an Erlang distribution is used if it is smaller than 1.

The weakest justification is one of convenience. For example, one may select an exponential distribution because of its memoryless property, which simplifies mathematical analysis. Another example occurs when we have extra parameters that can be

set to arbitrary values. For instance, a two-stage hyper-exponential distribution is defined by three parameters: θ_1, θ_2, and p. If we want to match only the first two moments of the distribution, we then have two equations with three unknowns. There are therefore many possible solutions, and we can pick one of them.

However, it is typically inappropriate to select a distribution if we know outright that it cannot be the correct one. For example, if interarrival times have a CV that is significantly larger than 1, it is inappropriate both to model them using an exponential distribution, and to assume that the arrivals constitute a Poisson process.

4.2.2 The Method of Moments

Obviously, given a distribution function, the moments can be calculated:

$$\mu'_r = \int_{-\infty}^{\infty} x^r f(x) \mathrm{d}x \tag{4.1}$$

Likewise, the central moments can be calculated

$$\mu_r = \int_{-\infty}^{\infty} (x-\mu)^r f(x) \mathrm{d}x \tag{4.2}$$

Calculating either of the integrals for a specific pdf typically leads to equations that portray a functional relationship between the parameters of the distribution and the moments. For example, the exponential distribution has one parameter, θ, which is found to equal its mean (i.e., the first moment). As another example, the gamma distribution has two parameters, α and β. Its first moment (the mean) and second central moment (the variance) are related to these parameters by the following equations:

$$\begin{aligned} \bar{X} &= \alpha\,\beta \\ \mathbb{V}\mathrm{ar}(X) &= \alpha\,\beta^2 \end{aligned}$$

Because the expressions specifying the moments as a function of the parameters can be inverted, it is possible to estimate the parameters based on measured moments. Thus we can compute the average and variance of a set of samples taken from a gamma distribution, and use them as estimators for the real average and variance. We then invert the above equations to derive

$$\begin{aligned} \hat{\alpha} &= \bar{X}^2 / \mathbb{V}\mathrm{ar}(X) \\ \hat{\beta} &= \mathbb{V}\mathrm{ar}(X) / \bar{X} \end{aligned} \tag{4.3}$$

If the distribution has more parameters, higher moments have to be used to provide the required number of equations.

Note, however, that these are not exact values for the parameters. The problem is that we do not really know the true moments — we can only estimate them from the data samples. This is particularly problematic when the data is highly skewed, because high moments are very sensitive to outliers. This issue is discussed further in Section 4.2.5.

Nevertheless, the approach of matching moments is mathematically pleasing, and several methodologies have been developed based on it. Osogami and Harchol-Balter show how the first three moments of a distribution can be used to define a Coxian distribution with minimal stages [527] or a combined Erlang-Coxian distribution [528]. Jann et al. show how to match the first three moments to a hyper-Erlang distribution [370]. Johnson has proposed a parameterized family of distributions that can be used to match the first four moments of the data by transforming it to the normal distribution (the special case of a simple logarithmic transformation thus induces the lognormal distribution) [383]. All these are examples of abstract modeling, which seeks to find the simplest mathematical model with certain properties, in this case the values of several moments.

4.2.3 The Maximum Likelihood Method

Given a chosen family of distributions, the question is what parameter values will yield the family member that best fits the data. The idea behind the maximum likelihood method is to derive those parameter values that would lead to the highest probability of sampling the given data values [480]. In the following we discuss a single parameter, but the extension to multiple parameters is straightforward.

The likelihood function is the probability of observing a set of samples $x_1, \ldots, x_n$ given that they come from a known distribution. If the distribution is defined by a parameter θ, we can write

$$L(\theta \mid x_1, \ldots, x_n) = \prod_{i=1}^{n} f(x_i \mid \theta) \tag{4.4}$$

In other words, the likelihood of observing a set of samples is the product of the probabilities of the individual values. This is based on the assumption that the random variables of the sample are independent. But note the reversal of roles: the pdf reflects the probability for a value x given the parameter θ, whereas the likelihood of the parameter θ depends on the given observations x.

Recall that we assume that the distribution from which the samples were drawn is essentially known. Only the parameter of the distribution is not known. We want to find the parameter value that is most likely to have given rise to these samples. To do so, we simply differentiate the likelihood function with respect to the parameter. We then equate the derivative to zero to find the value of θ that maximizes the likelihood.

When deriving maximum likelihood parameters, it is common to use a logarithmic transformation and work with the so-called log-likelihood function. Since taking the logarithm of a value is a monotonic transformation, the maximum of the log-likelihood function is also the maximum of the likelihood function. In practice, it is often easier to calculate the maximum of the log-likelihood function, because the logarithmic transformation turns a product into a sum and an exponent into a product.

The steps for estimating parameter values using the maximum likelihood method are therefore as follows:

1. Given the assumed distribution function f, compute the likelihood function according to Equation (4.4).

2. Take the logarithm of this expression.
3. Differentiate with respect to θ.
4. Equate the result with zero.
5. Extract an expression for θ.
6. Verify that this is a maximum, by verifying that the second derivative is negative.

As a concrete example, consider the case of the exponential distribution. The likelihood of sampling values $x_1, \ldots, x_n$, given that they come from an exponential distribution with parameter θ, is (by Equation (4.4))

$$L(\theta \,|\, x_1, \ldots, x_n) = \prod_{i=1}^{n} \frac{1}{\theta} e^{-x_i/\theta}$$

Taking a log and developing this equation leads to

$$\begin{aligned} \ln(L(\theta \,|\, x_1, \ldots, x_n)) &= \ln\left(\prod_{i=1}^{n} \frac{1}{\theta}\, e^{-x_i/\theta}\right) \\ &= \sum_{i=1}^{n} \left(\ln(1/\theta) - \frac{x_i}{\theta}\right) \\ &= n \ln(1/\theta) - \frac{1}{\theta}\sum_{i=1}^{n} x_i \end{aligned}$$

To find the θ that maximizes this expression we differentiate with respect to θ, and derive

$$\frac{\partial}{\partial\theta} \ln(L) = -n\,\frac{1}{\theta} + \frac{1}{\theta^2} \sum_{i=1}^{n} x_i$$

The extremum is obtained when this is equal to zero, leading to

$$n\,\theta = \sum_{i=1}^{n} x_i$$

and giving the solution that θ is equal to the mean of the samples. The second derivative can be seen to be negative, so this is indeed a maximum.

Note that this methodology for finding maximum likelihood parameters cannot always be applied. Not every imaginable distribution leads to a closed-form solution. In such cases, a numerical maximization procedure can be used in place of differentiation. But maximum likelihood estimation is directly applicable to the exponential family of distributions.

Background Box: The Exponential Family of Distributions

The exponential family of distributions is the set of distributions whose pdf can be written as an exponential of the product of two functions: a function of the argument x and a function of the parameter θ. The most general expression is

$$f(x, \theta) = e^{p(x)q(\theta)+r(x)+s(\theta)}$$

The exponential distribution is, of course, a special case, which is easily seen by selecting $p(x) = x$, $q(\theta) = 1/\theta$, $r(x) = 0$, and $s(\theta) = \ln(1/\theta)$. The hyper-exponential, Erlang, hyper-Erlang, normal, lognormal, and gamma distributions also belong to this family.

The importance of this family is that it is easy to estimate the parameter values based on samples from the distribution. In fact, each parameter can be expressed as a simple function of the moments of the samples, and this gives maximum likelihood values. The moments are said to be *sufficient statistics*, and completely define the distribution.

End Box

4.2.4 Estimation for Specific Distributions

Using the method of matching moments or the maximum likelihood method, the following results can be obtained.

The Exponential Distribution

The exponential distribution is defined by a single parameter, θ, which is also the mean of the distribution. It is therefore not surprising that when a set of data sampled from an exponential distribution is given, the maximum likelihood estimator for θ is the average of the samples. This was shown formally above.

The Hyper-Exponential Distribution

The number of parameters of a hyper-exponential distribution depends on the number of stages it has. The two-stage hyper-exponential distribution has three parameters, so in principle three moments are needed. If only the first two moments are calculated, we therefore have two equations with three unknowns [428, 568]:

$$\mu = \frac{p}{\lambda_1} + \frac{1-p}{\lambda_2}$$

$$\mu_2' = \frac{2p}{\lambda_1^2} + \frac{2(1-p)}{\lambda_2^2}$$

This leaves us the freedom to choose one of an entire family of distributions that all match these two moments.

One specific procedure for creating a hyper-exponential distribution that matches given values for the first two moments is as follows: [21].

1. Calculate the CV squared: $CV^2 = \dfrac{\mu_2' - \mu^2}{\mu^2}$.

2. Calculate $p = \frac{1}{2}\left(1 - \sqrt{\frac{CV^2 - 1}{CV^2 + 1}}\right)$. Note that this requires $CV \geq 1$.
3. Set $\lambda_1 = \frac{2p}{\mu}$ and $\lambda_2 = \frac{2(1-p)}{\mu}$.

This formulation distributes the mass equally among the two stages, as $\frac{p}{\lambda_1} = p\frac{\mu}{2p} = \frac{\mu}{2}$, and likewise for λ_2.

Hyper-exponential distributions can also be used to match the shape of a monotonic distribution (e.g., the tail of a heavy-tailed distribution). This application typically requires more than two stages, and is described in Section 4.4.3.

The Erlang Distribution

The Erlang distribution is defined by two parameters: the rate λ of each exponential stage and the number of stages k. A simple heuristic to find these parameters is as follows. Because k uniquely defines the distribution's coefficient of variation, it is easiest to start by estimating k. The CV of a k-stage Erlang distribution is $1/\sqrt{k}$. By calculating the CV of the data, we can find the nearest value of k (Allen suggests using $k = \lfloor 1/CV^2 \rfloor$ [21]). Given k, we can estimate $1/\lambda$ as $1/k$ of the mean of the samples.

A more formal approach is to match the first two moments of the distribution. These are calculated as

$$\begin{aligned} \mu &= \frac{k}{\lambda} \\ \mu_2' &= \frac{k(k+1)}{\lambda^2} \end{aligned}$$

providing two equations with two unknowns.

The Hyper-Erlang Distribution

The hyper-Erlang distribution is a generalization of the exponential, hyper-exponential, and Erlang distributions. Jann et al. have developed a procedure to match the simplest of these four distributions to given data [370]. This is done in two steps. First, check whether the data satisfies distribution-specific constraints on the first three moments. For example, the exponential distribution has only one parameter, so the moments must be related to each other; in particular, the mean and the standard deviation are equal. For the other distributions, the constraints are a set of inequalities. Second, find the parameters of the selected distribution by matching the first three moments.

The Gamma Distribution

The gamma distribution has two parameters: the shape parameter α and the scale parameter β. Finding maximum likelihood values for these parameters requires the solution of a pair of complicated equations — see Law and Kelton [427, p. 302]. A much simpler alternative is to match the moments of the data, yielding the equations shown above on page 142 (Equation (4.3)).

The Weibull Distribution

The Weibull distribution has the same two parameters as the gamma distribution: the shape parameter α and the scale parameter β. Finding maximum likelihood values for these parameters again involves the solution of a pair of complicated equations — see Law and Kelton [427, pp. 301 and 305].

The Lognormal Distribution

The lognormal distribution has two parameters: μ and σ, which are the mean and standard deviation of the logarithm of the values, respectively. Their estimation is therefore very simple:

$$\hat{\mu} = \frac{1}{n}\sum_{i=1}^{n} \ln x_i$$

$$\hat{\sigma} = \sqrt{\frac{1}{n-1}\sum_{i=1}^{n} (\ln x_i - \hat{\mu})^2}$$

This is another one of the cases where matching moments in fact leads to maximum likelihood estimates.

The Pareto Distribution

The Pareto distribution has two parameters: k and a. k is a location parameter, and signifies where the distribution starts. It is estimated by the smallest sample seen: $\hat{k} = \min_i\{x_i\}$. Given $\hat{k}$, the maximum likelihood estimate for a is

$$\hat{a} = \frac{1}{\frac{1}{n}\sum_{i=1}^{n} \ln \frac{x_i}{\hat{k}}}$$

This expression is justified, and other estimation methods are considered, in Section 5.4.1.

4.2.5 Sensitivity to Outliers

Calculation of moments sometimes plays an important part in fitting distributions. To begin with, one may select suitable distributions based on moments, especially the mean and the variance of the sample data. For example, these statistics typically indicate that the distribution of job runtimes has a wide dispersion (the CV is greater than 1), leading to a preference for a hyper-exponential model over an exponential one. More troubling is the use of estimated moments to calculate parameter values with the method of moments. Note that the number of equations needed is the number of unknown parameters of the distribution. Thus if the number of parameters is large, high-order moments are required. For example, the exponential distribution has only one parameter, which can

Dataset	Rec's omitted (% of total)		Statistic (% change) Mean	2nd moment	CV	Median
KTH SP2	0		7.44	212.84	1.69	3
job size	2	(0.010%)	7.43 (-0.19%)	210.56 (-1.07%)	1.68 (-0.47%)	3 (0.00%)
	4	(0.019%)	7.42 (-0.31%)	209.62 (-1.51%)	1.68 (-0.61%)	3 (0.00%)
	8	(0.039%)	7.40 (-0.55%)	207.79 (-2.37%)	1.67 (-0.88%)	3 (0.00%)
	16	(0.078%)	7.37 (-1.02%)	204.22 (-4.05%)	1.66 (-1.41%)	3 (0.00%)
	32	(0.156%)	7.30 (-1.90%)	197.77 (-7.08%)	1.65 (-2.35%)	3 (0.00%)
	64	(0.312%)	7.18 (-3.53%)	186.86 (-12.21%)	1.62 (-3.91%)	3 (0.00%)
KTH SP2	0		6145	247×10^6	2.36	583
runtime	2	(0.010%)	6125 (-0.33%)	243×10^6 (-1.78%)	2.34 (-0.67%)	582 (-0.17%)
	4	(0.019%)	6105 (-0.65%)	239×10^6 (-3.51%)	2.33 (-1.33%)	582 (-0.17%)
	8	(0.039%)	6065 (-1.29%)	230×10^6 (-6.91%)	2.29 (-2.66%)	582 (-0.17%)
	16	(0.078%)	5987 (-2.57%)	214×10^6 (-13.6%)	2.23 (-5.47%)	582 (-0.17%)
	32	(0.156%)	5840 (-4.97%)	184×10^6 (-25.6%)	2.10 (-10.97%)	581 (-0.34%)
	64	(0.312%)	5623 (-8.48%)	151×10^6 (-38.8%)	1.95 (-17.39%)	577 (-1.03%)
proc94	0		0.37	25.05	13.51	0.02
runtime	4	(0.002%)	0.35 (-4.46%)	7.34 (-70.7%)	7.61 (-43.7%)	0.02 (0.00%)
	8	(0.004%)	0.35 (-6.06%)	5.68 (-77.3%)	6.79 (-49.7%)	0.02 (0.00%)
	16	(0.009%)	0.34 (-8.08%)	4.36 (-82.6%)	6.06 (-55.1%)	0.02 (0.00%)
	32	(0.017%)	0.33 (-10.6%)	3.32 (-86.7%)	5.43 (-59.8%)	0.02 (0.00%)
	64	(0.035%)	0.32 (-14.2%)	2.30 (-90.8%)	4.67 (-65.4%)	0.02 (0.00%)
	128	(0.069%)	0.30 (-18.2%)	1.63 (-93.5%)	4.10 (-69.6%)	0.02 (0.00%)
	256	(0.139%)	0.28 (-23.2%)	1.12 (-95.5%)	3.59 (-73.4%)	0.02 (0.00%)
	512	(0.277%)	0.26 (-29.3%)	0.73 (-97.1%)	3.12 (-76.9%)	0.02 (0.00%)

Table 4.1: *Sensitivity of statistics to the largest data points. The KTH dataset includes only jobs that terminated successfully.*

be estimated based on only the first moment (the mean). But the two-stage hyper-Erlang distribution has five, so the fifth moment is also required.

The problem with using statistics based on high moments of the data is that they are very sensitive to rare large samples [190]. In the skewed distributions that are characteristic of workloads, there are always some samples that are much larger than all the others. Such outliers tend to dominate the calculation of high moments, leading to a situation in which most of the data is effectively ignored. Consider a sample with one outlier, e.g. 1, 1, 1, 1, 2, 2, 2, 3, 3, 4, 5, 15. The fifth moment, defined as $\mu'_5 = \frac{1}{n}\sum_{i=1}^{n} x_i^5$, is 63,675.8. The sum itself, without the $\frac{1}{n}$ factor, is 764,110; 99.4% of this is due to the outlier, $15^5 = 759,375$. Any parameters calculated based on the fifth moment are therefore dominated by this single rare sample, which is not necessarily very representative. For example, if the outlier was 16 instead of 15, the sum would jump to 1,053,311 and the fifth moment to 87,775.9 — an increase of 37.8%! And we could add dozens of samples with values of 1, 2, and 3, that would essentially have no effect.

Examples based on real data are given in Table 4.1, which shows the first two moments, the CV, and the median for three datasets. To show the effect of the largest samples, those samples are omitted and the statistics recalculated. This procedure is repeated several times, as more and more samples are omitted.

In the KTH-SP2 job-size dataset, removing just 2 jobs from a total of 20,542 causes the second moment to drop by a full percent; removing the 16 biggest jobs causes the average to drop by more than 1%, the second moment to drop by more than 4%, and the CV to drop by 1.4%. The runtime data shows a much larger drop in the second moment, and, as a result, also in the CV: when the 16 longest jobs are removed, the second moment and the CV drop by 13.6% and 5.5%, respectively. The results for the heavy-tailed Unix process dataset from 1994 are even more extreme. For this dataset, removing just the top 4 entries (out of 184,612) causes the average to drop by 4.5%, the second moment by 70%, and the CV by 44%. In all these cases, the median, as a representative of order statistics, hardly changes at all.

It should be understood that the sensitivity of higher moments to outliers is a real problem, and not just a byproduct of artificially removing selected samples. The samples omitted are from the extreme tail of the distribution. By definition, these samples are rare. It is very likely that if we were to collect data for a different period of time, we would have seen different samples from the tail. As a result, the computed moments would have been different too.

In general, the maximum likelihood method for parameter estimation is less sensitive to outliers than the moment matching method. When calculating the maximum likelihood, all samples have the same weight. Maximum likelihood estimators are also usually less biased. It is therefore preferable to use the maximum likelihood method when possible.

4.2.6 Variations in Shape

Another problem with fitting distributions by using moments is that the fitted distribution can have a rather different shape than the original one.

For example, Jann et al. have used hyper-Erlang distributions to create models that match the first three moments of the modeled data [370] (in practice, differences of 30% between moments calculated from the original data and from samples from the model are common, but the order of magnitude is always correct). But these distributions turn out to be distinctly bimodal, and do not reflect the shape of the original distributions which are much more continuous (Figure 4.1).

The use of distributions with the right shape is not just an aesthetic issue. Back in 1977 Lazowska showed that using models based on a hyper-exponential distribution with matching moments to evaluate a simple queueing system failed to predict the performance observed in practice [428]. Moreover, different hyper-exponentials that matched the same two first moments led to different results. Similar observations were made by Riska et al., who specifically showed that the queue length distribution was different for different hyper-exponential models, and that none matched the queue length distribution produced by the original data [568]. It may therefore be better to use distributions with

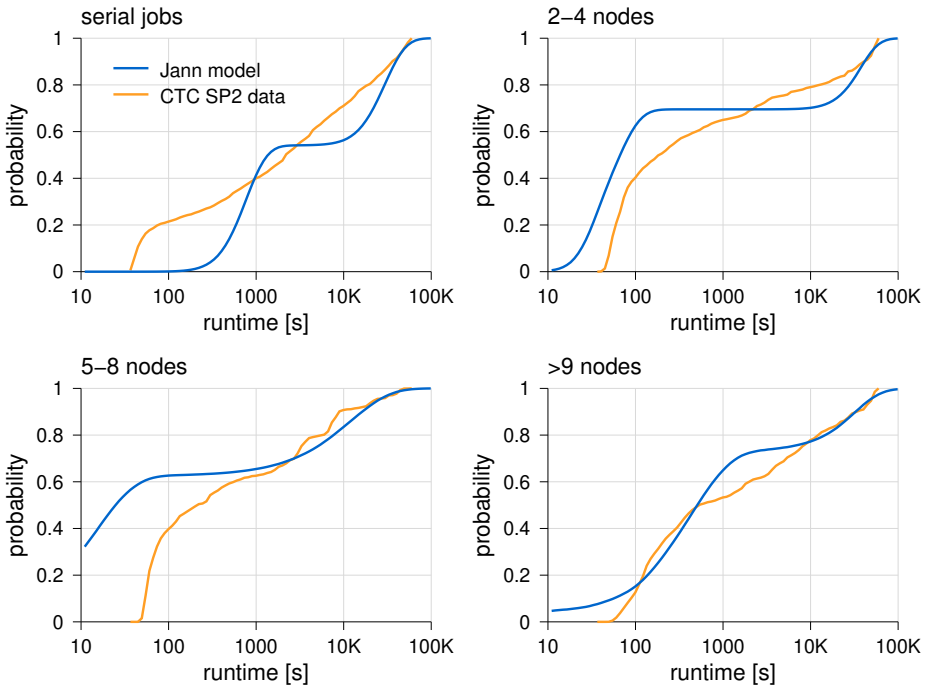


Figure 4.1: *Comparison of the CTC SP2 job runtime data for different ranges of job sizes, and the Jann models of this data. The models use a hyper-Erlang distribution that matches the first three moments, but fails to capture the shape.*

matching percentiles instead. We consider creating distributions that match a desired shape in Section 4.4

4.3 Parameter Estimation for a Mixture of Distributions

The previous section explained how a distribution is fitted to data. But in some cases, this might be the wrong thing to do. If the data comes from a mixture of distributions, we need to identify this mixture: doing so will both yield a better model and perhaps provide a physical explanation of where the samples come from.

In a mixture, each sample comes from one (and only one) of the distributions forming the mixture. But we don't know which one, making it harder to assess the parameters of the individual distributions. The problem of missing data regarding the association of samples to distributions is side-stepped by using the iterative EM (expectation maximization) algorithm.

4.3.1 Examples of Mixtures

In the general case, the pdf of a mixture is expressed as

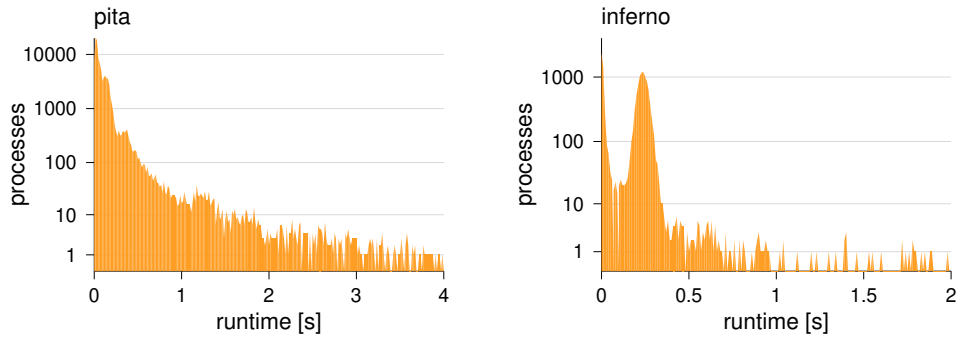


Figure 4.2: *Unix process runtimes from the pita and inferno datasets. On inferno, there seems to be an additional class of processes that run for about 0.2–0.3 seconds.*

$$f(x) = \sum_{i=1}^{k} p_i \, f_i(x)$$

This is a mixture of k distributions. The ith distribution of the mixture occurs with probability p_i and has a pdf $f_i(x)$. This might seem familiar, and indeed we have already encountered special cases of mixtures before: the hyper-exponential is a mixture of exponentials, and the hyper-Erlang is a mixture of Erlangs. But in general the constituent distributions need not all be of the same type.

There are two main reasons to use mixtures. One is to hand-tailor the shape of a distribution. For example, using a hyper-exponential distribution allows one to extend the distribution's tail, to approximate a heavy tail. This is demonstrated in Section 4.4.3. Another common construction is to use a mixture of Gaussians (that is, a mixture of normal distributions) to approximate a distribution with many modes, as shown in Section 4.4.2.

The second is using a mixture when modeling a multiclass workload, where the different classes have different distributions. As an example, consider the "inferno" data shown in Figure 4.2. This is the distribution of runtimes on a departmental Unix server during several hours one afternoon. It is obviously quite unusual, due to the large number of processes that ran for about 0.2–0.3 seconds; the typical distribution is monotonous and heavy-tailed (as in the "pita" dataset). We may therefore conjecture that the distribution actually consists of a mixture, composed of two components: the typical mix, plus a large group that ran for about 0.2–0.3 seconds.

To read more: Mixtures are typically not included in basic statistics textbooks or descriptions of distributions. An exception is the compendium by McLaughlin [481], which lists many mixtures and their properties. A book-length treatment is given by McLachlan and Peel [478].

4.3.2 The Expectation-Maximization Algorithm

The EM algorithm is based on two assumptions: that the number of distributions in the mixture and the functional form of each distribution are given, and that for each distribution estimating the parameters is not hard; it usually produces near-optimal results very quickly.

To read more: The EM algorithm was originally proposed by Dempster et al. [170]. It has since been the subject of a voluminous literature, including a review by Redner and Walker [561] and a book by McLachlan and Krishnan [479].

The algorithm is iterative. Each iteration involves two steps, called the E-step and the M-step. One version of the algorithm proceeds as follows:

1. Somehow initialize the parameters of the distributions.
2. E-Step: For each observation and for each distribution, decide what part of this observation "belongs to" this distribution. Given that the parameters of the distributions are set, we can find for each distribution the probability of getting the observation from that specific distribution. This probability is the "relative part" of the observed value that is assigned to this distribution.
3. M-Step: For each distribution estimate its parameters, using the maximum likelihood estimation method. This estimation is done based on the observations (or rather, their "parts") that are believed to "belong to" this distribution.
4. Repeat E and M steps until the likelihood converges.

Alternatively, it might be easier to initially partition the samples into rough groups that correspond to the different distributions. In this case, we start with the M-step:

1. Somehow partition the data into groups that are roughly characterized by the different distributions.
2. M-Step: For each distribution, estimate its parameters using the maximum likelihood estimation method. This estimation is done based on the observations associated with this distribution.
3. E-Step: For each observation and for each distribution, decide what part of this observation belongs to this distribution.
4. Repeat the two steps until the likelihood converges.

The M-step (maximization step) essentially fits a distribution to observed data. Because we know the functional form of the distribution and only have to find its parameters, this can be done using any of the parameter estimation methods of Section 4.2. EM is usually described using maximum likelihood estimation, which is also used in the proofs regarding EM's behavior. But in practice, matching moments also works (at least in cases where the moments are not unduly influenced by outliers and lead to representative parameter values).

The E-step is done as follows. For simplicity, we assume only two distributions in the mixture, with pdfs $f_1(\cdot)$ and $f_2(\cdot)$.

1. For an observed value x, find the probability densities that such a value is generated by either distribution. These are $p_1 = f_1(x)$ and $p_2 = f_2(x)$.
2. Find the total probability of such a value $p = p_1 + p_2$.
3. Assign a fraction $\frac{p_1}{p}$ of the sample to distribution 1, and a fraction $\frac{p_2}{p}$ to distribution 2.
4. Repeat this procedure for all other observations.

Background Box: Convergence of the EM Algorithm

To show the convergence we need to express the EM algorithm somewhat more formally. We do this for a simplified version, in which each sample is assigned to a single distribution in the mixture.

We start with a set of data samples $x = \{x_1, x_2, \ldots, x_n\}$. Assume that this comes from a mixture of distributions that are described by a set of parameters $\theta = \{\theta_1, \theta_2, \ldots, \theta_m\}$. Some of these parameters belong to the first distribution, some to the second, and so on.

The idea of the EM algorithm is to postulate a set of *latent* random variables $y = \{y_1, y_2, \ldots, y_n\}$. Each y_i assigns the corresponding sample x_i to one of the distributions: $y_i = j$ means that we currently think that the ith sample comes from the jth distribution. Assuming a mixture of only two distributions, $y_i \in \{1, 2\}$. The problem is that we do not know y. So in effect we need to infer y and estimate θ at the same time.

When executing the algorithm, the xs are actually constants. What we do is to iteratively revise the values of the θs and ys so as to maximize the probability of observing the xs. As in the maximum likelihood method, we do this in log-space. So the expression we want to maximize is

$$\log L(\theta) = \log \Pr(x|\theta)$$

The latent variables y are introduced by conditioning and summing over all the values that y may assume:

$$\log L(\theta) = \log \left(\sum_y \Pr(x|y,\theta) \Pr(y|\theta) \right)$$

In each iteration of the algorithm, we have a current estimate of θ that we shall denote θ^c. We use this to infer what y may be (i.e., the assignment of the samples in x to the different distributions). In short, we need to find the best y given x and θ^c. We introduce this consideration by multiplying and dividing the log-likelihood by a factor $\Pr(y|x,\theta^c)$. This gives

$$\log L(\theta) = \log \left(\frac{\sum_y \Pr(x|y,\theta) \Pr(y|\theta) \Pr(y|x,\theta^c)}{\Pr(y|x,\theta^c)} \right)$$

We can now get rid of the "logarithm of a sum" structure using Jensen's inequality, which states that for $\sum_j \alpha_j = 1$ we have $\log \sum_j \alpha_j x_j \geq \sum_j \alpha_j \log x_j$. Applying this using $\alpha = \Pr(y|x,\theta^c)$ we get

$$\log L(\theta) \geq \sum_y \Pr(y|x,\theta^c) \log \left(\frac{\Pr(x|y,\theta) \Pr(y|\theta)}{\Pr(y|x,\theta^c)} \right)$$

By finding a new set of parameters θ^n that maximize the right-hand side, we will achieve a log-likelihood that is at least as large. But note that θ is the *argument* of the expression, not

its value. So what we are looking for is "the argument that leads to the maximum value". This is expressed as

$$\theta^n = \operatorname*{argmax}_\theta \sum_y \Pr(y|x,\theta^c) \log\left(\frac{\Pr(x|y,\theta)\Pr(y|\theta)}{\Pr(y|x,\theta^c)}\right)$$

Let us simplify this expression. The numerator in the log is actually $\Pr(x,y|\theta)$. The denominator does not depend on θ, only on θ^c, so it does not change the maximization and can be dropped. Finally, regarding $\Pr(y|x,\theta^c)$ as a probability function, we find that $\sum_y \Pr(y|x,\theta^c)$ is actually a calculation of an expectation. Therefore we actually have

$$\theta^n = \operatorname*{argmax}_\theta \mathbb{E}[\log \Pr(x,y|\theta)]$$

which is the essence of the EM algorithm: to maximize the expectation. (The current values of the parameters, θ^c, are not lost — they are in the calculation of the expectation.)

But what about the convergence? In each iteration we maximize the expectation. So the expectation for θ^n is at least as large as for θ^c. Because the log-likelihood is always larger than this expectation, it too will be at least as large as it was before. In other words, it does not decrease.

This goes on until we reach a maximum of the expectation. This is typically also a local maximum of the log-likelihood. While not guaranteed to be a global maximum, practice indicates that the results are typically very good. Although the algorithm may actually converge to a local minimum or a saddle point, this rarely if ever occurs in practice. To verify the result, one can repeat the entire procedure with a different initialization (either different parameters or a different assignment of observations to distributions) and compare the results.

End Box

Usage Example

As an example, let us use the EM algorithm to partition the "inferno" dataset shown above in Figure 4.2 into two groups. For the typical background workload we will assume a Pareto distribution, which has been shown to provide a good model of the tail of Unix process runtimes. We burden it with modeling the full distribution. For the added component we assume a normal distribution, based on its shape in the figure. To initialize, we consider all the processes running from 0.2–0.5 seconds to be the special group, and all those outside this range to be the conventional workload. This range is somewhat off-center on purpose, to show how the EM algorithm corrects for this offset.

The progress of the algorithm is shown in Figure 4.3. It converges after two iterations and gives what is intuitively a good partitioning of the data. In fact, it is also good in comparison with the real classification. The original data includes the Unix command name and user name of each process. This information can be used to identify the added processes as repeated invocations of the Unix ps command by a small set of users. The large number of these processes is a result of runaway processes that repeatedly execute this command — a bug in the implementation of an exercise in the operating systems course...

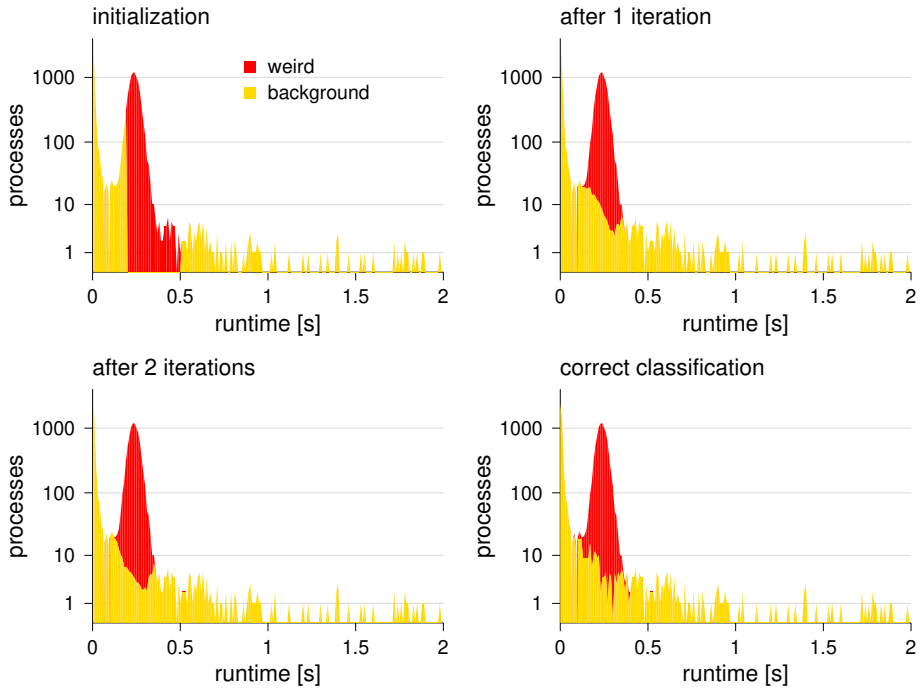


Figure 4.3: *Progress of the EM algorithm in classifying the processes into a background class with a Pareto distribution and an abnormal class with a normal distribution.*

4.4 Re-Creating the Shape of a Distribution

The previous sections showed two problems with parameter estimation: that it is sensitive to outliers that are not necessarily representative, and that it may lead to distributions that are quite different from the original data. An alternative is therefore to fashion a distribution with exactly the desired shape, or to use the empirical distribution function directly. This is therefore a less abstract form of descriptive modeling.

4.4.1 Using an Empirical Distribution

The easiest way to re-create the shape of a distribution, which is always available, is not to create a mathematical model but to use the raw data as is. This is called the *empirical distribution*. It is especially useful when the data does not resemble any commonly used mathematical distribution. An example is when the distribution has distinct modes, as happens for parallel jobs sizes or packet sizes in computer networks.

Given a dataset, representing its empirical distribution is trivial — simply generate a histogram of the observed values. Generating random variates according to this distribution is also simple. Start with a random variable selected uniformly from the interval $[0, 1]$. Then scan the histogram, until this fraction has been covered, and use the value you have reached.

More precisely, the procedure is as follows.

1. Start with a histogram h. This is a table indexed by the sampled values x_i. $h(x_i)$ is the number of times that the value x_i was seen in the samples.
2. Convert this into a CDF-like form. To do so, create a table c also indexed by the same set of x_is, whose values are normalized cumulative sums of the values of h:

$$c(x_i) = \frac{\sum_{x_j \le x_i} h(x_j)}{\sum_{x_j} h(x_j)} = \frac{|\{x_j|\ x_j \le x_i\}|}{n}$$

   where n is the number of samples. Note that $c(x)$ is monotonically increasing with x, with values in the interval $[0, 1]$. In particular, for the highest sampled value x_m we have $c(x_m) = 1$. In fact, this table embodies $F_n(x)$ of Equation (3.4).
3. Select a value u uniformly from the interval $[0, 1]$.
4. Find the smallest value x_i such that $c(x_i) \ge u$. Such a value exists because of the properties of c outlined above. Use this value.

Repeating steps 3 and 4 many times will generate a set of values that are distributed according to the original histogram h.

This procedure is suitable as is if the xs come from a set of discrete values, such as the number of processors used by a parallel job. But what about xs that are actually continuous, such as the runtimes of jobs? In such cases we would probably prefer to interpolate between the values x_i that appeared in the original sample. The changes from the previous procedure occur in steps 2–4:

2. Calculate the values of c thus:

$$c(x_i) = \frac{\sum_{x_j < x_i} h(x_j)}{\sum_{x_j} h(x_j) - 1} = \frac{|\{x_j|\ x_j < x_i\}|}{n - 1}$$

   This is again monotonically increasing to 1, but starts from 0 at the minimal value.
3. Select a value u uniformly from the interval $[0, 1]$.
4. Find two consecutive values x_i and x_{i+1} such that $c(x_i) < u \le c(x_{i+1})$. Calculate x as

$$x = x_i + \frac{u - c(x_i)}{c(x_{i+1}) - c(x_i)}(x_{i+1} - x_i)$$

   Use this value.

An open question is whether we should also extrapolate. In other words, should the minimal and maximal samples we started with be the minimal and maximal values possible, or should we allow some leeway? Intuition argues for allowing values that extend somewhat beyond the original samples, but does not say how much beyond.

Handling Censored Data

The above procedure works when we indeed have multiple samples to work with. But sometimes, we only have partial data. In particular, some of the data may be censored. This means that some of the samples have been cut short, and thus do not reflect real values. Examples of censored data include the following.

- One source of data for the distribution of interactive user sessions is a list of login sessions maintained by the system; on Unix systems, such a list is available using the last command. But issuing this command also returns data about ongoing sessions or sessions that were interrupted by a system crash. For such sessions we do not know their real duration; we only know that it is longer than the duration observed so far.
- Data about process lifetimes from an accounting system typically includes processes that were killed for various reasons. For example, some processes could be killed because they exceeded their maximal CPU time. Again, this implies that the "real" process lifetime should have been longer.

Censoring is especially common in medical statistics (e.g., in the study of survival after being diagnosed with some disease). The censoring arises due to patients who die from other causes, or patients who survive until the end of the study period. The terminology used to discuss censoring borrows from this context.

Censored items contain some information, but cannot be used directly in a histogram. The way to incorporate this information in the empirical distribution function is to note its effect on the survival probability. Recall that the survival function expresses the probability to exceed a value x:

$$\bar{F}(x) = \Pr(X > x)$$

Given that you have survived until time x, what is the probability that you will die soon thereafter? This is called the *hazard*, and may be expressed as the quotient of the pdf (giving the probability of dying at time x) divided by the survival function (expressing the condition of not dying before):

$$h(x) = \Pr(x < X < x + \delta \mid X > x) = \frac{f(x)}{\bar{F}(x)}$$

Applying these concepts to a set of samples that have been censored, denote the sampled values by X_i. Let d_i represent the number of *real* samples with magnitude X_i (that is, excluding any censored data items). Let n_i represent the number of samples with magnitude larger than or equal to X_i (both real *and* censored). The hazard, or risk of surviving for time X_i and then dying, is then d_i/n_i. Hence the probability of reaching time X_i and surviving it is $1 - \frac{d_i}{n_i}$. The general probability of surviving time X_i has to account for the probability of reaching this time at all. Thus the probability of surviving an arbitrary value x is the product of the probabilities of surviving all smaller values:

$$\bar{F}_n(x) = \prod_{X_i<x} \left(1 - \frac{d_i}{n_i}\right)$$

This is called the Kaplan-Meier formula [391]. The empirical distribution function is then $F_n(x) = 1 - \bar{F}_n(x)$.

Note, however, that this formula depends on the assumption that censoring is random. This has two implications. The first is that the formula is inapplicable if the data is actually multiclass, and one class has a greater propensity for censoring than the other. The other is that the formula assumes that censoring occurs independently and at random at any given time instant, so that large samples have a higher probability to be censored. The distribution of real samples is therefore biased toward lower values, and the Kaplan-Meier formula attempts to correct this bias by using the censored values, which tend to be larger. However, it cannot reconstruct the tail beyond the largest sampled value. This can be a problem with distributions that have a long tail.

The Problem of Overfitting

Using an empirical distribution does not abstract away and generalize the data, but rather uses all the data as is. Thus, by definition, it risks the danger of overfitting. This means that evaluations using this data will be correct only for the specific conditions that existed when the data was collected, but not for any other situations.

A possible way to avoid overfitting is to use *cross-validation*. Cross-validation is a general technique to assess the degree to which a dataset is representative. The idea is to partition the available data into two subsets, and to see how well an analysis based on one subset can predict the properties of the other subset. This is often repeated with several different partitions of the data.

A simple use of this idea in the context of using empirical distributions is to partition the data into two, and then check the similarity between the two empirical distributions. Because one of the main problems with workload data is that the workload changes with time, in this case it is better to partition the data into the first half and the second half rather than using a random partitioning. If the two parts are dissimilar one should suspect that the nature of the workload changes with time. In such a situation the sensitivity of performance evaluations to such evolution should be checked.

Using an empirical distribution is especially problematic if the data includes outliers. This emphasizes the need to first clean the data as described in Section 2.3.

4.4.2 Modal Distributions

A special case of using empirical distribution functions is the modeling of modal distributions. These are quite common in computer workloads.

A good example for a modal distribution is the distribution of job sizes on parallel supercomputers (Figure 4.4). This distribution has prominent modes at powers of two, although the reasons for this shape are not always clear. Some parallel machines, such as hypercubes or the Thinking Machines CM-5, only support jobs that require a power-of-two number of processors. But most architectures do not impose such restrictions,

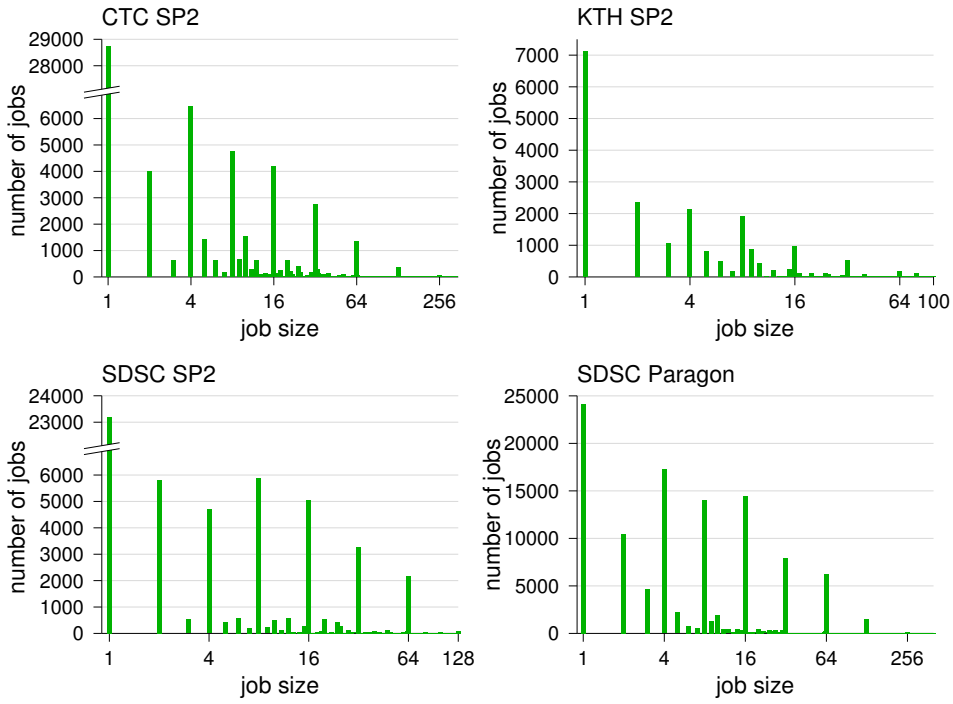


Figure 4.4: *Histograms showing the distribution of job sizes on parallel machines.*

and furthermore, users do not necessarily favor them either [136]. It may be that the root of this behavior is the way commonly used to configure batch queueing systems, in which different queues are assigned for different classes of jobs, often delimited by binary orders of magnitude.

Another well-known example of a modal distribution is the distribution of packet sizes on computer communication networks. In this case the sizes that will occur commonly are determined by various communication protocols. Each protocol has its characteristic maximum transmission unit (MTU). For example, the MTU of an Ethernet is 1500 bytes. To this you need to add various headers; in Ethernet these include 6 bytes of destination address, 6 bytes of source address, and 2 bytes to specify the length. Thus many packets that originated on an Ethernet will have a size of 1514 bytes (possibly plus some additional header bytes) also when they are transmitted across other networks.

A third example of a modal distribution is the distribution of memory object sizes (meaning general structures that are stored in memory, not necessarily in the context of object-oriented programming) [384]. Programs can in principle request memory allocations of arbitrary sizes, but in practice the requests tend to be repetitive, requesting the same size over and again. Thus the vast majority of memory objects conform to a relatively small repertoire of different sizes. The sizes may be even more limited if a single phase of the computation is considered, rather than the full execution of the program. And not only sizes tend to be repetitive — values stored in memory also tend to repeat

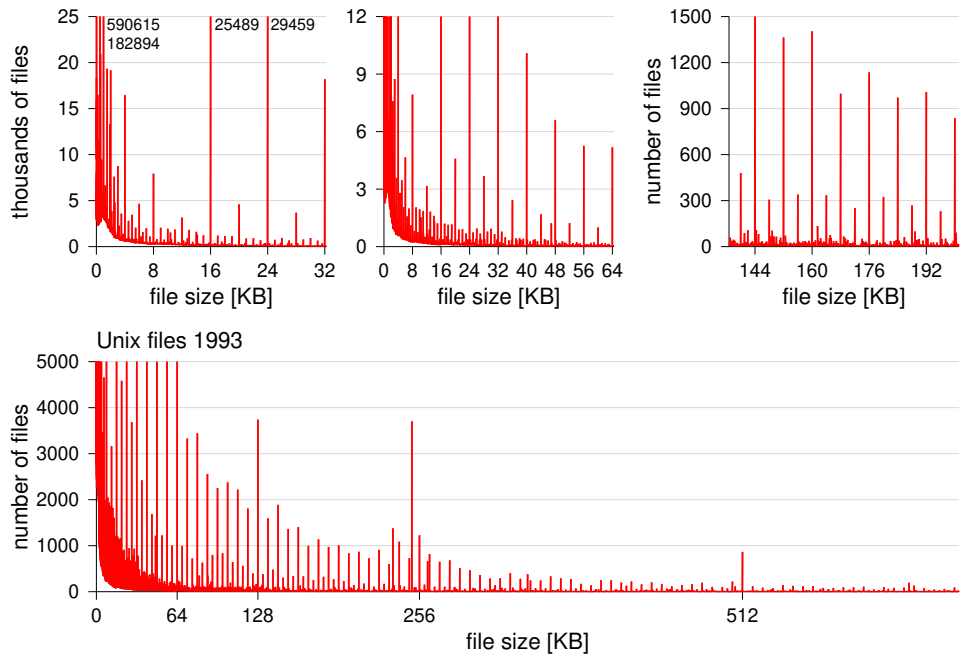


Figure 4.5: *The file size distribution in the Unix 1993 survey has a comb structure combined with a strong emphasis on small sizes.*

themselves [139, 759]. Characterization of stored data in Facebook servers, which is organized as a key-value store, revealed that both keys and values tend to have modal distributions [42].

File sizes also tend to be modal. For example, many file sizes on Unix are found to be multiples of a single block or of 8 or 16 blocks (Figure 4.5). This leads to a comb-like structure, rather than just a few discrete sizes as in previous examples. Again, the reason for this structure is not clear, but it may be due to applications that allocate large blocks of data in the interest of efficiency, because partial blocks would be lost to fragmentation anyway. And incidentally, file-access size distributions also tend to be modal, because they are generated by applications that access the same type of data structure over and over again in a loop [634].

Modeling of modal distributions is essentially the same as using an empirical distribution. However, there is an option to use some shortcuts if the modes can be characterized easily, or if it is considered sufficient to focus exclusively on the modes and ignore other values.

Modeling with a Mixture of Gaussians

In some cases, modal distributions do not have strictly discrete modes. Rather, the modes can exhibit some variance. For example, file sizes observed in P2P file-sharing systems

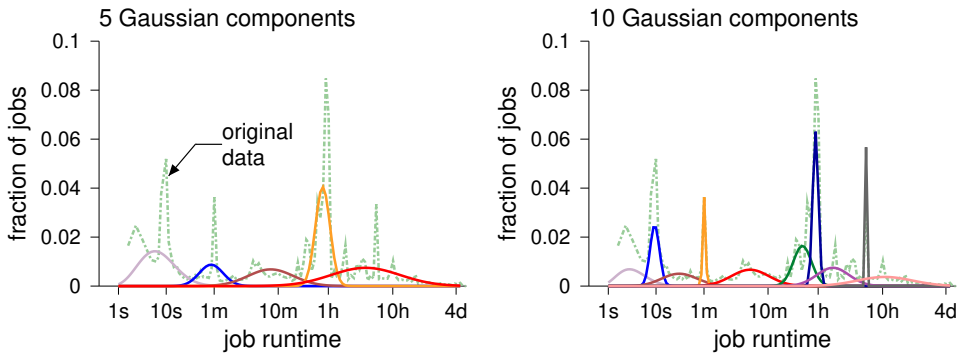


Figure 4.6: *Using a mixture of Gaussians to model a modal distribution. The quality of the results depends on the number of components used. Data is job runtimes on the HPC2N cluster.*

tend to have two major modes, corresponding to relatively short music clips and relatively long full-length feature movies [309]. Files in a corporate media server had several modes that led to a relatively wide distribution [675].

A common way to model such distributions is by using a mixture of Gaussians, or, in other words, a mixture of normal distributions — one normal distribution for each mode. The parameters of the normal components can be found using the EM algorithm, as described above in Section 4.3.2.

An example of this procedure is shown in Figure 4.6. The distribution in question is the distribution of job runtimes from the HPC2N Linux cluster. This dataset contains information about more than a half-million jobs submitted over a period of $3\frac{1}{2}$ years (we use the full log here, not the cleaned version). As seen in the figure, there are a number of prominent modes where very many jobs had similar runtimes. The fitting is done in log-space, so it actually uses lognormal components rather than normal ones. The degree to which these components of the mixture manage to model the original distribution depends on the number of components used — the more components, the better they can approximate individual modes. However, when the distribution has many modes, this leads to a very complex model. Thus if the precise structure of the modes is deemed important, it might be better to simply use the empirical distribution as described in Section 4.4.1.

4.4.3 Constructing a Hyper-Exponential Tail

Although an empirical distribution can easily be used in simulations, it cannot be used in a mathematical analysis. Moreover, in many cases such analysis can only be done based on Markovian models (i.e., using exponential distributions). This is the reason for the interest in phase-type distributions.

Of particular interest is the use of the hyper-exponential distribution to model the tail of a dataset. Although the tail by definition contains only a small number of samples,

these samples may be very big, and may therefore dominate the behavior of the system. This is discussed at length in Chapter 5. For now it is enough to know that an accurate representation of the tail is very important.

The Iterative Feldmann and Whitt Procedure

A simple algorithm has been proposed by Feldmann and Whitt. Given a desired number of phases, it iteratively computes the parameters of each so as to match certain points on the CDF [253]. The procedure starts from the tail and works its way toward the origin, taking into account the phases that have already been defined and matching what is left over. As each phase is an exponential, it has a characteristic scale given by the parameter $\theta = 1/\lambda$. The scale parameter of the last phase matches the end of the tail. All the previous phases have a much smaller scale parameter, and moreover, they decay exponentially. Therefore the last phase is the only one that is relevant for the end of the tail, and all the others can be ignored. After matching the end of the tail, this last phase is subtracted, and the procedure is repeated for the new (shorter) tail.

The procedure is based on the following parameters:

- k — the number of exponential phases to use. Given enough phases (e.g. 20 or so) it is possible to achieve a very close match to the distribution being modeled, but this comes at the price of a more complex model (i.e., a model with more parameters).
- A set $0 < c_k < c_{k-1} < \cdots < c_1$ of points that divide the range of interest into exponentially related subranges. Specifically,
  - c_1 represents the highest values that are of interest; higher values will not appear in our model. The issue of setting an upper bound on the range of interest is problematic and further discussed in Section 5.4.2. For now, we assume that this is somewhat smaller than the largest sample observed in our dataset.
  - c_k represents the smallest values that are of interest. Because we are typically dealing with distributions that have "many small values", c_k should be one of the smaller ones.
  - The ratio c_i/c_{i+1} is set to some constant b. This is determined by the number of phases k and by the scale of interest c_1/c_k, because $c_1/c_k = b^{k-1}$. In other words, we select $b = \sqrt[k-1]{c_1/c_k}$.
- An arbitrary value q that satisfies $1 < q < b$. For example, $q = \sqrt{b}$ could be a good choice. But there is a constraint that $q\,c_1$ must not be larger than the highest data point. (This is why c_1 was chosen earlier to be smaller than the maximum.)

Armed with all these parameters, we set out to model a given survival function $\bar{F}(x) = \Pr(X > x)$. This is done as follows.

1. Initially we match the first phase ($i = 1$) to the tail of the given survival function. In other words, in step 1 we have $\bar{F}_1(x) = \bar{F}(x)$.

2. In general, in step i we match the ith phase to the tail of the remaining survival function $\bar{F}_i(x)$.

   An exponential phase has two parameters, p_i and λ_i. It is characterized by the survival function $\bar{F}_i^{exp}(x) = p_i\, e^{-\lambda_i x}$. To find the values of the two parameters, we equate this survival function to the survival function we are attempting to match at two points: c_i and $q \cdot c_i$. This gives two equations with two unknowns:

$$\begin{aligned} p_i\, e^{-\lambda_i c_i} &= \bar{F}_i(c_i) \\ p_i\, e^{-\lambda_i q\, c_i} &= \bar{F}_i(q\, c_i) \end{aligned}$$

   From the first equation we can easily extract an expression for p_i:

$$p_i = \bar{F}(c_i)\, e^{\lambda_i c_i}$$

   Plugging this into the second equation and simplifying yields

$$\lambda_i = \frac{1}{(1-q)c_i} \ln \frac{\bar{F}(qc_i)}{\bar{F}(c_i)}$$

3. Given the parameters for phases 1 through i, we define the survival function that we need to match in the next step. To do so, we simply subtract the contributions of the first i phases that have already been done from the original survival function:

$$\begin{aligned} \bar{F}_{i+1}(c_{i+1}) &= \bar{F}(c_{i+1}) - \sum_{j=1}^{i} p_j\, e^{-\lambda_j c_{i+1}} \\ \bar{F}_{i+1}(q\, c_{i+1}) &= \bar{F}(q\, c_{i+1}) - \sum_{j=1}^{i} p_j\, e^{-\lambda_j q c_{i+1}} \end{aligned}$$

   We now reiterate step 2.

4. In the last step, when $i = k$, the procedure is slightly different. First, the value of p_k is set so that the probability of all phases equals 1:

$$p_k = 1 - \sum_{j=1}^{k-1} p_j$$

   Therefore only one equation is needed in order to define λ_k. It is

$$p_k\, e^{-\lambda_k c_k} = \bar{F}_k(c_k)$$

   which yields the last missing parameter,

$$\lambda_k = \frac{-1}{c_k} \ln \frac{\bar{F}_k(c_k)}{p_k}$$

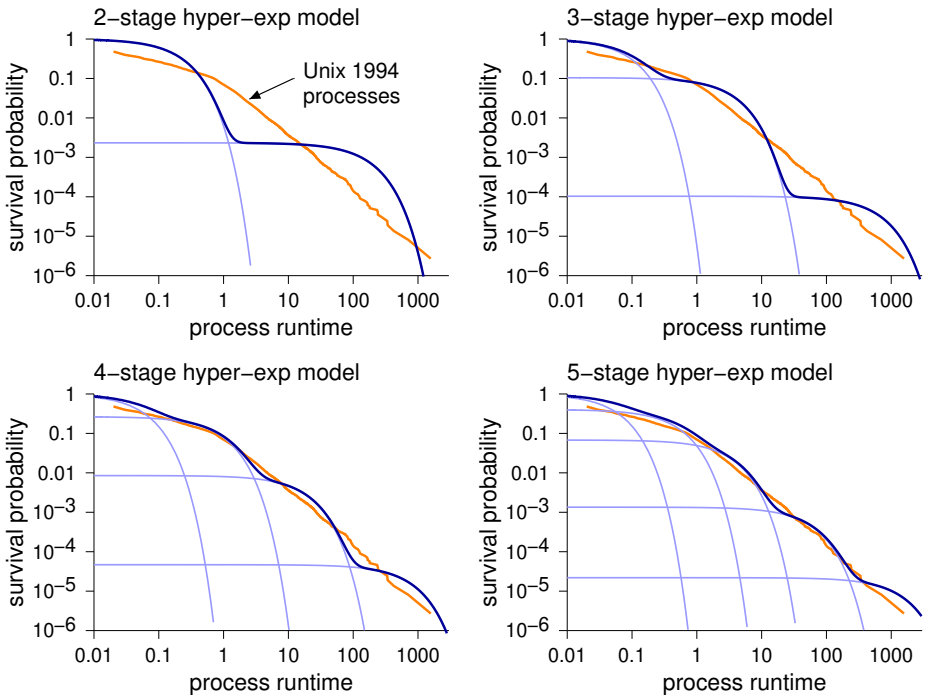


Figure 4.7: *Approximations to a heavy tail using hyper-exponentials with increasing numbers of phases, as computed by the Feldmann and Whitt procedure. These graphs are LLCD plots, which portray the distribution's tail by showing the survival probability on log-log axes; they are explained on page 199.*

As originally described, this procedure should be applied to a known survival function (e.g., to a Weibull distribution) [253]. This implies a two-step modeling procedure: first fit some appropriate distribution to the data, and then approximate it with a hyper-exponential. However, it is also possible to apply the procedure directly using the empirical survival function of the data.

An example of using this procedure is given in Figure 4.7. The dataset is the runtimes of Unix processes from 1994 [320]. These have a heavy tail, and figure prominently in Chapter 5. But a hyper-exponential with as few as four or five stages seems to model the tail of the distribution pretty well.

While this algorithm is simple and direct, it may actually generate a suboptimal result. In particular, it has been criticized as not matching the moments of the data, and as being sensitive to the chosen values of c_i and q [208]. A visual inspection of the result is therefore required, and some experimentation with values of c_i (and especially the largest one, c_1) is recommended.

Using the EM algorithm

The hyper-exponential distribution is a mixture of exponentials. Its parameters can therefore be estimated using the EM algorithm, which is a general technique for finding the parameters of mixtures (as explained above in Section 4.3.2) [41].

The problem is that the EM algorithm only finds parameter values for known distributions. Therefore, when applying it to find the parameters of a hyper-exponential tail, we need to decide in advance how many phases to use. One approach is to start with a small number, even just one (that is, an exponential distribution) [208]. Given this basic version, find the parameter values that give the best fit to the data, and evaluate this fit. Then iterate as follows:

1. Double the number of phases.
2. Find the parameter values that provide the best fit given the new number of phases, and evaluate the quality of the resulting model.
3. If it is not significantly better than the previous model, stop. Otherwise, continue refining the model further.

An alternative method is to first split the data into disjoint ranges, such that each range has a relatively small variability (the suggested criterion is to strive for a CV in the range of 1.2 to 1.5) [569]. Next, model the data in each range independently, using the EM algorithm to find parameters for a hyper-exponential model. Given that the variability in each range is limited, a modest number of phases will suffice (e.g. four phases). The final model is then the union of the separate models for the different ranges, with their probabilities adjusted to reflect the relative weight of each range.

Matching a Complete Distribution

The main drawback of the methods just outlined is that they only handle the tail of a distribution. What this means is that the pdf has to be monotonically decreasing throughout the range. But many actual distributions have a mode near the origin.

To handle such distributions, it has been suggested that a slightly richer phase-type distribution be used. The simplest (yet quite general) approach is to create a mixture of hyper-exponential and Erlang distributions as shown in Section 3.2.6 [568, 341, 342]. The hyper-exponential part contributes the tail, and the Erlang part contributes the mode.

Alternatively, one can simply model the body of the distribution separately, and use the hyper-exponential model only for the tail.

4.5 Tests for Goodness of Fit

The previous sections discussed various ways to find a distribution function that matches given data. But how can we verify that this is indeed a good match? This section presents four approaches: the qualitative graphical method of Q-Q plots, the more rigorous statistical tests devised by Kolmogorov and Smirnov and by Anderson and Darling, and the

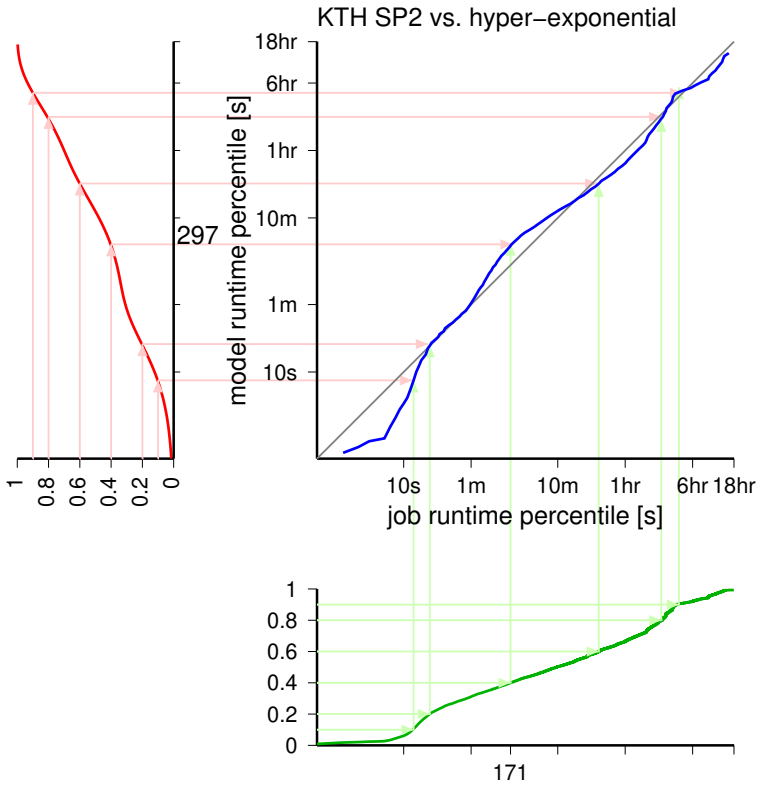


Figure 4.8: *The construction of a Q-Q plot.*

χ^2 method. In essence, what all of them do is to compare two distributions (the data and the model) and decide whether they are actually one and the same.

To read more: As with distribution fitting in general, the coverage of goodness of fit provided by Law and Kelton is especially good [427]. In addition, there are books devoted specifically to this subject, such as the volume edited by D'Agostino and Stephens [162].

4.5.1 Using Q-Q Plots

Q-Q plots are a simple graphical means to compare distributions. The idea is to find the percentiles of the two distributions, and then plot one set as a function of the other set. If the distributions match, the percentiles should come out at the same distances, leading to a straight line with slope 1.

The process is illustrated in Figure 4.8, which compares the distribution of job runtimes on the KTH SP2 machine with a three-stage hyper-exponential model. We start with the CDFs of the two distributions: the empirical CDF of the original data on the bottom, and the CDF of the model on the left. We then generate the Q-Q plot by creating a point for each percentile. Take 0.4 as an example. In the original data, the 0.4

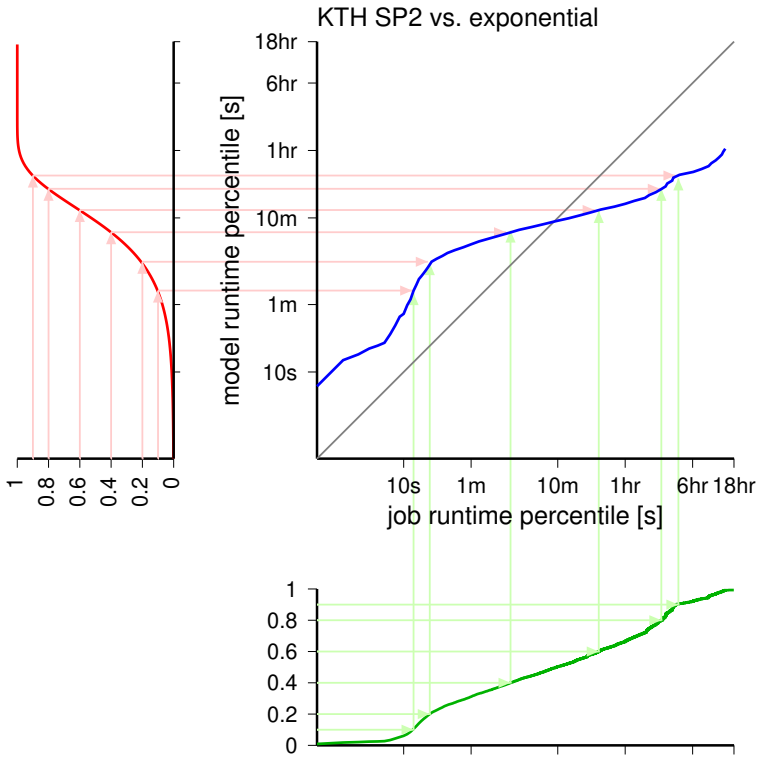


Figure 4.9: *Example of a Q-Q plot where the distributions don't match.*

percentile is achieved at a data value of 171 seconds. In the model, in contrast, this percentile is only achieved at a value of 297 seconds. The corresponding point in the Q-Q plot is therefore drawn at $(171, 297)$, slightly above the diagonal. Overall, the Q-Q plot is pretty close to a straight line with slope 1, indicating that the model is a pretty good match for the data. The main deviation is at the low end of the scale, for jobs that are shorter than 10 seconds.

For comparison, Figure 4.9 shows a Q-Q plot in which the distributions do not match: it compares the same runtime data from KTH to an exponential model. That model is not a good model for this data, so we do not get a straight line with slope 1.

The procedure to construct the Q-Q plot is as follows:

1. Select the samples to use. It is possible to use all the samples that are available, but if there are very many of them it is also possible to use a subsample. One option is then to use a randomly selected subsample. Another is to select a set of percentiles and to use the samples closest to them. For example, if we have 1000 samples, we can sort them and use every tenth sample, each of which represents a percentile of the distribution.

Distribution	X *value*	Y *value*
exponential	$X_{(i)}$	$-\theta \ln(1-p_i)$
Weibull	$\ln(X_{(i)})$	$\frac{1}{\alpha}\ln[-\ln(1-p_i)] + \ln\beta$
Lognormal	$\ln(X_{(i)})$	$\text{sgn}(p_i - 0.5)(1.238\,t + 0.0324\,t^2)$ where $t = \sqrt{-\ln[4p_i(1-p_i)]}$
Pareto	$\ln(X_{(i)})$	$\ln k - \frac{1}{a}\ln(1-p_i)$

Table 4.2: *Formulas for creating Q-Q plots for different model distributions. In all cases* $p_i = \frac{i-0.5}{n}$.

Sort the selected samples with $X_{(i)}$ denoting the ith sample. Denote the number of (selected) samples n.

2. Find the model values that correspond to the samples. This is done in two steps.

   (a) Associate each sample with the percentile it represents. Note that using the fraction i/n leads to an asymmetrical situation, in which the last sample is associated with 1 but no sample is associated with 0. It is therefore more common to use
   $$p_i = \frac{i-0.5}{n}$$

   (b) Calculate the model value that would be associated with this percentile. This is done by inverting the model CDF, as is done when generating random variates from the model distribution. Thus the model value corresponding to sample i is $F^{-1}(p_i)$.

3. Draw the Q-Q plot, using the n points
   $$\left(X_{(i)},\ F^{-1}\left(\frac{i-0.5}{n}\right)\right) \qquad 1 \le i \le n$$

For example, if the model is the exponential distribution, the CDF is $F(x) = 1 - e^{-x/\theta}$, and the inverse is $F^{-1}(p) = -\theta\ln(1-p)$. Each sample $X_{(i)}$ is then associated with a model value $-\theta\ln\left(1 - \frac{i-0.5}{n}\right)$, and these are plotted for all i to generate the Q-Q plot. This and the equations for other distributions are summarized in Table 4.2. The expression for the lognormal distribution is based on an approximation of the normal, because no closed-form expression is available. An alternative approach for this and other cases where $F(x)$ is not easily inverted (such as the hyper-exponential) is to generate a large number of samples from the model distribution, and then compare the two empirical distributions directly.

Practice Box: Interpretation of Q-Q Plots

A Q-Q plot that turns out to be a straight line with slope 1 indicates that the two distributions match. But what can we learn from deviations from this straight line?

The simplest deviation is a straight line with another slope. This means that the shape of the distribution is correct, but the scale is not. If the slope is steeper than 1, the original data is more condensed, and the scale of the model distribution should be reduced. If the slope is flatter than 1, it is the other way around.

Deviations from a straight line indicate that the relative scale is different in different parts of the range. For example, a Q-Q plot that tapers off to the right means that the tail of the data exhibits much higher values than the tail of the model distribution. A model with a heavier tail is therefore needed. Conversely, a Q-Q plot that goes straight up indicates that the tail of the model is too long, and a distribution with a shorter tail is needed.

End Box

An alternative to the Q-Q plot is the P-P plot. The idea is very similar, but instead of creating a point from the two values that correspond to each percentile, one creates a point from the two percentiles that correspond to each value. As a result a Q-Q plot deviates from a straight line due to the horizontal distance between the two compared distributions (that is, between their CDFs), whereas a P-P plot deviates due to the vertical distances. This makes the P-P plot more sensitive to variations in the body of the distribution, where most of the mass is concentrated. A Q-Q plot, in contradistinction, is more sensitive to variations in the tails of the distributions. In the context of workload modeling the tails are often very important, so we prefer to use Q-Q plots.

However, the sensitivity to slight variations in the tails may at times be too much of a good thing, because extreme values from the tail are rare by definition and therefore not very representative. This sensitivity can be reduced by not using the extremal values. For example, if we divide the samples into 100 percentiles, this gives 101 points that define the 100 intervals. It may be better to use the internal 99 points for the Q-Q plot, and ignore the two end points. However, this has the obvious shortcoming of not showing whether or not the tail matches the model. A better option is to use a logarithmic scale, as was indeed done in Figures 4.8 and 4.9.

The sensitivity to tail values is illustrated in Figure 4.10, using the Unix process runtimes dataset from 1994, and comparing it to itself after trimming the top 10 data points (of a total of 184,612). Because the distribution of process runtimes is heavy-tailed, there are very many small samples; in fact, 34% of them are 0.005 seconds, and an additional 17% are 0.02 seconds, but the longest process observed ran for 1573.50 seconds. As a result practically all the data in the Q-Q plot is concentrated at the origin, and the entire plot actually only shows the tail (which in this derived dataset compares the top 10 points to the next 10). If only the first 99 percentiles are used, much of the data is still concentrated at the origin, but we see that the rest follows a straight line. However, this comes at the expense of eliminating all information about the tail. Using a logarithmic scale is a good compromise, because it shows both the body of the distribution and its tail.

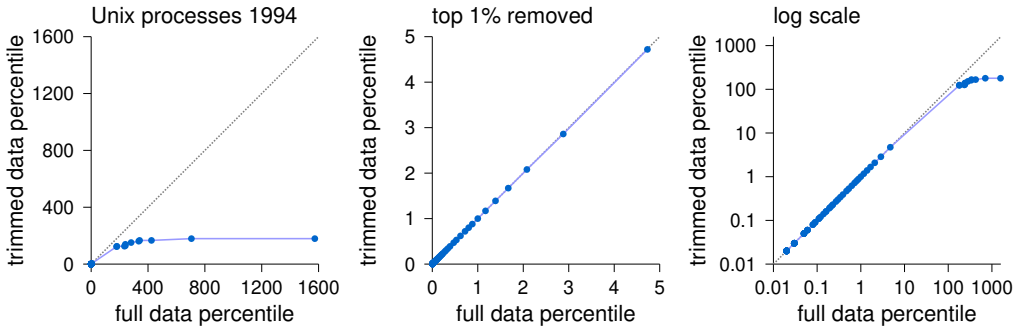


Figure 4.10: *Q-Q plots may be dominated by tail values (left). This effect can be eliminated by not showing the tail (center) or reduced by using a logarithmic scale (right). In all these plots there is one data point per percentile, plus the 10 largest values.*

4.5.2 The Kolmogorov-Smirnov Test

The Kolmogorov-Smirnov test is based on calculating the *maximal* distance between the cumulative distribution function of the theoretical distribution and the sample's empirical distribution, over all the sampled values [659]. This is done as follows:

1. Sort the sampled observations to obtain the order statistics $X_{(1)} \dots X_{(n)}$ such that $X_{(1)} \leq X_{(2)} \dots \leq X_{(n)}$.
2. Define the samples' empirical cumulative distribution function:

$$F_n(x) = \begin{cases} 0 & \text{if } x < X_{(1)} \\ i/n & \text{if } X_{(i)} \leq x < X_{(i+1)} \\ 1 & \text{if } x \geq X_{(n)} \end{cases}$$

   The empirical CDF $F_n(x)$ is an estimator for the theoretical distribution $F(x)$. Furthermore, if the samples are indeed from the theoretical distribution $F(x)$, then $\Pr(\lim_{n\to\infty} |F(x) - F_n(x)| = 0) = 1$. Note that we assume that $F(x)$ is a continuous distribution, and therefore there is a negligible probability that a given value will be sampled more than once.

3. Define $D_n = \sup_x \{|F(x) - F_n(x)|\}$. Since $F_n(x)$ is a step function with all steps of height $1/n$, this is equivalent to

$$D_n = \max_{i=1\dots n} \left\{ \left| \frac{i}{n} - F(X_{(i)}) \right|, \left| F(X_{(i)}) - \frac{i-1}{n} \right| \right\} \tag{4.5}$$

   For each data point, we need to check if the distance is greatest to its left or to its right.

4. If the value of D_n is large, we will reject the hypothesis that the theoretical and empirical distributions are the same. If D_n is small enough we will not reject that hypothesis. The actual threshold to use depends on the sample size n and the desired confidence level α, and can be found in statistical tables [474, 659].

The justification for focusing on the maximal difference is the following. Consider two samples of the same size from the same distribution, and draw their empirical CDFs. Given that the two samples come from the same distribution, the CDFs are expected to be close to each other and even cross each other. The precise dynamics can be described in terms of a random walk: at each sample, the distance between the CDFs either grows or shrinks by one step. But at both ends the distance is constrained to be 0 (thus this random walk is not like Brownian motion, but rather like a so-called "Brownian bridge"). The greatest divergence is therefore expected in the middle, and assuming n samples it should be proportional to $\sqrt{n}$. The Kolmogorov-Smirnov test flags a mismatch (the distributions do not fit) if the maximal expected divergence is exceeded to such a degree that it cannot be ascribed to chance fluctuations.

Of course, the test comes with some statistical fine print. The main caveat is that the parameters of $F(x)$ have to be known in advance, and are not estimated from the data [427]. But in our case the whole point is that the parameters are *not* known in advance, and we need to estimate them from the data. When this is the case, specialized tables should be used [659], which exist only for select distributions. Using the general table leads to a conservative test.

An alternative approach that solves this problem is to use bootstrapping [480, 205, 206, 178, 212]. In essence, this creates a threshold value tailored to the model and the number of samples being considered. The idea is simple: given the model distribution $F(x)$, generate n samples from it. Now apply the Kolmogorov-Smirnov test to these samples, which are known to come from the correct model. Repeat this a large number of times, e.g. 1000 times. This gives us 1000 samples from the distribution of the K-S metric for the situation at hand. We can now see where the K-S metric of the real data falls in this distribution. If it falls in the body of the distribution, it is reasonable to assume that the data is compatible with the model $F(x)$. But if the metric falls in the extreme tail of the distribution, we know that there is only a low probability that the data comes from $F(x)$.

Another problem with applying the K-S test to computer workload data is that it always fails. The reason is that statistical tests are designed for moderate numbers of samples, from several dozens to maybe a thousand. In computer workloads, however, we may have from tens of thousands to millions of samples. As the tests require greater precision when there are more samples, but the data never really comes from a mathematical distribution, the test fails. The way to circumvent this problem is to check only a subsample of the data. For example, randomly select 500 of the data points, and check whether they conform to the proposed distribution.

Finally, the Kolmogorov-Smirnov test is rather insensitive to the tails of the distribution. The CDF in the tail is invariably close to 1, so the differences between the distribution's CDF and the empirical CDF are small. The test result is therefore determined by the body of the distribution, where most of the data resides, and not by the tail. Using this test may mislead us into thinking that a distribution provides a good model, despite the fact that it does not model the tail appropriately.

4.5.3 The Anderson-Darling Test

The Anderson-Darling test is a variation on the Kolmogorov-Smirnov test, in which more emphasis is placed on the tails of the distribution [29, 660]. This is done by computing the weighted mean of the differences between the two CDFs, rather than the maximal difference. The weight is defined to be $\frac{f(x)}{F(x)\,(1-F(x))}$, so that it is big wherever the distribution has significant mass and in either tail; in the case of workloads, only the right tail is typically important. In addition, the difference between the CDFs is squared to ensure that it is positive. The test statistic is thus

$$A^2 = n \int_{-\infty}^{\infty} (F_n(x) - F(x))^2 \, \frac{f(x)}{F(x)\,(1-F(x))} \, \mathrm{d}x \tag{4.6}$$

where $F(x)$ is the CDF of the model distribution. To calculate this, divide the range into $n+1$ segments using the order statistics as endpoints: $-\infty$ to $X_{(1)}$, $X_{(1)}$ to $X_{(2)}$, $X_{(2)}$ to $X_{(3)}$, and so on. In each such range $F_n(x)$ is constant. By integrating and collecting terms it is then possible to derive the expression

$$A^2 = -n - \frac{1}{n} \sum_{i=1}^{n} (2i-1) \left(\ln[F(X_{(i)})] + \ln[1 - F(X_{(n-i+1)})] \right)$$

This statistic is then compared with values from a table to determine whether the data indeed fit the proposed distribution to a given degree of confidence [659].

4.5.4 The χ^2 Method

When $F(x)$ is not available in the form of a simple equation, the alternative is to create random samples from this distribution, and then to check whether it is reasonable to say that the original data and these samples come from the same distribution. This is done using the χ^2 (read: "kai squared") test. The number of samples generated should equal the number of original data observations.

In this test, the range of values is partitioned into a certain number k of subranges [502]. Then the number of observations that fall into each range (O_i) and the expected number of observations that *should* fall into each range (E_i) are tabulated. If $F(x)$ is known, and denoting the beginning of range i by b_i, this is $E_i = n(F(b_{i+1}) - F(b_i))$. But in our case E_i is not based on the formula, but rather on the created random samples. The metric is then

$$\chi^2 = \sum_{i=1}^{k} \frac{(O_i - E_i)^2}{E_i} \tag{4.7}$$

This is then compared with statistics tables to ascertain if it is small enough to warrant the conclusion that the distributions are probably the same. Alternatively, the bootstrap method can be used as explained above.

The drawback is that there should be a minimal number of samples in each range (e.g. at least five, preferably more). This means that there will only be a small number of rather large ranges representing the tail of the distribution. Thus this method is not very good for comparing heavy-tailed distributions.

4.6 Software Packages for Distribution Fitting

A few commercial software packages exist that automatically test data against some 40 distributions, find optimal parameters values, and suggest which provides the best fit. However, it is necessary to verify the results by inspecting the produced graphs (e.g. Q-Q plots). These software products cannot handle mixtures of distributions, and in particular do not include the hyper-exponential distribution.

- ExpertFit from Averill Law and associates. This is based on Law's textbook, which provides a detailed and extensive treatment of distribution fitting [427]: http://www.averill-law.com/distribution-fitting/
- Stat::Fit from Geer Mountain Software Corp.: http://www.geerms.com/
- EasyFit from MathWave Technologies: http://www.mathwave.com/
- @Risk from Palisade Corp., which includes a distribution fitting package that used to be available separately under the name BestFit: http://www.palisade.com/risk/

In addition, some research groups provide software for distribution fitting. Among them are a tool for fitting phase-type distributions to heavy-tailed data called PhFit (available from http://webspn.hit.bme.hu/%7Etelek/tools.htm) [342], and another called EMpht that uses the EM algorithm (available from http://home.imf.au.dk/asmus/pspapers.html) [41].

5

Heavy Tails

There are several advanced topics in statistical modeling that are typically not encountered at the introductory level. However, they reflect real-life situations, so they cannot be ignored. Probably the most prominent of them is heavy tails.

In fact, a very common situation is that distributions have many small elements and few large elements. The question is how dominant are the large elements relative to the small ones. In heavy-tailed distributions, the rare large elements (from the tail of the distribution) dominate, leading to a skewed distribution. Examples where this is the case include the following:

- The distribution of process runtimes [320].
- The distribution of file sizes in a file system or retrieved from a web server [361, 155, 57, 188].
- The distribution of popularity of items on a web server, and the distribution of popularity of websites [57, 85, 572, 9, 525].
- The distribution of flows in the Internet, both in terms of size and in terms of duration [610, 215].
- The distribution of think times for interactive users [146].
- The distribution of the number of queries sent by a peer on a P2P network, and the distribution of session lengths [312].
- The distribution of register instance lifetimes (the time until a value in a register is overwritten) [198].
- The distribution of in and out degrees in a social network [492].
- The distribution of in and out degrees in the Internet topology graph [226, 89].

The qualifier "rare" regarding the tail elements is actually somewhat confusing. For example, the most popular page on a web server, the one that dominates the downloads, is popular rather than rare when we look at the list of downloads. But it is rare if we look at the population of web pages: there is only one such page. This duality leads to the effect of mass-count disparity, discussed in Section 5.2.2.

Heavy-tailed data is not new. To quote from the preface of *A Practical Guide to Heavy Tails* [16],

> Ever since data has been collected, they have fallen into two quite distinct groups: "good data", which meant that their owners knew how to perform the analysis, and "bad data", which were difficult, if not impossible, to handle. ... This, in short, was data with heavy-tailed histograms.

Of course, heavy-tailed distributions are not limited to data regarding computer usage. Heavy-tailed data has been observed in a variety of natural and human-related phenomena [442, 514, 566, 142]. The main problem with such data is deciding how to deal with the sparsity of data from the tail, and, specifically, how to model and extrapolate it. For example, if so far we have only recorded earthquakes of magnitudes up to 9.5 on the Richter scale, what can we say about the probability of an earthquake of magnitude 9.7? The same applies for computer workloads, where we need to consider the effect of rare events such as extremely long jobs, extremely large Internet flows, and so on.

To read more: Papers with good background material on heavy tails include the survey by Newman [514], the classic regarding network traffic by Paxson and Floyd [540], and the overview by Park and Willinger [538]. The volume edited by Adler et al. [16] is devoted to dealing with heavy tails, but despite its name (*A Practical Guide to Heavy Tails*) it is still more on the mathematical than the practical side. An even more general and theoretical mathematical treatment is given by Samorodnitsky and Taqqu [590].

5.1 The Definition of Heavy Tails

Heavy tails are most commonly defined as those governed by power laws, and most of this chapter is devoted to such tails. But other definitions also exist.

5.1.1 Power-Law Tails

The tail of an exponential distribution decays, well, exponentially. This means the probability of seeing large values decays very fast. In heavy-tailed distributions, this probability decays more slowly. Therefore large values are distinctly possible.

Formally, it is common to define heavy-tailed distributions as those whose tails decay like a power law — the probability of sampling a value larger than x is proportional to 1 over x raised to some power [540]:

$$\bar{F}(x) = \Pr(X > x) \propto x^{-a} \qquad 0 < a \leq 2 \tag{5.1}$$

where $\bar{F}(x)$ is the survival function (that is, $\bar{F}(x) = 1 - F(x)$), and we use $\propto$ rather than $=$ because some normalization constant may be needed. The exponent a is called the tail index. The smaller it is, the heavier the tail of the distribution.

Saying that the tail probability decays polynomially is a very strong statement. To understand it, let us compare the tail of a heavy-tailed distribution with the tail of an exponential distribution. With an exponential distribution, the probability of sampling

a value larger than, say, 100 times the mean is e^{-100}, which is totally negligible[1]. But for a Pareto distribution with $a = 2$, this probability is $1/40,000$: one in every 40,000 samples will be bigger than 100 times the mean. Although rare, such events can certainly happen. When the shape parameter is $a = 1.1$, and the tail is heavier, this probability increases to one in 2,216 samples. This propensity for extreme samples has been called the *Noah effect* [466, p. 248], referring to one of the earliest cases on record: the 40 days and nights of rain of the deluge described in the book of Genesis.

The range of a is typically limited to $0 < a \leq 2$. When $a > 2$, this is still a power law, but it has less striking characteristics. In particular, its mean and variance are finite, and the central limit theorem applies (this is defined and explained later). In practical terms, however, the distribution's tail still decays slowly even for large as.

5.1.2 Properties of Power Laws

The fact that a distribution has a power law tail immediately sets it apart from the distributions encountered in basic probability courses. For example, if the tail index is small enough, the distribution does not have a mean, and the central limit theorem does not hold, at least not in its simple and best-known version.

Infinite Moments

Consider data sampled from an exponential distribution. A running average of growing numbers of data samples then quickly converges to the mean of the distribution. But heavy-tailed data is ill behaved: it does not converge when averaged, but rather continues to grow and fluctuate. In fact, the mean and variance might actually tend to infinity as more samples are added.

How can a distribution have an infinite mean? It is best to start with a simple example. Consider a distribution over powers of two, which is defined thus:

2 with a probability of 1/2
4 with a probability of 1/4
8 with a probability of 1/8
16 with a probability of 1/16
32 with a probability of 1/32
and so on.

This is a probability distribution function because the probabilities sum to 1:

$$\frac{1}{2} + \frac{1}{4} + \frac{1}{8} + \frac{1}{16} + \cdots = \sum_{i=1}^{\infty} \frac{1}{2^i} = 1$$

The expected value is calculated by multiplying each possible value by its probability and summing up. This gives

[1] Assume you can observe a billion events per second. This is 3.15×10^{16} events a year. At this rate, your chance of observing an event that occurs with probability e^{-100} within the current age of the universe is only 0.000000000000000016.

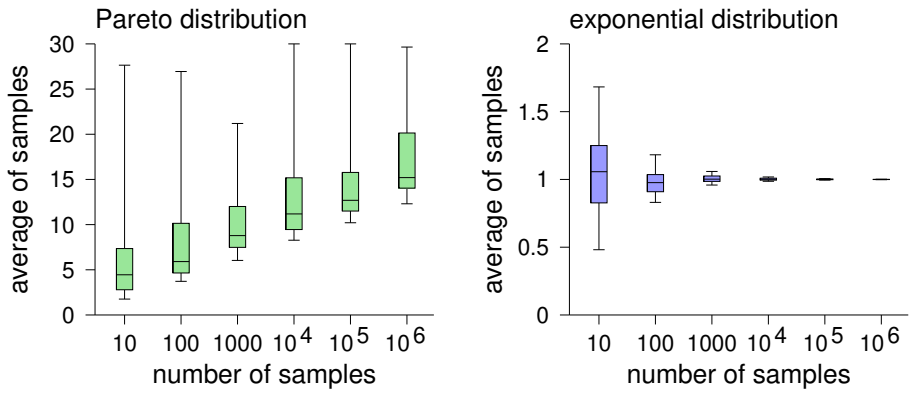


Figure 5.1: *The averages of growing numbers of samples from a Pareto distribution with $a = 1$ grow logarithmically, as opposed to the averages of an exponential distribution which converges according to the law of large numbers.*

$$\frac{1}{2}\,2 + \frac{1}{4}\,4 + \frac{1}{8}\,8 + \frac{1}{16}\,16 + \cdots = \sum_{i=1}^{\infty} \frac{1}{2^i}\,2^i = \sum_{i=1}^{\infty} 1 = \infty$$

The result is infinity, because as the probabilities go down, the values go up fast enough to compensate and make a non-negligible contribution to the sum.

With heavy-tailed distributions, not only the mean but other moments as well may be infinite. Specifically, using the definition stated above, if $a \leq 1$ the mean will be infinite, and if $a \leq 2$ the variance will be infinite. More generally, for each integer k, if $a \leq k$ then the first k moments are infinite.

Of course, even when the mean (or another moment) of a distribution is infinite, we can still calculate it for a set of samples. How does this mean behave? Consider a Pareto distribution with $a = 1$, whose probability density is proportional to x^{-2}. Trying to evaluate its mean leads to

$$\mu = \int_1^{\infty} cx\,\frac{1}{x^2}\,\mathrm{d}x = c \ln x \,\Big|_1^{\infty}$$

so the mean is infinite. But for any finite number of samples, their average obviously exists. The answer is that the average grows logarithmically with the number of observations (note that this is for the specific case of $a = 1$; for $a < 1$ it grows faster, and for $a > 1$ it grows slower). If we only have a few samples from the distribution, the average will most probably be relatively small. As more samples are added, we have a bigger chance of sampling from the tail of the distribution. Such samples from the tail dominate the sum. Therefore the average grows when they are added. With more and more samples, we delve deeper and deeper into the tail of the distribution, and the average continues to grow.

The above is visualized in Figure 5.1. Start by creating 10 independent samples from a Pareto distribution, and calculate their average. Repeat this 100 times. This gives 100

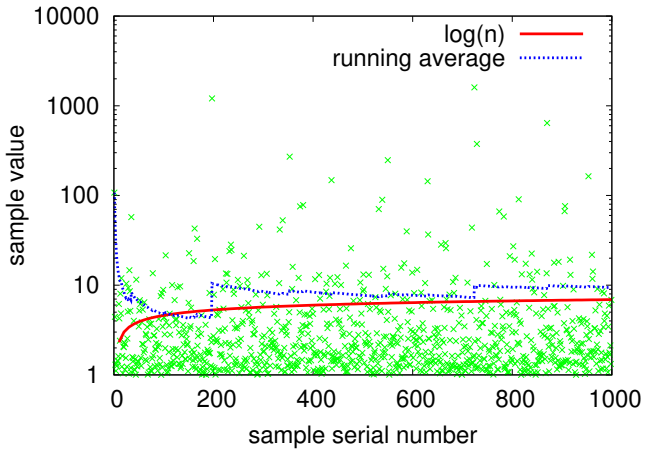


Figure 5.2: *Example of the running average of samples from a Pareto distribution.*

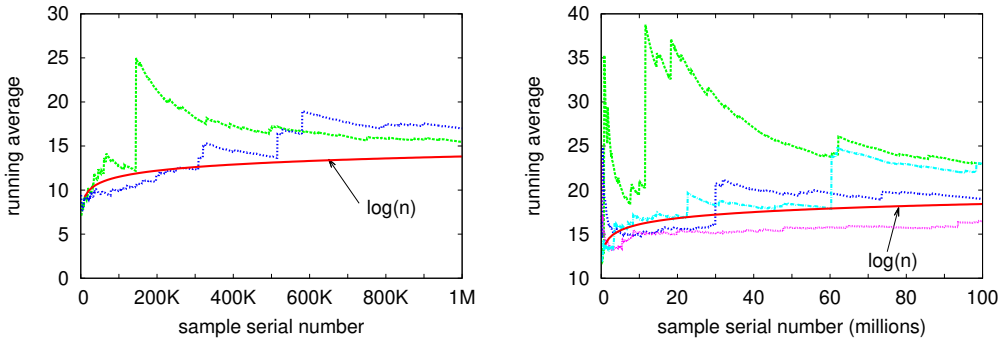


Figure 5.3: *More examples of the running average of samples from a Pareto distribution, over a long range and using different random number generator seeds.*

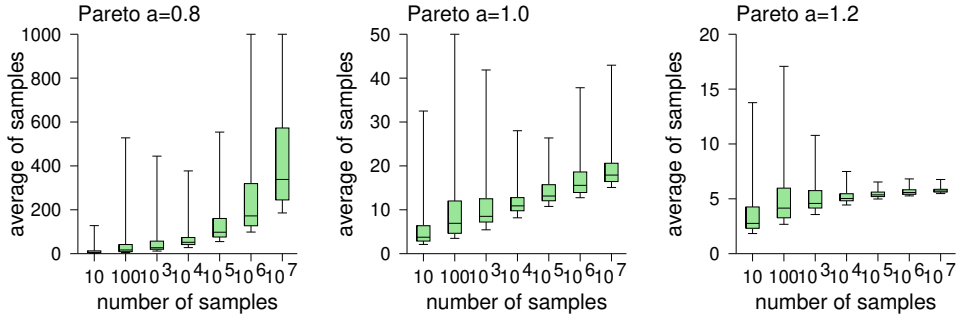
averages of 10 samples each, and we can characterize the distribution of such averages using a box plot. Now do this for larger numbers of samples: 100 samples, 1000 samples, and so on up to a million samples. As shown in the figure, the distribution of averages shifts up in direct proportion to the log of the number of samples, and moreover, it remains disperse and skewed. With an exponential distribution, in contradistinction, the distribution of averages quickly converges to the mean, in this case 1.

However, if we follow the running average of increasing numbers of samples, it does not actually resemble the log function. In fact, it grows in big jumps every time a large observation from the tail of the distribution is sampled, and then it slowly decays again toward the log function. An example is found in Figure 5.2, which shows a thousand samples from a Pareto distribution along with their running average. Most of the samples are small, rarely more than 10, and concentrate near the X axis. But sample number 197 is an exceptionally high value: 1207.43. This is big enough to contribute 6.13 to the running average, which indeed jumps up by this amount.

The same thing continues to happen as we add more and more samples (Figure 5.3). We continue to see samples from the tail of the distribution that are big enough to dominate all of the previous samples. Because these samples are unique and rare events, their occurrence depends on the random number generator and the specific seed being used. Thus different runs do not converge, and huge differences in the running average can be observed even after 100,000,000 samples or more.

Details Box: Different Values of a

The above discussion alludes to logarithmic growth, but this is actually a special case that occurs only for a Pareto distribution with $a = 1$. It is also a boundary case, because when $a > 1$ the average actually exists and adding more and more samples will eventually lead to convergence to this average. But when $a < 1$, in contradistinction, the averages diverge more wildly. This is illustrated in the following extension of Figure 5.1:



End Box

The Law of Large Numbers

The law of large numbers states that, given n samples drawn from a distribution with a mean μ, the average of the samples converges to the mean as n grows. This makes it easy to estimate the mean of an unknown distribution: just calculate the average of enough samples.

But if the mean is infinite, the law of large numbers obviously does not apply. The average of many samples will not converge to the mean, because there is no mean to which to converge (Figure 5.1). Even if the mean is in fact finite (which happens if the tail index satisfies $a > 1$), convergence may be very slow. In practice, the borderline between convergence and lack thereof is not sharp [157, 307, 302]. Very close to the borderline, an extremely large number of samples may be needed in order to converge to the true value. The average may seem to converge, but then another sample from the tail of the distribution will cause it to change.

A rough estimate of how many samples are needed is given by Crovella and Lipsky [157]. As noted below, it is well known that the sum of n random variables with finite variance tends to a normal distribution, and that the width of this distribution is proportional to $\sqrt{n}$. The generalization for variables with an infinite variance, such as Pareto variables with $1 < a < 2$, is that

$$\left| \frac{1}{n} \sum_{i=1}^{n} X_i - \mu \right| \propto n^{(\frac{1}{a}-1)}$$

(note that for $a = 2$ we indeed get $1/\sqrt{n}$). Now assume we want two-digit accuracy, which means that we require

$$\frac{\left|\frac{1}{n} \sum_{i=1}^{n} X_i - \mu\right|}{\mu} \leq 10^{-2}$$

Putting the two expressions together and ignoring constants (including the specific value of the mean μ), we then get

$$n^{(\frac{1}{a}-1)} \leq 10^{-2}$$

Switching sides by inverting the signs of the exponents and then raising both sides to the power $1/(1 - \frac{1}{a})$ leads to the approximation

$$n \geq 10^{2/(1-\frac{1}{a})}$$

For $a = 2$, this indicates that 10,000 samples are needed to achieve two-digit accuracy. For $a = 1.5$, the number grows to 1,000,000. As $a \to 1$ this grows asymptotically to infinity; at $a = 1.1$ it is already 10^{22}.

So what happens if we are simulating a system in which job runtimes are heavy-tailed, with a low value of the tail index a? Trying to find average performance results is doomed to fail, because the average does not converge (either at all or within a reasonable time frame). In effect, the simulation is perpetually in a transient state, never reaching a steady state [156]. And the average results we obtain depend on the random number generator and the length of the simulation.

It should be noted that the divergence of the moments is a direct consequence of the power-law tail. Thus if the tail is truncated, as in the truncated version of the Pareto distribution, all the moments become finite. In particular, the law of large numbers now applies. But in practice, if the truncation is at a very high value, convergence may still be very slow [307].

Background Box: Convergence in Probability and Convergence in Distribution

Convergence in probability deals with the convergence of a series of random variable $X_1, X_2, X_3, \ldots$ to a finite limit b. For every $\epsilon > 0$, we find the probability that the difference between X_n and b is smaller than ϵ. The claim is that this probability tends to 1 as n tends to ∞, or symbolically

$$\lim_{n\to\infty} \Pr(|X_n - b| < \epsilon) = 1$$

A related concept is convergence in distribution. This deals with a set of distributions that converge to a limit distribution. Thus if we have distributions $F_1(x), F_2(x), F_3(x)$, and so on, we say that they converge to a limit distribution $F^*(x)$ if

$$\lim_{n\to\infty} F_n(x) = F^*(x)$$

for all x where $F^*(x)$ is continuous.

Note the difference between the two notions. Convergence in probability deals with the convergence of random variables. In the case of the law of large numbers, the random variables are the averages of increasing numbers of samples: $\bar{X}_n = \frac{1}{n}\sum_{i=1}^{n} X_i$. The law essentially states that, as more samples are included, the estimate they provide for the distribution's mean improves. Convergence in distribution, in contrast, deals with complete distributions. This lies at the base of the central limit theorem.

End Box

Limit Theorems

The central limit theorem is widely considered to be one of the most important results in probability theory. Start with a set of independent random variables $X_1, X_2, \ldots$ sampled from any distribution with finite variance. Now create another random variable that is their average. Then the distribution of this average will converge to a normal distribution. The problem lies in the fine print: the original distribution must have a finite variance, and heavy-tailed distributions do not fulfill this requirement. The central limit theorem therefore does not apply.

However, generalizations of the central limit theorem do apply to such distributions. A more general version is that the appropriately scaled sum of random variables converges to a *stable distribution*. The normal distribution is a special case, but there are others. In particular, the Pareto distribution (for $a \leq 2$) converges to what is known as an α-stable distribution, which has a Pareto tail, and specifically, one characterized by the same tail index a.

Figure 5.4 shows and contrasts the convergence of samples from exponential and Pareto distributions. In all cases, the distribution's mean is 1, the average of 100 independent samples is computed, and this is repeated 10,000 times. For the exponential samples, the distribution of the obtained averages is indeed normal. The average of the 100 samples can be as low as 0.75 or as high as 1.25, but in most cases it is very close to 1. For the Pareto samples, the distribution of the averages depends on the tail index. When it is 2, the distribution is close to normal (it would be closer if we averaged over more than 100 samples). But as the tail index goes down toward 1, the distribution of the averages becomes more skewed, and the mode becomes distinctly smaller than the mean of the underlying Pareto distribution. This occurs because the Pareto distribution itself is very skewed and most of the samples are small. So when we take 100 samples, they are typically small, leading to a small average. (When $a = 1.15$, the average of 100 samples is typically around 0.5!) But occasionally the set of 100 samples includes one from the tail of the Pareto distribution, and then the average is very large, and comes from the tail of the α-stable distribution. Therefore the distribution of averages has a heavy tail just like the initial Pareto distribution. That the tails are indeed different is shown in the right-hand graph, using log-log complementary distribution plots (these are explained later on page 199).

But if the typical observation of an average is so much smaller than the true mean, why is the running average of the samples typically higher (as shown in Figure 5.3)? The answer is that in the vast majority of cases the average of an isolated subset of, say, 100 samples, is indeed small. But once we hit on a large sample as part of a long

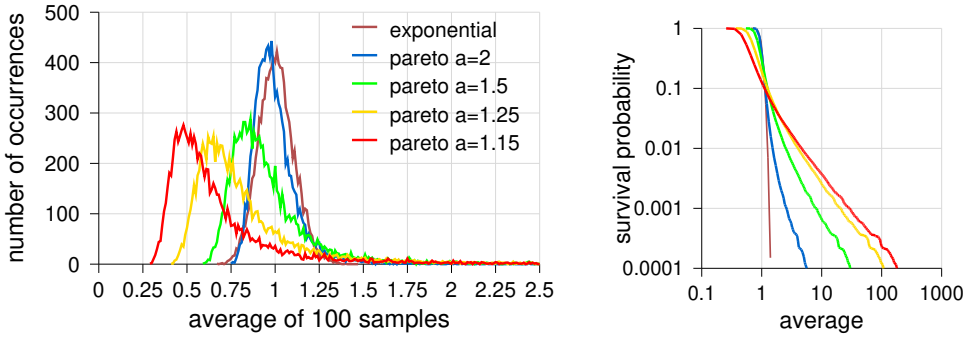


Figure 5.4: *Left: distributions of the averages of 10,000 sets of 100 samples from exponential and Pareto distributions. Right: LLCD plots of the tails of these distributions.*

sequence, this affects the running average for a long time. The subsequent samples may all be small, but when combined with the large one that came before them, we get a large running average.

Background Box: Stable Distributions

Stable distributions are defined by the following procedure. Take n independent samples from a distribution. Perform some operation on them to derive a single new random variable. If the distribution of this newly produced random variable is the same as the original distribution, the distribution is stable.

Depending on the operation that is applied, different distributions can be shown to be stable. Examples include the following [495]:

Summation — The distribution of the sum of n samples is the same as that of a single sample, with possible adjustments of scale and location:

$$X_1 + X_2 + \cdots + X_n \stackrel{\text{d}}{\sim} c_n X_1 + b_n$$

This is the most commonly discussed type [520]. Distributions that satisfy this equation are the normal distribution and the α-stable distributions. Note that the adjustments may depend on n. In fact, the only possible scaling constants are $c_n = n^{1/a}$. The normal distribution corresponds to the special case where $a = 2$, so $c_n = \sqrt{n}$.

Maximum — The distribution of the maximum of n samples is the same as that of a single sample, with possible adjustments of scale and location:

$$\max\{X_1, X_2, \ldots, X_n\} \stackrel{\text{d}}{\sim} c_n X_1 + b_n$$

The extreme value distribution satisfies this equation.

Multiplication — The distribution of the product of n samples is the same as that of a single sample, again with the appropriate adjustments:

$$X_1 X_2 \cdots X_n \stackrel{\text{d}}{\sim} X_1^{c_n} b_n$$

This equation is satisfied by the lognormal distribution.

An important characteristic of stable distributions is that they each have a basin of attraction. Thus you can perform these operations on samples from another distribution, and as n grows you will converge to the stable distribution. It is well known that the normal distribution is the limit of all distributions with a finite variance, with respect to summing independent samples. But other stable distributions are also the limit of sets of related distributions. For example, the α-stable distribution is the limit of distributions with a Pareto tail. Indeed, the Pareto distribution is especially interesting, because it passes from one basin of attraction to another depending on its tail index. For $a \leq 2$ its variance is infinite. In this case it converges to the α-stable distribution when aggregated. But if $a > 2$ the variance is finite, and then summed samples converge to the normal distribution.

The α-stable distribution is characterized by no less than four parameters [519]. In general the distribution has a single mode and possibly heavy tails. The parameters are

- α is the index of stability, which characterizes the distribution of mass between the body of the distribution and its tails. For $\alpha < 2$ the tails are Pareto distributed with tail index α themselves.
- β is a skewness parameter, assuming values in the range $[-1, 1]$. $\beta = 0$ implies symmetry, and $\beta = \pm 1$ maximal skewness in either direction.
- μ is a location parameter (but not the mean or mode).
- σ is a scale parameter (but not the standard deviation).

To read more: Stable distributions are not mentioned in basic probability textbooks. An introduction and some intuition are provided by Crovella and Lipsky [157]. Stable distributions are central to the discussion of Feller [254], with the basics introduced in Chapter VI, and additional material presented later in the book. Even more advanced material on stable distributions and their relationship to self-similarity is found in *A Practical Guide to Heavy Tails* [16] and in the book by Samorodnitsky and Taqqu [590]. An extensive treatise on stable distributions is being prepared by Nolan [520]. The original development of the theory is largely due to Lévy.

End Box

Scale Invariance

What happens when we change the scale of observation? Changing the scale simply means that we use a different unit of measurement; in mathematical terms it is just the transformation of multiplying by a constant, and replacing x with mx. Does this change the shape of the tail of the distribution? Plugging the transformation into the definition, we get

$$\begin{aligned} \bar{F}(mx) = \Pr(X > mx) &\propto (mx)^{-a} \\ &\propto m^{-a} x^{-a} \end{aligned}$$

This requires re-normalization, but is of exactly the same functional form. The tail decays with x in exactly the same manner as before: it is a power law with the exponent $-a$. Thus this distribution is scale invariant — we can change the scale by a factor of m, which might be large, and still observe the same behavior.

In contrast, other distributions are not scale invariant. For example, if we apply the same transformation to the exponential distribution we get

$$\begin{aligned}\bar{F}(mx) = \Pr(X > mx) &\propto e^{-(mx)/\theta} \\ &\propto e^{-x(m/\theta)}\end{aligned}$$

This changes the way in which the tail decays. Instead of decaying at a rate that is proportional to θ, it now decays at a rate that is proportional to θ/m. The parameter of the distribution actually specifies its characteristic scale. For an exponential distribution, this scale is the range over which the value changes by $1/e$. Scaling changes this attribute of the distribution.

Connection Box: Pink Noise

Noise is characterized by its power spectrum: the power of the different frequencies that make up the noise (see the explanation of the periodogram on page 351). White noise has the same power at all frequencies. Pink noise has decreasing power at higher frequencies, and specifically, the power is inversely proportional to the frequency; it is therefore also called $1/f$ noise. Alternatively, it can be described as having equal power in bands of frequency that constitute octaves (i.e., such that the top frequency is double the bottom one).

What happens if you record pink noise on a tape, and then play it back at a different speed? When you change the speed of the tape by a factor r, all the frequencies change by the same factor r. But if the spectrum is described by a power law, it is scale invariant, and the shape of the spectrum does not change! Therefore the noise will sound exactly the same — only slightly stronger or weaker, depending on whether the tape was sped up or slowed down [602]. In contrast, human voices, music, and non–power-law noises will sound different when played back at the wrong speed, with their characteristic pitch altered in correspondence with the change in speed.

End Box

An interesting property of scale-invariant functions is that they have the product property [207]. Thus, the two following statements are equivalent:

1. $f(x)$ is scale invariant, meaning that $f(mx) = c_1 f(x)$.
2. $f(x)$ has the product property, meaning that $f(xy) = c_2 f(x) f(y)$.

It is easy to verify that these indeed hold for power laws of the form $f(x) = ax^b$.

5.1.3 Alternative Definitions

Although it is common to define heavy-tailed distributions as those having a tail that decays according to a power law, this is not the only definition. A chief contender is subexponential distributions [290], sometimes also called *long-tailed* [253]. This name expresses the idea that the tail decays more slowly than exponentially. To explain this, we consider the effect of multiplying the survival function by a factor that grows exponentially. If the result diverges, it means that the exponential growth of the added factor is "stronger" than the decay of the original tail, and hence that the tail decays more slowly than exponentially.

In more detail, the survival function characterizes the tail because it is the complement of the CDF: $\bar{F}(x) = 1 - F(x) = \Pr(X > x)$. We multiply this by an exponential

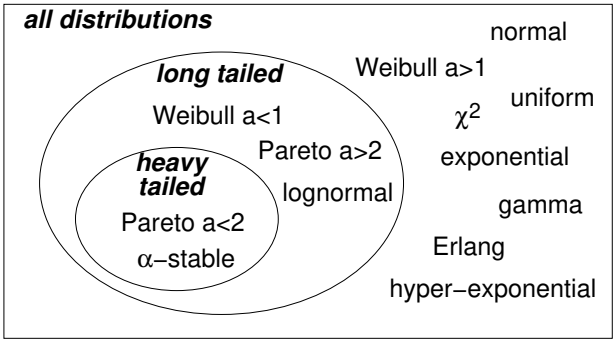


Figure 5.5: *Inclusion relations of classes of distributions with different tails.*

factor $e^{\gamma x}$, with $\gamma > 0$. One possible outcome is that the result tends to 0 as x grows, at least for some value of γ. This means that the tail decays fast enough to compensate for the added exponential factor, or, in other words, that the tail decays exponentially (or faster). For example, for the exponential distribution we have $\bar{F}(x) = e^{-x/\theta}$. Therefore

$$e^{\gamma x}\bar{F}(x) = e^{\gamma x}e^{-x/\theta} = e^{(\gamma - 1/\theta)x}$$

and the limit is

$$\lim_{x\to\infty} e^{\gamma x}\bar{F}(x) = \lim_{x\to\infty} e^{(\gamma-1/\theta)x} = \begin{cases} 0 & \text{for } \gamma < 1/\theta \\ 1 & \text{for } \gamma = 1/\theta \\ \infty & \text{for } \gamma > 1/\theta \end{cases}$$

Because there exist values of γ for which the limit is 0, the exponential distribution is short-tailed.

The other possible outcome is that the result diverges for *all* values of γ. For example, consider the Weibull distribution, which has a survival function $\bar{F}(x) = e^{(-x/\beta)^\alpha}$. In the case that $\alpha < 1$, the limit is

$$\lim_{x\to\infty} e^{\gamma x}\bar{F}(x) = \lim_{x\to\infty} e^{\gamma x - (x/\beta)^\alpha} = \infty \qquad \text{for } \alpha < 1$$

Therefore Weibull distributions with parameter $\alpha < 1$ are long-tailed.

Note that the class of long-tailed distributions is bigger than and contains the class of heavy-tailed distributions (Figure 5.5). The Pareto distribution — and in fact any distribution with a power-law tail — is also long-tailed. The lognormal distribution, which is not heavy-tailed, is nevertheless long-tailed [540], like the Weibull distribution. The gamma distribution, in contrast, is not long-tailed. Neither are all the other distributions in the exponential family.

Justification for using this definition rather than the power-law tail definition is based on the following property. It turns out that a sufficient condition for being long-tailed is that the following limit holds for all n:

$$\lim_{x\to\infty} \frac{\Pr(X_1 + X_2 + \cdots + X_n > x)}{\Pr(\max\{X_1, X_2, \ldots, X_n\} > x)} = 1$$

In plain English this means that the sum of n samples from the distribution is large if and only if the biggest sample is large. This is a formal way of saying that the largest samples dominate the lot — the hallmark of heavy-tailed distributions.

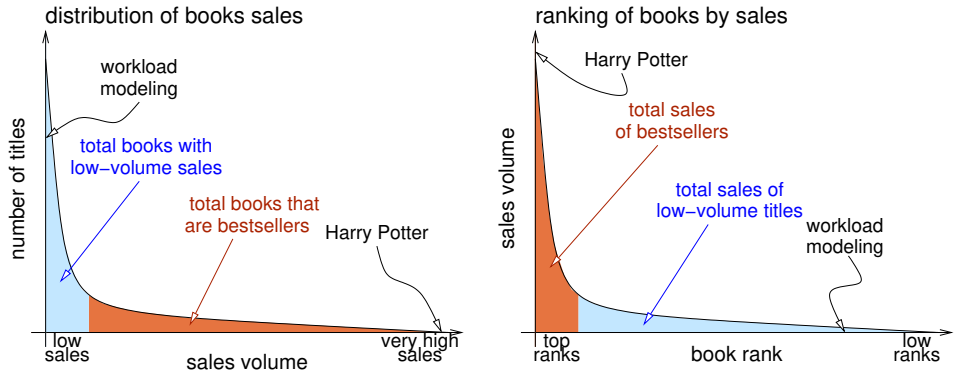
To read more: A comprehensive survey of subexponential distributions is given by Goldie and Klüppelberg [290].

Terminology box: The Meaning of Long Tails

There is some confusion regarding the use of the term "long tail".

First, not all authors make a clear distinction between heavy tails and long tails. In this book, we use "heavy tail" to describe power-law tails and "long tail" to describe subexponential tails, but others sometimes use "long tail" to describe a power law.

Worse, there is a recent trend in the description of modern Internet-based economics to use the term in reverse. In statistics (and in this book), the body of a distribution is the range of values where most of the items are concentrated, whereas the tail is those few items that are rare and have much higher values. In more precise statistical terms, this is actually the "right tail" of the distribution. Using book sales as an example, most books have small or modest sales, whereas Harry Potter has huge sales, and is therefore in the extreme tail of the distribution, as shown in the following schematic (left):



But people involved with the new economy of the networked world do not really look at the distribution as a statistician would. Instead, they look at book rankings, where the books are ranked from the most popular to the least popular. Thus Harry Potter becomes number one, followed by other bestsellers, and the "tail" becomes all those books that sell only few copies because they cater to a niche (such as those few people who are interested in computer workloads). This is illustrated in the right-hand schematic.

In the old economy, distribution to customers was the bottleneck. A book could only sell many copies if it got shelf space in lots of book stores. So only books that could compete for physical shelf space could return their investment, and it did not make economic sense to stock books that only a few people might want. But with the Internet, you can finally find such books even if they are stocked only in some remote warehouse. Therefore someone, such as Amazon, will sell them. And if you sum up the numbers, it turns out that most of Amazon's revenue is from the "long tail" of low-volume, niche books that are relatively unpopular. Of course, this is not unique to books. Chris Anderson, in his book *The Long Tail: Why the Future of Business Is Selling Less of More* [24], brings many more examples, mainly from the sales of music and movies in the entertainment industry.

In effect, this situation therefore amounts to claiming that the distribution does *not* exhibit the important characteristics of heavy-tailed distributions. In particular, instead of having most of the mass concentrated in a very small number of hugely successful items, a large part of the mass is distributed across myriad small items. So the majority of the mass is in the (statistical) body of the distribution, not in its tail.

And if this is not complicated enough, here is the final twist: the new "long-tailed" distributions, where most of the mass is actually not in the tail, may nevertheless have a (statistical) heavy tail. The point is that we are actually looking at two distinct statistical phenomena:

1. Having a heavy tail, defined as a tail that decays as a power law (or a long tail that is subexponential).
2. Exhibiting mass-count disparity, where most of the mass is concentrated in the tail (mass-count disparity is discussed at length later).

Usually these two phenomena occur together, and heavy-tailed distributions exhibit mass-count disparity. But it is possible for a distribution to have a heavy tail without mass-count disparity, such that only a small part of the mass is in the tail. Referring to the left schematic above, this can happen if the tail of the distribution (the part portraying bestsellers) decays according to a power law, but nevertheless most of the mass is in the body of the distribution (the books with low sales volume). This can occur when only the extreme tip of the tail is heavy, say, not the top 1% but only the top 0.001%, and moreover, the tail index is large (close to 2), so that the disparity between the body and the tail is reduced. These conditions are enough to satisfy the mathematical definition, but may not be enough to make a real difference in the distribution of mass in realistic situations where the number of observations is finite.

End Box

5.2 The Importance of Heavy Tails

The definitions of heavy or long tails capture different properties and identify different distributions. However, it is not clear that these properties are the most important in practical terms. In fact, it has been suggested that the qualitative properties discussed next are actually the hallmark of heavy-tailed distributions.

5.2.1 Conditional Expectation

In general, the relative importance of the tail of a distribution can be classified into one of three cases [540]. Consider trying to estimate the length of a process, given that we know that it has already run for a certain time, and that the mean of the distribution of process lengths is m.

- If the distribution of process lengths has a *short* tail, then the more we have waited already, the less additional time we expect to wait. The mean of the tail is smaller than m. For example, this would be the case if the distribution has finite support (i.e., it is zero beyond a certain point).

  In technical terms, the hazard rate, which quantifies the probability that a process has a specific length given that it is not shorter, grows with time.

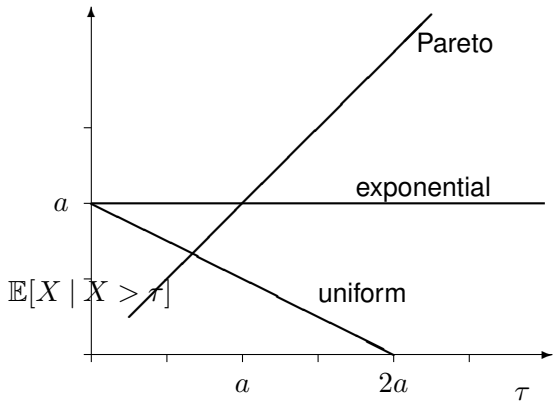


Figure 5.6: *The expected additional mass as a function of how much has already been seen, for sample short, memoryless, and heavy tails.*

- If the distribution is *memoryless*, the expected additional time we need to wait for the process to terminate is independent of how long we have waited already. The mean length of the tail is always the same as the mean length of the entire distribution; in other words, the hazard rate is constant. This is the case for the exponential distribution.

- But if the distribution is heavy-tailed, the additional time we may expect to wait until the process terminates *grows* with the time we have already waited. This counterintuitive effect is called an "expectation paradox" by Mandelbrot [466, p. 342]. The reason is that the mean of the tail is larger than m, the mean of the entire distribution. And in terms of the hazard, it is decreasing with time. An example of this type is the Pareto distribution.

These three cases are illustrated in Figure 5.6. In general, given a random variable X and a threshold value τ, the amount by which X is expected to exceed τ, given that it is larger than τ, is

$$\begin{aligned}\mathbb{E}[X - \tau | X > \tau] &= \int_{\tau}^{\infty} (x - \tau) \frac{f(x)}{\int_{\tau}^{\infty} f(t)\mathrm{d}t} \, \mathrm{d}x \\ &= \frac{\int_{\tau}^{\infty} x f(x) \, \mathrm{d}x}{\int_{\tau}^{\infty} f(t) \, \mathrm{d}t} - \tau\end{aligned}$$

The integral in the denominator re-normalizes $f(x)$ for the range $x > \tau$.

As an example of a short-tailed distribution let us use the uniform distribution on the range $[0, 2a]$. In this range $f(x) = \frac{1}{2a}$; out of this range it is 0. Inserting this into the formula we get

$$\begin{aligned}\mathbb{E}[X-\tau|X>\tau] &= \frac{\int_\tau^{2a} x\frac{1}{2a}\,\mathrm{d}x}{\int_\tau^{2a}\frac{1}{2a}\,\mathrm{d}t} - \tau \\ &= \frac{\frac{1}{2}((2a)^2-\tau^2)}{2a-\tau} - \tau \\ &= \frac{1}{2}(2a-\tau)\end{aligned}$$

and the expected value drops linearly.

The exponential distribution is the only memoryless distribution. Plugging the pdf $f(x) = \frac{1}{a}e^{-x/a}$ into the formula gives

$$\begin{aligned}\mathbb{E}[X-\tau|X>\tau] &= \frac{\int_\tau^{\infty} xe^{-x/a}\,\mathrm{d}x}{\int_\tau^{\infty} e^{-t/a}\,\mathrm{d}t} - \tau \\ &= \frac{ae^{-\tau/a}(\tau+a)}{ae^{-\tau/a}} - \tau \\ &= a\end{aligned}$$

and the expected value remains a regardless of τ.

For a heavy-tailed distribution we use the Pareto distribution with parameter a and the pdf $f(x) = a/x^{a+1}$, defined for $x > 1$. This leads to

$$\begin{aligned}\mathbb{E}[X-\tau|X>\tau] &= \frac{\int_\tau^{\infty}\frac{ax}{x^{a+1}}\,\mathrm{d}x}{\int_\tau^{\infty}\frac{a}{t^{a+1}}\,\mathrm{d}t} - \tau \\ &= \frac{\frac{-1}{(a-1)\tau^{a-1}}}{\frac{-1}{a\tau^a}} - \tau \\ &= \frac{1}{a-1}\tau\end{aligned}$$

which grows linearly with τ.

Connection Box: Failure Models

The issue of conditional expectations has an interesting application in failure models, which address the question of when a system may be expected to fail.

Failure is akin to mortality. A system that fails is dead, and the question is how long it will live before this happens. Common failure models are the Gaussian and the exponential [166]. The Gaussian model specifies lifetimes with a normal distribution. This implies an *increasing* hazard rate, which increases sharply in the higher values of the distribution. A special case is human mortality. This has a very low hazard rate for the first few decades of life, starts to increase at about 40 to 50 years of age, and grows precipitously from about 70 years, with few surviving beyond 90. Other examples include parts that are subject to wear, such as incandescent lamps, dry cell batteries, and new bus motors. Disk drives also seem to have an increasing hazard rate, as the rate of disk replacements correlates with their age [598].

The exponential model of lifetimes implies a *constant* hazard. This is typical of random human errors, such as typing errors, errors in ledgers made by clerks, or forgotten ID

badges (but not on the first day of a workweek, when more are forgotten). Exponential lifetimes have also been found for radar components such as vacuum tubes and resistors, indicating that the failures were random and not due to wear (the paper reporting these findings [166] is from the 1950s).

There are also cases that combine the two models. For example, a full model for bus engine failures includes a small initial phase of random failures due to manufacturing deficiencies (exponential), a dominant second phase of first failures due to wear (Gaussian), and then a phase of repeated failures that is again exponential because it results from a mixture of both original and renovated components.

But there are other types of systems where the mortality rate, or hazard, is *decreasing* with time. For example businesses have a relatively high infant mortality, because new ones often do not succeed [451]. But the longer a business has survived, the higher the probability is that it will stay in business for many more years to come. Thus business lifetimes can be modeled by a Pareto distribution. In fact, this is the context in which the Lomax distribution (the shifted Pareto) was first defined.

End Box

5.2.2 Mass-Count Disparity

An important consequence of heavy-tailed distributions is the mass-count disparity phenomenon: a small number of samples account for the majority of mass, whereas all small samples together only account for negligible mass [154]. Conversely, a typical sample is small, but a typical unit of mass comes from a large sample. Using concrete examples from computers, a typical process is short, but a typical second of CPU activity is part of a long process; a typical file is small, but a typical byte of storage belongs to a large file. This disparity is sometimes referred to as the "mice and elephants" phenomenon. But this metaphor may conjure the image of a bimodal distribution[2], which could be misleading: in most cases, the progression is continuous.

A better characterization is obtained by comparing the "mass" distribution with the "count" distribution. The count distribution is simply the CDF,

$$F_c(x) = \Pr(X < x)$$

because this counts how many items are smaller than a certain size x. The mass distribution weights each item by its size; instead of specifying the probability that an item is smaller than x, it specifies the probability that a unit of mass *belongs to* an item smaller than x. Assuming a pdf $f(x)$, this can be expressed as [154]

$$F_m(x) = \frac{\int_0^x x' f(x')\,\mathrm{d}x'}{\int_0^\infty x' f(x')\,\mathrm{d}x'} \tag{5.2}$$

(this is for positive distributions; in general, the lower bounds of the integrals should be $-\infty$).

[2] A typical mouse weighs about 28 grams, whereas an elephant weighs 3 to 6 tons, depending on whether it is Indian or African. Cats, dogs, and zebras, which fall in between, are missing from this picture.

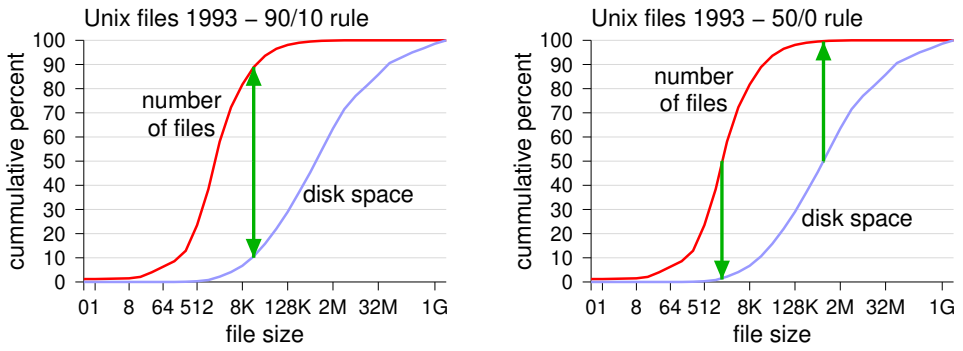


Figure 5.7: *The "count" and "mass" distributions of file sizes, showing the 90/10 and 50/0 rules.*

The disparity between the mass and count distributions can be used to visualize the Pareto principle, also known as the 80/20 rule or the 90/10 rule. An example is shown in Figure 5.7. This shows data from a 1993 survey of 12 million Unix files [361]. The left graph shows that the data is very close to the 90/10 rule: 10% of the files are big files, and account for a full 90% of the disk space. At the same time, the remaining 90% of the files are small, and together account for only 10% of the total disk space. The boundary between big and small in this case is 16 KB. Less extreme datasets would be closer to a ratio of 80/20.

An even more dramatic demonstration of mass-count disparity is the 50/0 rule [238]. As shown on the right-hand side of Figure 5.7, a full *half* of the files are so small, less than 2 KB each, that together they account for a negligible fraction of the disk space. At the other extreme, half of the disk space is occupied by a very small fraction of large files (1 MB or more).

Implications

The implication of mass-count disparity is that it may be beneficial to focus on the few large items, because they have the largest impact and in fact may dominate the outcome. There are many examples of putting this principle to use.

An early example comes from load balancing. Process runtimes on Unix workstations have a Pareto tail, meaning that a small number of processes are extremely long. Migrating such processes from overloaded workstations to underloaded ones therefore will have a long-term beneficial effect on the load conditions. Moreover, it is easy to identify the long processes due to the conditional expectation of heavy tails: they are the processes that have run the longest so far [320, 435]. Moving a randomly chosen process will only have a minor effect, because such a process will probably terminate shortly after being moved.

Workloads that display such behavior have profound implications on system modeling. In particular, if the distribution of job sizes is heavy-tailed, there will be occasionally

a job of such magnitude that it dwarfs all the other jobs put together. In effect, this means that the system cannot be considered to be in a steady state, but rather is constantly in a transient state [156, 157].

Another example comes from accounting for Internet usage. The elements of Internet usage are flows (e.g., the servicing of a request from a web server). The sizes of flows are heavy-tailed. Thus keeping track of all flows is extremely arduous: there are huge numbers of very small flows, which together do not account for much resource usage, and few large ones, which use most of the bandwidth. It has therefore been proposed that it is best to keep track of only the large flows, and simply ignore all the small ones [215]. Likewise, traffic engineering may benefit from focusing on the few large flows that dominate the bandwidth consumption [92]. Even routing can benefit from offloading the large flows to a fast hardware forwarding device, thereby freeing capacity on the slower software-based router [591].

Several interesting examples relate to caching. In memory reference streams some data blocks are much more popular than others — so much so, that sampling a block at random will produce a block that is seldom used, but sampling a reference at random will most probably identify a block that is highly used [216, 217, 218]. This can be used to selectively insert only highly used blocks into the cache, without any need to maintain historical information about access patterns to each block.

In the context of caching web pages, it has been pointed out that caching to reduce the number of requests reaching a server is different from caching to reduce the number of bytes requested from the server [37]. To reduce the number of requests, the most popular items should be cached. But if they are small, this will only cause a marginal reduction in the number of bytes served. To reduce the number of bytes, it is better to cache the large items that use most of the bandwidth. Depending on whether the bandwidth or the per-request processing is the bottleneck, either policy may be better.

Another subtle use of mass-count disparity comes from task assignment in the context of server farms [319]. Consider a situation in which two servers are available to serve a stream of incoming requests, as is done in many large websites (typically, many more than two are used). Each new request has to be assigned to a server for handling. A good policy turns out to be based on size: requests for small files are handled by one server, whereas requests for large files are handled by another. This prevents situations in which a small request may be held up waiting for a large one to complete. But the really interesting observation is that the division of labor should not necessarily be done so as to balance the loads [317, 599]. Instead, it is better to assign a smaller fraction of the load to the server handling the small requests. Due to mass-count disparity, we then have a situation in which the vast majority of requests benefit from a relatively underloaded server, while only a small fraction of requests suffer from loaded conditions.

The use of mass-count disparity is not limited to computer systems. In *The 80/20 Principle: The Secret to Achieving More with Less*, Richard Koch explains how you can get twice as much done in 2/5 the time by getting rid of the ineffective 80% of what you do, and doubling the effective 20% [408].

Connection Box: Rare Events in Trading and Insurance

Heavy tails exist and have serious implications in areas outside of computer science, most notably in finance.

A specific field where this is the case — apparently without being widely recognized — is stock market trading. Traders may succeed for years, making huge profits by betting that the market will continue to behave in the same way that it did before. But then, when something that has never occurred before suddenly happens, they can suffer losses that wipe them out completely within one day. The description of such rare events is the subject of Taleb's book *Fooled by Randomness* [672].

Another area where heavy tails are extremely important is insurance. Insurance companies depend on their ability to calculate the odds against rare events. A large-scale destructive event, such as an earthquake or hurricane, occurs only rarely. But when it does, it can cause huge losses. The long-term survival of the insurance company depends on its ability to balance the odds, collecting enough revenues during the good times to survive the losses of the rare events. Interestingly, insurance is an area where highly improved modeling capabilities may ultimately undermine the whole industry: with perfect predictions, insurers would not be willing to insure anyone or anything who is about to suffer destructive losses, so being able to buy insurance would be a sign that you do not need it.

End Box

Metrics for Mass-Count Disparity

The impact of mass-count disparity is so great that it has been suggested that it is actually the defining feature of heavy-tailed distributions [317]. It does not matter whether the tail indeed decays according to a power law and how many orders of magnitude are spanned. The important thing is that the data displays something similar to the 90/10 or 50/0 rules: a very small fraction of the samples are responsible for a sizable fraction of the total mass.

Formalizing this idea, we can suggest several numerical measures that indicate the degree to which a distribution exhibits mass-count disparity [238]. They are based on comparing the CDFs of the items and the mass. In effect, they simply formalize the notions presented in Figure 5.7. Although the names for these metrics are new, the metrics themselves have been used by others (e.g., by Irlam in his descriptions of the Unix files data from 1993 [361]).

The simplest metric is the *joint ratio* (also called the *crossover* [92]). This is a direct generalization of the 90/10 rule and the 80/20 rule. The 90/10 rule, for example, says two things at once: that 10% of the items account for a full 90% of the mass, and also that 90% of the items account for only 10% of the mass. This is more extreme than the 80/20 rule, which says that 20% of the items account for 80% of the mass and 80% of the items account for 20% of the mass. The generalization is to find the percentage p such that $p\%$ of the items account for $100 - p\%$ of the mass, and $100 - p\%$ of the items account for $p\%$ of the mass. The smaller p is, the stronger the mass-count disparity (and the heavier the tail of the distribution).

In mathematical notation, this idea can be expressed as follows. Denote the count CDF by $F_c(x)$ and the mass CDF by $F_m(x)$. Because CDFs are nondecreasing, the

complement of a CDF is nonincreasing. Therefore there is a unique x that satisfies the condition[3]

$$F_c(x) = 1 - F_m(x)$$

Given this x, compute $p = 100F_m(x)$. The joint ratio is then $p/(100 - p)$.

The 50/0 rule is generalized by two metrics. In practice, the 0 is not really 0; the metrics quantify how close to 0 we get. The first metric is $N_{1/2}$ (pronounced N-half). This quantifies the percentage of items from the tail needed to account for half of the mass:

$$N_{1/2} = 100(1 - F_c(x)) \quad \text{such that} \quad F_m(x) = 0.5$$

The second is $W_{1/2}$, which quantifies the percentage of the total mass due to the bottom half of the items:

$$W_{1/2} = 100F_m(x) \quad \text{such that} \quad F_c(x) = 0.5$$

These expressions can also be written more directly as $N_{1/2} = 100(1 - F_c(F_m^{-1}(0.5)))$ and $W_{1/2} = 100F_m(F_c^{-1}(0.5))$, where F_c^{-1} and F_m^{-1} are the inverse of F_c and F_m, respectively.

Note that all three metrics measure the vertical distance between the two distributions. This is because the vertical distance best characterizes the mass-count disparity. But it may also be interesting to know how much larger the tail items are. This can be measured by the *median-median distance*, that is, the distance between the medians of the two distributions. The farther apart they are, the heavier the tail of the distribution. Because absolute values depend on the units used, it makes sense to express this distance as a ratio (or take the log of the ratio to express the distance as the number of orders of magnitude that are spanned).

These metrics are illustrated for several datasets in Figures 5.8 and 5.9. The top two plots in Figure 5.8 show data about files from Unix file systems. The top-left plot is the 1993 survey mentioned earlier, with data about some 12 million files from more than a thousand file systems. The right one is a departmental file system with more than 18 million files sampled in June 2005. Despite the time gap, these two datasets are amazingly similar, both in terms of their general shapes and in terms of the specific values assumed by our metrics.

On the bottom is data for Unix process runtimes. The left plot is a dataset from 1994 used by Harchol-Balter and Downey, with about 185,000 processes that arrived over eight hours [320]. The right-hand dataset contains more than 450,000 Unix processes from a departmental server, covering about a month in 2005. This dataset is the most extreme, especially in terms of the median-median distance.

These examples are all related to the sizes of workload items. But mass-count disparity can also be applied to popularity — most items are not very popular, but a few are very popular and enjoy most of the references. This can be visualized as follows. Rank the items in order of popularity. $F_c(x)$ is then the probability that a randomly chosen item receives up to x references. Now look at references, and rank them according to

[3]The only problem may be if the distribution is modal, and this x value is a mode.

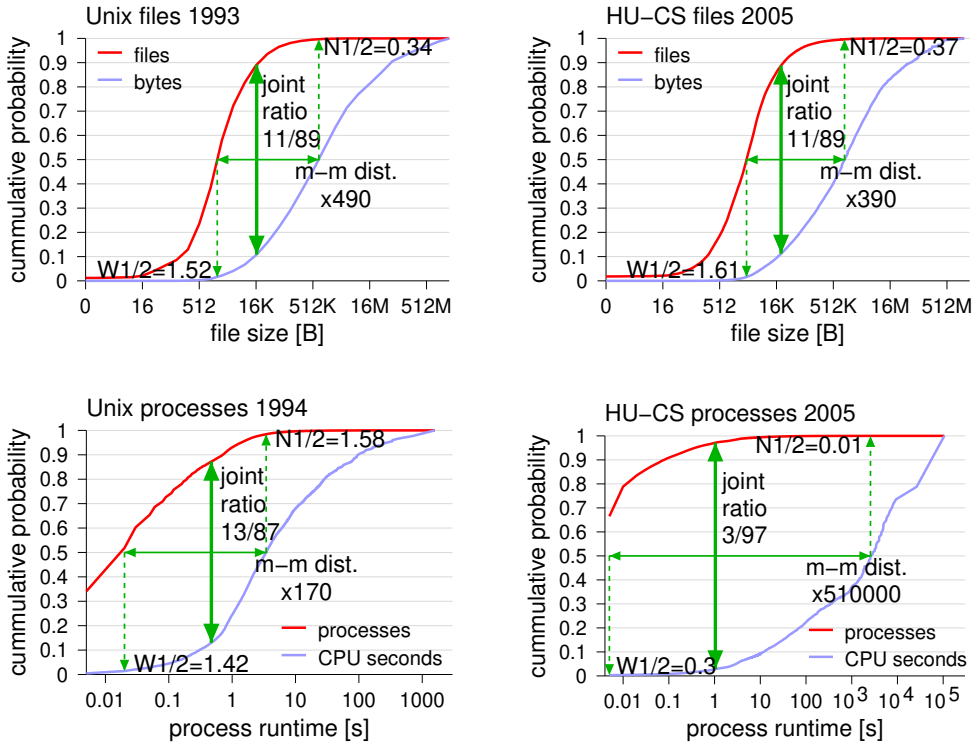


Figure 5.8: *Examples of the joint ratio, $N_{1/2}$, $W_{1/2}$, and median-median distance metrics, for size-related distributions.*

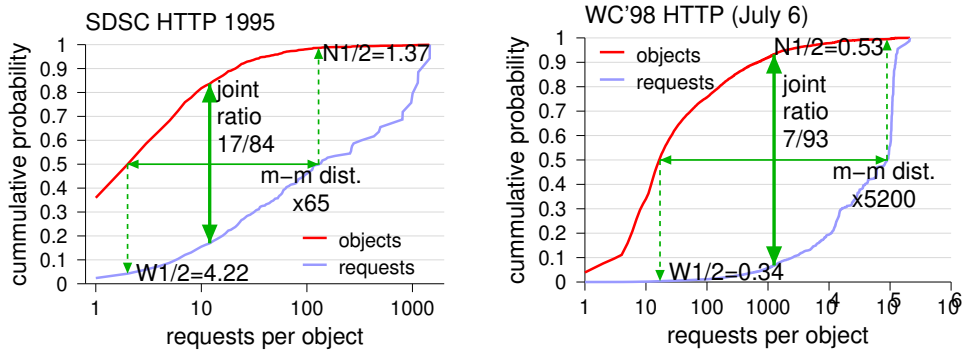


Figure 5.9: *Examples of mass-count disparity metrics for popularity data.*

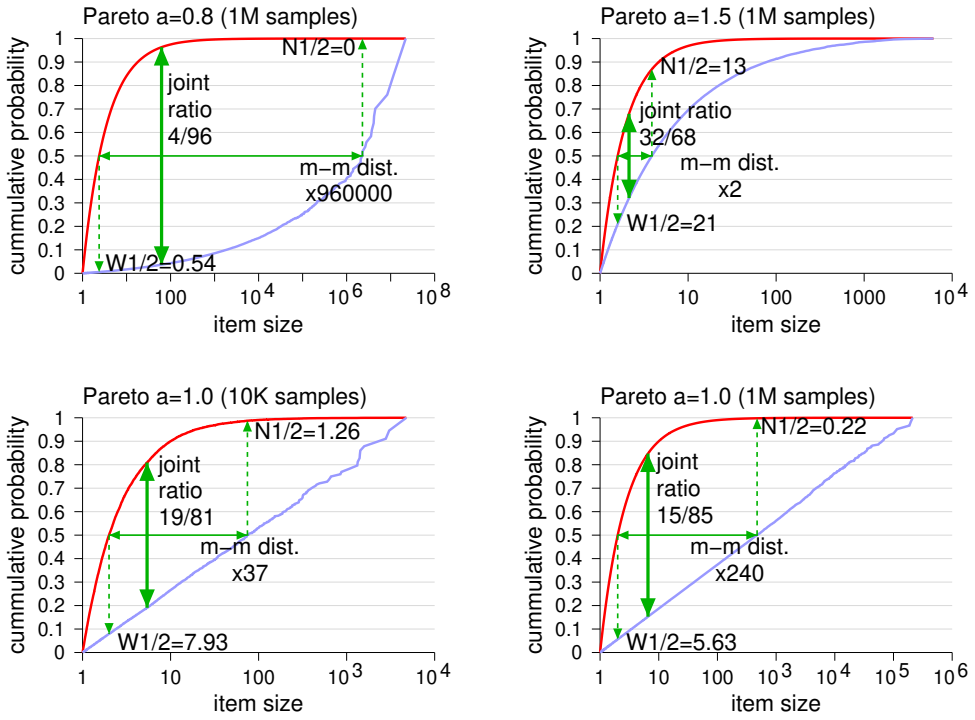


Figure 5.10: *Mass-count disparity plots and metrics for the heavy-tailed Pareto distribution.*

the popularity of the item to which they refer; thus first there will be those references to items that are accessed only once, then references to items with some repetition, and at the end all those references to the most popular item. $F_m(x)$ is the probability that a randomly chosen reference references an item that is accessed up to x times.

Figure 5.9 shows examples related to popularity, and, specifically, how many times a web page was downloaded in a single day. The dataset shown on the left is from an HTTP log from SDSC from 1995, and contains about 28,000 requests. The one on the right is from the France'98 World Cup website. The World Cup data is much more extreme than the SDSC data, indicating that the SDSC requests were more evenly spread, whereas the traffic to the World Cup site was much more focused on a select set of documents.

The above examples show mass-count plots for empirical data, and extract the metrics from these plots. But given a mathematical expression for a distribution, it may also be possible to compute the metrics analytically. For example, the Pareto count distribution is

$$F_c(x) = 1 - \left(\frac{k}{x}\right)^a$$

where k is the minimal value possible (that is, the distribution is defined for $x \geq k$), and a is the tail index. The mass distribution is then

$$F_m(x) = \frac{\int_0^x x' \frac{ak^a}{x'^{a+1}}\,dx'}{\int_0^\infty x' \frac{ak^a}{x'^{a+1}}\,dx'} = 1 - \left(\frac{k}{x}\right)^{a-1}$$

The integrals only converge for the case when $a > 1$; if a is smaller, the tail is so heavy that the mean is infinite. This is reflected in the shape of the plots (Figure 5.10). When a is small, a significant part of the mass occurs in the few top samples, and we get very high mass-count disparity. When $a = 1$, the mass distribution is a straight line. In both cases, the shape of the graphs, and thus the metric values, actually depend on the number of samples observed. For example, for $a = 1$ the slope changes to reflect the number of samples, and correlates with how far into the tail we see (and if we are "lucky" to see a rare sample from deep in the tail, we will get a broken line).

When a is larger (the case described by the equations) the metric values are rather moderate — and in fact, indicate significantly lower mass-count disparity than observed in the above examples of real data. A hint regarding a possible explanation for this is given in Figure 5.8. Looking at the plots for file sizes and 1994 Unix processes, we see the expected shape of the Pareto plots on the right of each plot, but a different behavior on the left. This is because the data are actually not well modeled by a Pareto distribution: only the *tail* is Pareto. The full dataset is a mixture of a Pareto tail with a body that contains many more small items than in a Pareto distribution.

The Lorenz Curve and Gini Coefficient

Another way to display the relationship between the count distribution and the mass distribution is the Lorenz curve, which has its roots in measuring inequality in the distribution of wealth [452]. This is essentially a P-P plot of these two distributions. Given the two CDFs, a P-P plot is constructed by pairing percentiles that correspond to the same value (as opposed to Q-Q plots, which pair values that correspond to the same percentile). Thus in our case, for each value x we will find

$$p_c = F_c(x) \quad \text{and} \quad p_m = F_m(x)$$

and then use this to express the mass percentile as a function of the count percentile:

$$p_m = F_m(x) = F_m(\, F_c^{-1}(p_c)\,)$$

where F_c^{-1} is the inverse of F_c.

An example using the Unix 1993 file sizes and 1994 process runtimes is shown in Figure 5.11. Focusing on the process runtime data, we can say that about 90% of the processes consumed roughly 20% of the CPU time. In general, if mass is distributed equitably the Lorenz curve would simply be the diagonal, because $p\%$ of the items would also be responsible for $p\%$ of the mass. The greater the mass-count disparity, the farther the curve is from the diagonal.

Based on the Lorenz curve, the Gini coefficient reduces the degree of inequality to a single number. The calculation is simply the ratio of two areas: the area between the

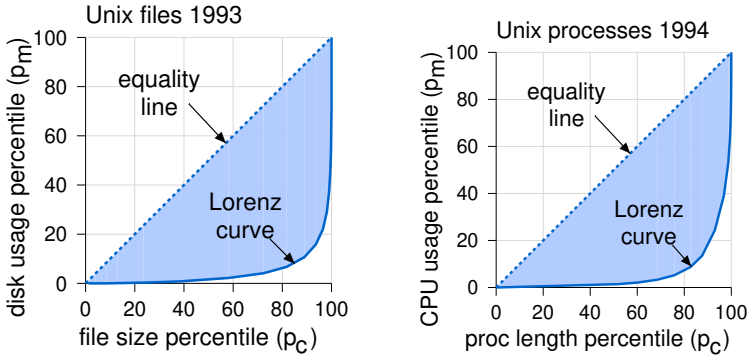


Figure 5.11: *The Gini coefficient corresponds to the shaded area above the Lorenz curve.*

equality line and the Lorenz curve, and all the area below the equality line. Denoting the Lorenz curve by $L(x)$, this is

$$G = 2 \int_0^1 (x - L(x))\, \mathrm{d}x$$

(the factor of 2 reflects the area under the equality line, which is a triangle with area $\frac{1}{2}$). The closer the Lorenz curve is to the line of equality, the smaller the area between them; in this case the Gini coefficient tends to 0. But when a small fraction of the items account for most of the mass, the Lorenz curve is close to the axes, and the Gini coefficient is close to 1. This is the case illustrated in Figure 5.11. Specifically, the Gini coefficient for file sizes data is 0.913, and for the process runtime data it is 0.882. (Inequality of wealth is typically lower, with Gini coefficients in the range 0.3–0.4.)

The Gini coefficient is popular, especially in economics, because of its frugality. But Lorenz curves with different shapes, for example closer to the X axis or rather to the Y axis, may lead to the same Gini coefficient. Thus the metrics presented previously do indeed provide more information. In the context of the Lorenz curve, the $W_{1/2}$ metric measures the distance from the middle of the X axis, $N_{1/2}$ measures the distance from the middle of the Y axis, and the joint ratio measures the distance along the diagonal from the bottom-right corner.

5.3 Testing for Heavy Tails

Testing for heavy tails depends on the definition you use. For example, if the existence of significant mass-count disparity is used as the definition, a simple visual inspection may suffice: plotting CDFs as in Figure 5.7 allows for easy verification whether the data follows the 80/20 rule, the 90/10 rule, or the 50/0 rule. This can be strengthened by computing the metrics of joint ratio, $N_{1/2}$, and $W_{1/2}$.

The most common definition, however, is the existence of a power-law tail. We

therefore focus on tests for this feature. Note, however, that these tests assume a perfect power-law tail, and may not work well if the tail is truncated.

LLCD Plots

Testing for a power-law tail can be done using log-log complementary distribution plots. This is based on the definition of heavy tails as given in Equation (5.1). Taking the log from both sides we observe that

$$\log \bar{F}(x) = \log x^{-a} = -a \log x \tag{5.3}$$

So plotting $\log \bar{F}(x)$ (the log of the fraction of observations larger than x) as a function of $\log x$ should lead to a straight line with slope $-a$. As we are plotting $\bar{F}(x)$ in logarithmic axes, the name "log-log complementary distribution plot" (LLCD) is a natural one[4].

Practice Box: Drawing an LLCD

When drawing distributions from large datasets, it is common practice to use binning: partition the data into bins of equal size, and draw a collective data point for each bin to reduce the size of the dataset. But when drawing an LLCD we go down to the very low probabilities associated with samples from the tail. It is therefore necessary to include all the available samples from the tail individually. However, it is still possible to use binning for the body of the distribution.

The LLCD actually shows the empirical survival function. Strictly speaking, the estimate of the probability of seeing a value higher than a certain sample x_i is the relative number of samples (out of n) larger than it:

$$\Pr(X > x_i) = \frac{|\{x_j \mid x_j > x_i\}|}{n}$$

However, with this definition the probability of being higher than the largest sample is zero, so we are essentially not going to use the largest sample, which may be the most prominent representative of the tail.

This problem can be eliminated by increasing all probabilities by $\frac{1}{2n}$ and replacing the first one by $1 - \frac{1}{2n}$. To see why such an approach is reasonable, consider the case when we have only a single sample x_1. Using the original approach, the postulated survival function is a step function: we would say that there is a probability of 1 of being bigger than any x that satisfies $x < x_1$, and a probability of 0 of being bigger than any x that satisfies $x > x_1$. In effect, we turn x_1 into an upper bound on the distribution. The second approach essentially regards the lone sample as the median, and postulates that there is a probability of $\frac{1}{2}$ of being either above or below it. Assuming we know nothing about the distribution, this is a much more reasonable approach.

Another issue regarding the drawing of LLCD plots is their aspect ratio. Because we are drawing a line and looking at its slope, the scales of both axes should be the same. This way a line with slope –1 will indeed be seen to have a slope of –1.

End Box

[4]Note, however, that generally the term "survival function", rather than "complementary distribution", is preferred in this book.

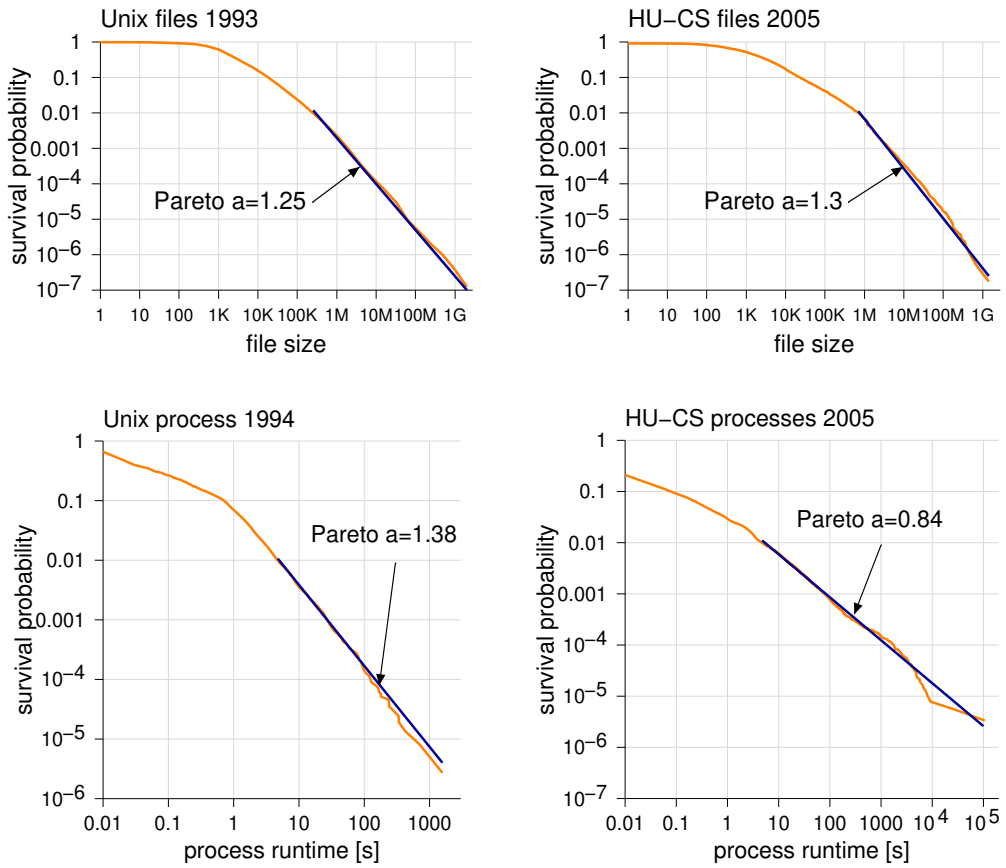


Figure 5.12: *LLCD plots for several datasets (the same ones as in Figure 5.8), and fitted straight lines for the top 1%.*

A number of examples of LLCD plots are shown in Figure 5.12. Recall that a straight line indicates that a power law is present, and the slope of the line provides an estimate of the tail index a. Note also that it is not necessary for the entire distribution to fall on a straight line. This happens only for the Pareto distribution, which is characterized by a power law across the entire range of values. But it is also possible to have a distribution in which the body is described by some other function, and only the tail is governed by a power law. In this case the transition from the body to the tail can be identified by the point at which the LLCD begins to look like a straight line. It is customary to require that the straight line segment covers several orders of magnitude.

In these examples we arbitrarily define the top 1% of the distribution to be the tail. A linear regression (in log-space) then finds the best linear representation and its slope. The results are that power-law tails appear to be present in all of them.

For comparison, LLCD plots of several statistical distributions are shown in Figure 5.13. The exponential distribution has a mean of $\theta = 100$. Obviously its tail decays

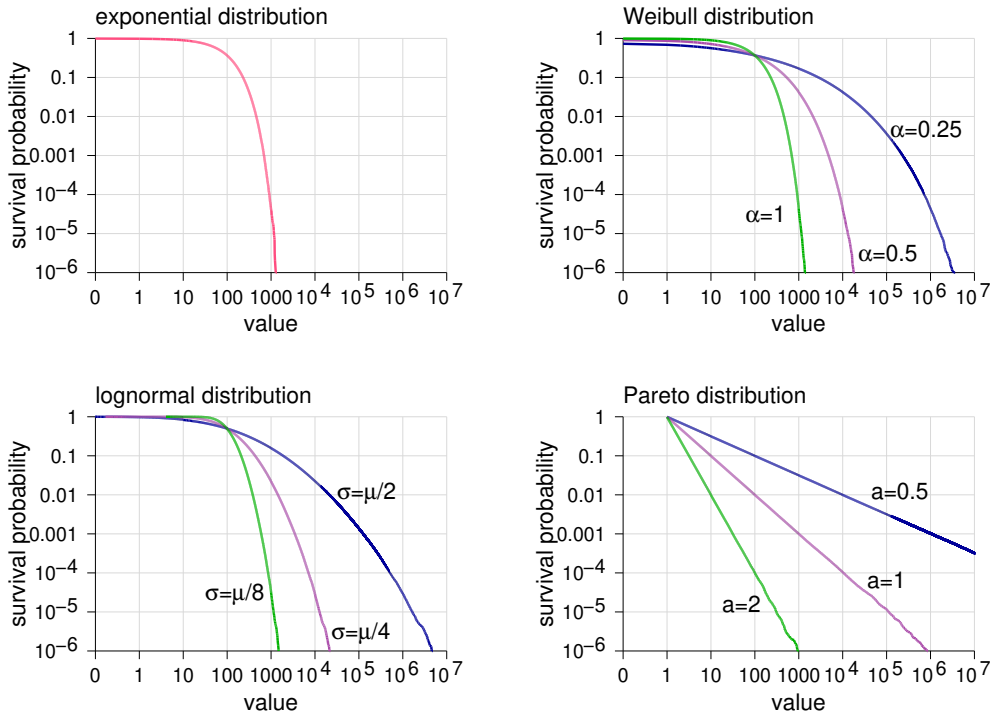


Figure 5.13: *LLCD plots for distributions with different tail characteristics.*

rapidly (in fact, exponentially) for higher values. Mathematically, the expression is as follows. We know that $\bar{F}(x) = e^{-x/\theta}$. Therefore $\log \bar{F}(x) = -x/\theta$. But we want this as a function of $\log x$, not as a function of x. We therefore write

$$\log \bar{F}(x) = -\frac{e^{\log x}}{\theta}$$

The plot is then the same as that of an exponential going to ∞, but with a $-$ sign, so it goes down instead of up.

The Weibull distribution with $\alpha < 1$ is long-tailed, and indeed in such cases the tail decays more slowly, especially for small αs (for $\alpha = 1$ it is actually equivalent to the exponential). However, the slope of the LLCD plots still becomes steeper for higher values of x. In all these examples, the scale parameter is $\beta = 100$.

The lognormal distribution is also long-tailed. Again, the decay of the tail depends on the parameter values. In the examples shown the location parameter is $\mu = 4.6$, which leads to a mode at 100 (because $4.6 \approx \ln 100$). The tail behavior then depends on the relative size of σ. For smaller σ, the distribution becomes less disperse, and consequently, the tail becomes less pronounced. For large σ, the tail becomes heavier, and may look rather straight in the LLCD. To explain this, consider the logarithm of the distribution's pdf (given in Equation (3.29)) as a function of $\log x$ (of course, we should actually look at the survival function, but it does not have a closed form) [497]. This is

$$\begin{aligned}\ln f(x) &= -\ln\left(x\sigma\sqrt{2\pi}\right) - \frac{(\ln x - \mu)^2}{2\sigma^2} \\ &= \frac{-1}{2\sigma^2}(\ln x)^2 + \left(\frac{\mu}{\sigma^2} - 1\right)\ln x - \frac{\mu^2}{2\sigma^2} - \ln\sigma\sqrt{2\pi}\end{aligned}$$

If σ^2 is large the quadratic term is small, and the linear one dominates.

The Pareto distribution is the only one with an LLCD that is a completely straight line. This reflects its power-law form, and of course, the slope of this line corresponds to the tail index.

Curvature

While the Pareto distribution has an LLCD that is completely straight, this is often not observed in real data. In many cases, the whole distribution is not well modeled as Pareto — only the tail is a Pareto tail. Thus the LLCD is not completely straight, but just has a straight segment at the end.

Note, however, that the Weibull and lognormal distributions (with appropriate parameter values) can also have portions of the tail that look approximately linear across two or three orders of magnitude. In principle the deviation from a straight line can be found by continuing to higher values and lower probabilities, but in practice this is often impossible because sufficient data is not available. It is therefore desirable to find a test that can distinguish between a Pareto tail and a lognormal tail that look very similar.

Such a test has been proposed by Downey, based on a detailed measurement of the curvature of the LLCD [189]. It is done in two steps. First, the curvature is estimated, and then a check is made whether such a curvature may reasonably occur for a Pareto distribution. If the answer is no, then another model such as a lognormal is preferred.

Measuring the curvature of the LLCD is done by computing a numerical first derivative — the ratios of the vertical and horizontal differences for successive points. If the original plot is a straight line, these ratios should all be very similar to each other, so plotting them should lead to a line with a slope near 0. Fitting a straight line using linear regression provides the actual slope. Repeating this on multiple sets of samples from a synthetic Pareto distribution with the same putative tail index shows whether this value may have occurred by chance (another example of validation using the bootstrap method [205, 206]).

A somewhat simpler procedure has been proposed by Eeckhout [196]. Its point of departure is the observation that fitting a straight line to the right end of an LLCD plot (or alternatively, to the left end of a Zipf size-rank plot) depends on the definition of the starting point, which separates the body from the tail. Thus, using a sequence of different starting points produces a sequence of estimates for the tail index. If the distribution is indeed heavy-tailed, all these estimates should be more or less the same. But if it is, for example, actually a lognormal distribution, the estimates should increase monotonically.

An example is shown in Figure 5.14. This intentionally starts off by defining a full 20% of the data to be the tail, in order to enable a wider view of the distribution before focusing on the tail. Note that the results for the Unix 1994 processes are nevertheless

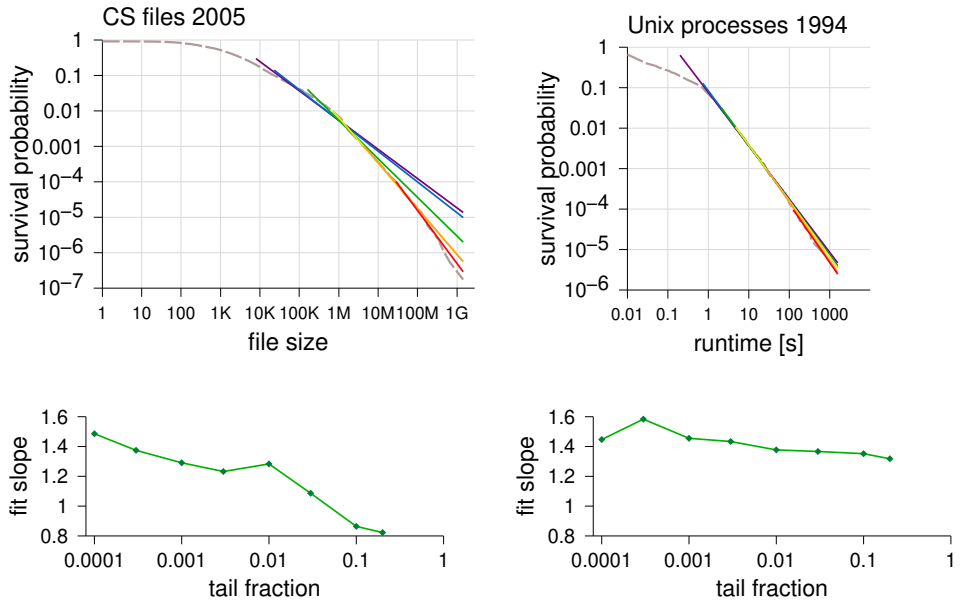


Figure 5.14: *Slope of the fitted line as a function of the definition of where the tail starts.*

all consistent, giving a strong indication that this is indeed a Pareto tail. However, for the CS file-size data the initial slopes are too shallow and do not reflect the tail; only focusing on the last 1% of the data leads to more or less stable results (note that in the graphs showing the fit slope as a function of how much of the data is considered to be the tail, having less data belonging to the tail corresponds to moving left in the graph). In fact, this example also shows that the technique can be used to judge where the tail starts.

Aggregation

The LLCD technique can be further improved by aggregating successive observations. This essentially means looking at the data at a coarser resolution, instead of looking at individual samples:

$$X_i^{(m)} = \sum_{j=(i-1)m+1}^{i\,m} X_j \tag{5.4}$$

Distributions for which such aggregated random variables have essentially the same distribution as the original are called stable distributions. The normal distribution is the only stable distribution with finite variance. This is a direct corollary of the central limit theorem, which states that if we sum independent random variables from *any* distribution with finite variance, the sums will be normally distributed.

But heavy-tailed distributions (according to definition of Equation (5.1)) have an infinite variance. Thus the central limit theorem does not apply, and the aggregated random

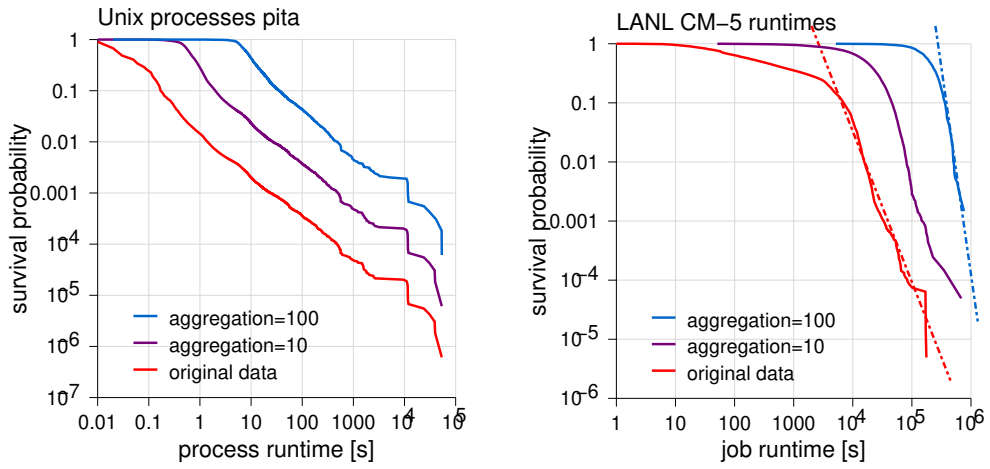


Figure 5.15: *LLCD plots with aggregation of data. The pita dataset on the left is heavy-tailed, whereas the LANL dataset on the right is not.*

variables do not have a normal distribution. Rather, they have an α-stable distribution, which has a heavy right tail with the same tail index as the original (recall Figure 5.4). This can be verified by creating LLCD plots of the aggregated samples, and checking that they too are straight lines with the same slope as the original [158, 155]. If the distribution is not heavy-tailed, the aggregated samples will tend to be normally distributed (the more so as the level of aggregation increases), and the slopes of the LLCD plots will increase with the level of aggregation.

An example is given in Figure 5.15, which compares two datasets. The Unix processes from pita seem to have a power-law tail: when aggregated by a factor of 10 or 100 we get the same straight line as for the original data. The job runtimes on the LANL CM-5, in contrast, are not heavy-tailed. Although the tail in the original data leads to a reasonably straight line, its slope is rather high, and it becomes steeper with aggregation.

Using the requirement that LLCDs of aggregated samples be straight and parallel, Crovella and Taqqu suggest a methodology for identifying exactly that part of the tail that behaves like a heavy tail [158] (described later). Interestingly, when this is applied to data from various long-tailed distributions, the results are inconclusive. In particular, large parts of the tail of a lognormal distribution look like a heavy tail, but not all of the tails of Pareto and α-stable distributions pass the test.

Deviations From a Zipf Distribution

Until now the discussion has concerned the problem of characterizing the tail of a distribution in situations in which the large samples from the tail are extremely rare and therefore we do not have enough data. For example, this happens when we want to characterize the distribution of the largest files or the longest processes. But when we want

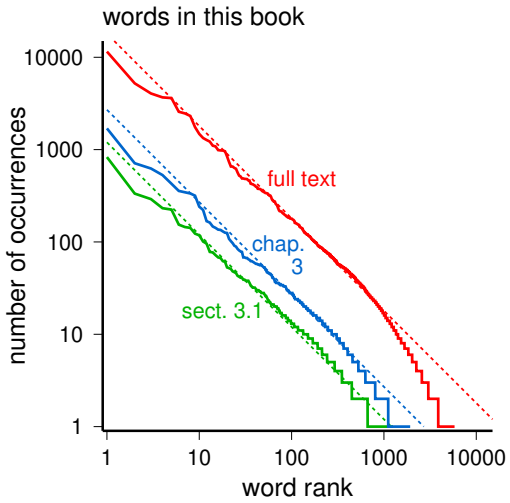


Figure 5.16: *When a popularity distribution has a heavy tail, taking only a small number of samples makes it look like a Zipf distribution.*

to characterize a popularity distribution, the situation is reversed, and the tail is easily characterized.

In the context of popularity distributions, a heavy tail leads to a Zipf distribution. This is typically rendered as a rank-size plot, in which items are sorted from the most popular to the least popular, and the number of occurrences of each item is plotted as a function of its rank in log-log axes (Figure 5.16). In such a plot, the tail of the popularity distribution is represented by the high counts of the top-ranked items at the top left of the graph. The body of the distribution is represented by the low-ranking items at the bottom right.

An interesting effect occurs as the number of samples observed is increased. Initially, when we only have a relatively small number of samples, we typically only see the most popular items. Because the popularity distribution has a heavy tail, these items will be Zipf distributed (the correspondence between the Zipf distribution and the Pareto distribution was established on page 133). Assume a true Zipf distribution with a slope of -1 in log-log axes. This implies that the axes are intersected at the same value: if the top-ranked item is seen k times, the top rank is also k (i.e., we see a total of k different items). As more and more samples are collected, a Zipf distribution implies that more and more distinct items must be observed.

But in many cases the underlying population is actually finite. For example, when looking at word frequencies, the underlying population is the vocabulary of the author. Thus the distribution must be truncated at some point. When more and more samples are observed, we will see a deviation from the straight line of the Zipf distribution, and the greater the number of samples, the greater the deviation [690]. This is shown in Figure 5.16 for the distribution of words in this book. The heavy-tail statistics govern the few

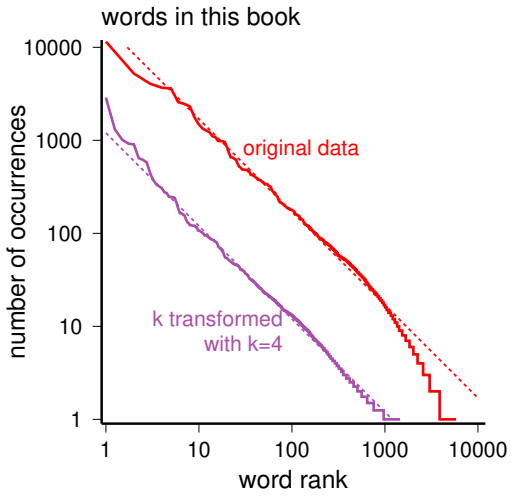


Figure 5.17: *Using a k-transform leads to an apparent Zipf distribution. The dashed lines used for reference have a slope of -1.*

hundred most popular words. The 8,362 words of Section 3.1, which represent only 1,254 distinct words, are still nearly Zipf distributed. The 17,922 words of Chapter 3 (1,924 distinct words) begin to show a slight deviation. But when considering all 134,987 words in the book, comprising a vocabulary of 5,819 distinct words, the deviation from the Zipf distribution is unmistakable.

The above discussion assumes that the distribution of popularity is fixed. But the effect of deviations is much stronger if the popularity of different items changes with time, as it does for news stories or movies [382, 675]. In this case the distribution of popularity on an average day may be Zipfian, but taken over a long period the popularities of all the items do not conform to a Zipf distribution, because the popular movies in one week are replaced by other movies a month later, rather than continuing to accrue additional viewings. The same may happen in a professional text, where the unique vocabulary of one chapter may be different from that of another.

Tang et al. have suggested using the k-transform to uncover the underlying Zipf nature in such situations [675] (Figure 5.17). The idea is that, instead of considering the items in isolation, one creates equivalence classes. For example, all the most popular movies in the different weeks are one class, all the second most popular ones are a second class, and so on. Technically this is achieved by assuming that the different weekly distributions are perfectly merged with each other. Thus if there are k weekly datasets, the first k elements will be the k top movies from the different weeks, and so on. We therefore replace the elements by their group using the following transformation:

$$x' = \frac{x + k - 1}{k}$$
$$y' = \frac{y + k - 1}{k}$$

This is highly effective in producing a straight line when the rank-size plot is shown in log-log axes, as illustrated in Figure 5.17. The slope can then be used to parameterize a model [675].

5.4 Modeling Heavy Tails

By definition, a heavy tail is Pareto distributed. The way to model it is therefore to find the appropriate parameters for a Pareto distribution that describes the tail data. This often implies that the data is partitioned into two parts, and the body of the distribution is modeled separately from its tail [473].

One should nevertheless bear in mind the possibility of using other distributions that lead to effects such as mass-count disparity, even if they do not have a tail that decays according to a power law. This is especially true when a single distribution provides a reasonable fit to the complete dataset. In fact, this can also be done using variants of the Pareto distribution, based on shifting and truncation.

5.4.1 Estimating the Parameters of a Power-Law Tail

The Pareto distribution has two parameters: k and a. k is a location parameter, and signifies where the distribution starts. It is therefore easy to estimate using the smallest sample seen. When we are modeling the tail of the distribution separately from its body, k is set to the boundary point where the tail begins.

It is harder to estimate the value of the shape parameter a (also called the tail index of the distribution). There are two major problems. First, by definition we do not have many samples to work with, because we are only dealing with the tail of the whole distribution. Second, the most important samples are the biggest ones, which are, of course, the most rare. As a result it is often the case that it is hard to find exact values for a, and that different techniques yield somewhat different results.

Graphical Methods

The simplest approach for estimating a is graphical, based on drawing the histogram of the distribution on log-log axes. If the distribution follows a power law, it will yield a straight line, and performing a linear regression (in log-space!) will find its slope and provide an estimate for $-(a+1)$. The problem with this approach is that the high values only appear once (or a small number of times) each, leading to much noise and biased estimates [291]. It is therefore necessary to use logarithmic binning when drawing the histogram [514]. Figure 5.18 demonstrates the improvement when using logarithmic binning. Note that counts have to be normalized by bin width to retain the correct shape of the histogram.

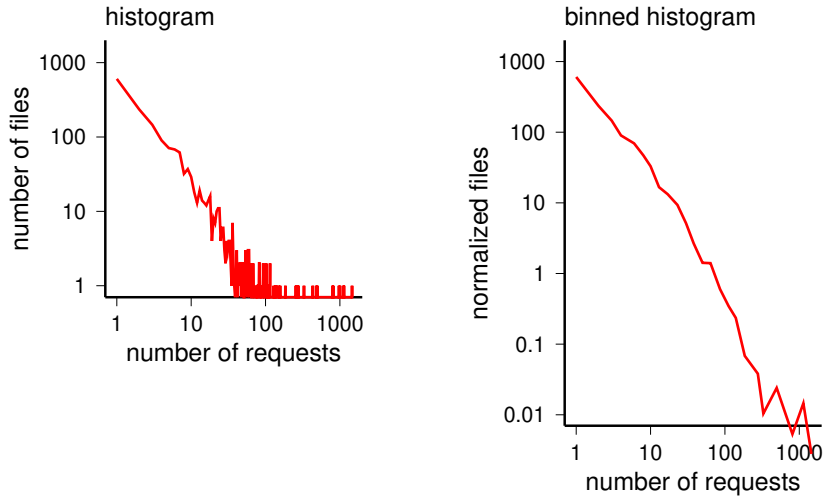


Figure 5.18: *Raw and logarithmically binned histogram of heavy-tailed data, in this case the popularity of web pages from SDSC.*

A related technique is based on the LLCD plot. As shown earlier, if the tail indeed follows a power law, the LLCD plot ends in a straight line. This time the slope of the line is equal to $-a$, so finding a linear best fit for the data provides an estimate for a (this was demonstrated in Figure 5.12). This method has two major advantages over the previous one: it uses all the data from the tail, rather than fudging them due to the use of binning, and it allows one to easily analyze the tail even if the distribution as a whole does not subscribe to a power law.

Another graphical method is based on the relationship between the Pareto distribution and the exponential distribution. Consider a rescaled Pareto distribution, in which the samples are divided by the minimal value k. Because the distribution is scale invariant, it still has the same power-law tail. The survival function is

$$\Pr(X > x) = x^{-a}$$

This does not change if we apply some monotonic transformation to both the variable X and the value x on the left-hand side. For example, we can use a logarithmic transformation:

$$\Pr(\ln X > \ln x) = x^{-a}$$

But if we apply a transformation only to the value x, we must apply the same transformation also to the x in the right-hand side. Let us do so with an exponential transformation. This yields

$$\Pr(\ln X > x) = (e^x)^{-a}$$

which can be rewritten as

$$\Pr(\ln X > x) = e^{-ax}$$

which is the survival function of the exponential distribution with parameter $1/a$! This means that if we take the log of Pareto samples, they will be exponentially distributed. We can therefore draw a Q-Q plot using log-transformed Pareto samples as the data, and an exponential distribution as the model. This should yield a straight line, and the slope will allow us to estimate the parameter value a [565].

Crovella and Taqqu's aest Method

A central problem with all methods to find a distribution's tail index is defining exactly where the tail starts. In many cases the end of the LLCD is not completely straight, and the transition from the body to the tail is gradual. Thus there is some leeway is choosing what data to use, producing some uncertainty about the results.

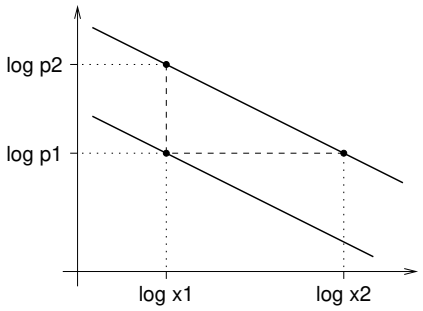
The aest method (for "a estimation") proposed by Crovella and Taqqu [158] solves this problem by automatically identifying those parts of the distribution that most closely follow the definition of a heavy tail. It does so by aggregating the data at several levels of aggregation, and using those data points where the aggregated LLCDs exhibit the correct relationship.

Consider a dataset $X = (X_1, X_2, X_3, \ldots)$, and two aggregated series $X^{(m_1)}$ and $X^{(m_2)}$, where aggregation is defined by

$$X_i^{(m)} = \sum_{j=(i-1)m+1}^{i\,m} X_j$$

Assume m_2 is a higher level of aggregation, for example $m_2 = m_1^2$ (Crovella and Taqqu use powers of two to create up to 10 LLCDs with growing levels of aggregation). If X is heavy-tailed, and we plot the LLCDs of $X^{(m_1)}$ and $X^{(m_2)}$, we expect to get two parallel straight lines with the same slope.

Now consider three points on these LLCDs, as illustrated to the right. The first is on the first LLCD, and indicates that with an aggregation level of m_1 there is a probability of p_1 of seeing values in excess of x_1. The other two are on the second LLCD, and correspond to the coordinates of the first point. Specifically, x_2 is identified as the value that, when using an aggregation level of m_2, is exceeded with the same probability p_1. Likewise, p_2 is the probability of exceeding the original value x_1.



The fact that the X_is are assumed to come from a stable distribution implies that

$$\sum_{i=1}^{n} X_i \overset{\mathrm{d}}{\sim} n^{\frac{1}{a}} X$$

where $\overset{\mathrm{d}}{\sim}$ means "has the same distribution". But n is just the level of aggregation, so this is the same as writing

$$X^{(m)} \overset{\text{d}}{\sim} m^{\frac{1}{a}} X$$

Now, by changing sides, we can find the relationship between different levels of aggregation. This is simply

$$\frac{1}{m_1^{1/a}} X^{(m_1)} \overset{\text{d}}{\sim} \frac{1}{m_2^{1/a}} X^{(m_2)} \tag{5.5}$$

This allows us to give expressions for the distances $\log p_2 - \log p_1$ and $\log x_2 - \log x_1$ between the two LLCDs just pictured. Let us start with the first one. Assuming the distribution has a heavy tail, and that we are looking at x values that are large enough to indeed be in the tail, then by definition $\Pr(X > x) = x^{-a}$. Upon aggregation by a factor of m, this probability also grows by the same factor, because exceeding the value x will typically be the result of just one of the samples being large enough to exceed it by itself. Therefore $\Pr(X^{(m)} > x) = mx^{-a}$. Calculating the vertical distance then gives

$$\begin{aligned}
\log p_2 - \log p_1 &= \log \Pr(X^{(m_2)} > x) - \log \Pr(X^{(m_1)} > x) \\
&= \log(m_2 x^{-a}) - \log(m_1 x^{-a}) \\
&= (\log m_2 - a \log x) - (\log m_1 - a \log x) \\
&= \log m_2 - \log m_1
\end{aligned}$$

Importantly, this calculation depends only on m_1 and m_2. It can therefore serve to identify points where x is in the tail, and, moreover, exhibits heavy-tailed behavior.

Once we identify these points, we can calculate the horizontal distance. By construction we know that $\Pr(X^{(m_2)} > x_2) = \Pr(X^{(m_1)} > x_1)$. By using Equation (5.5), we also know that for any x

$$\Pr\left(\frac{m_1^{1/a}}{m_2^{1/a}} X^{(m_2)} > x\right) = \Pr\left(X^{(m_1)} > x\right)$$

Using this for x_1 in the above equation, we derive

$$\Pr\left(X^{(m_2)} > x_2\right) = \Pr\left(\frac{m_1^{1/a}}{m_2^{1/a}} X^{(m_2)} > x_1\right)$$

Therefore $x_2 = \left(\frac{m_2}{m_1}\right)^{1/a} x_1$, and the distance is

$$\begin{aligned}
\log x_2 - \log x_1 &= \log \left(\tfrac{m_2}{m_1}\right)^{1/a} x_1 - \log x_1 \\
&= \tfrac{1}{a}(\log \tfrac{m_2}{m_1}) + \log x_1 - \log x_1 \\
&= \tfrac{1}{a}(\log m_2 - \log m_1)
\end{aligned}$$

Thus by measuring the horizontal distance, we can find the tail index a.

Putting this all together, the algorithm is as follows:

1. Create LLCD plots at multiple levels of aggregation (e.g., powers of 2).

2. Select the upper 10% of the data as potential tail points.
3. For each tail point on each LLCD, do the following:
   (a) Measure the vertical distance to the next LLCD. If it is within 10% of the expected value based on the aggregation levels, accept this point. Otherwise ignore it.
   (b) If the point is accepted, measure the horizontal distance to the next LLCD and use this to get an estimate for a.
4. Find the average of all the estimates of a that were generated.

Plotting the LLCDs and showing the accepted points also gives a qualitative view of how many of the tail points were accepted.

Maximum Likelihood Estimation

The second approach is to use maximum likelihood estimation. Following the procedure described in Section 4.2.3, we write the log-likelihood function for the Pareto distribution

$$\ln L(a, k \mid x_1, \ldots, x_n) = \sum_{i=1}^{n} \ln \frac{a\, k^a}{x_i^{a+1}}$$

Rewrite $\frac{a\,k^a}{x_i^{a+1}} = \frac{a}{k}\left(\frac{x_i}{k}\right)^{-(a+1)}$ and then differentiate with respect to a:

$$\begin{aligned}\frac{\partial}{\partial a}\ln(L) &= \sum_{i=1}^{n} \frac{\partial}{\partial a} \ln \frac{a}{k} + \sum_{i=1}^{n} \frac{\partial}{\partial a} \ln \left(\frac{x_i}{k}\right)^{-(a+1)} \\ &= \sum_{i=1}^{n} \frac{1}{a} - \sum_{i=1}^{n} \ln \left(\frac{x_i}{k}\right)\end{aligned}$$

Equating this to 0 reveals that the estimate for a is

$$\hat{a} = \frac{1}{\frac{1}{n}\sum_{i=1}^{n} \ln \frac{x_i}{k}} \tag{5.6}$$

Note that only x_i from the tail should be included, and that k is estimated as the minimal value (that is, where the body ends and the tail begins).

If the distribution is discrete, as is the case for file sizes, for example, a better estimation is obtained by using $k - \frac{1}{2}$ rather than k [141]. However, this is typically meaningless with the large values and number of samples typical of workload data.

The Hill Estimator

The Hill estimator is based on the largest k samples[5]. The formula is [334]

[5] This has nothing to do with the parameter k of the Pareto distribution; it is just one of those unlucky coincidences of commonly used notation.

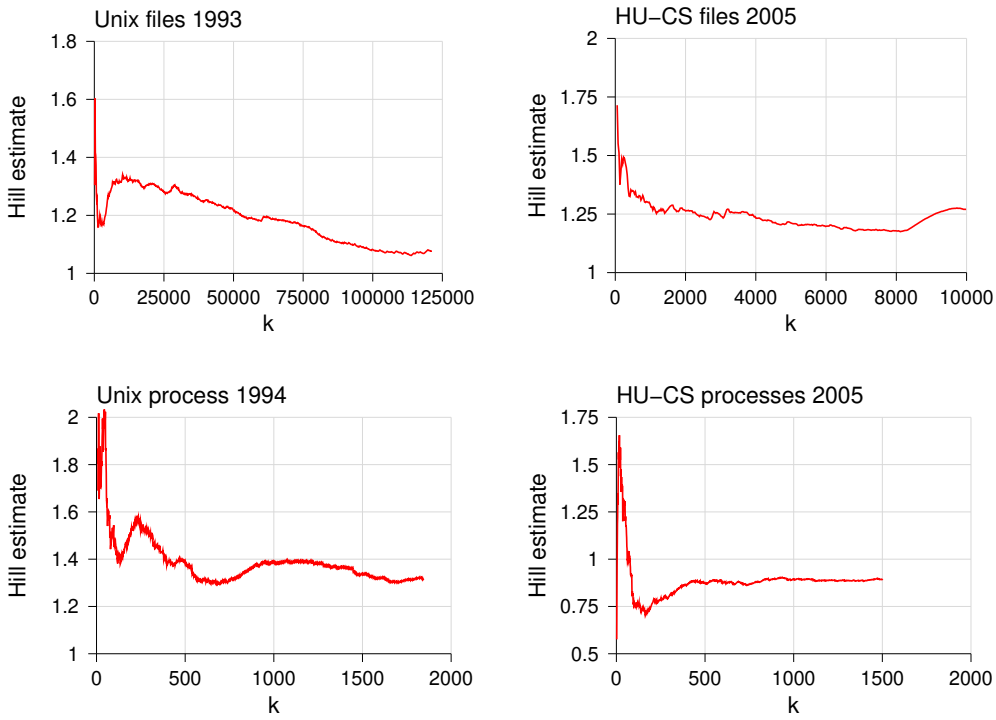


Figure 5.19: *The Hill estimator for several datasets used above.*

$$\hat{a}_k = \frac{1}{\frac{1}{k}\sum_{i=1}^{k} \ln \frac{X_{(n-i)}}{X_{(n-k)}}} \tag{5.7}$$

where $X_{(m)}$ is the mth order statistic (i.e., the mth largest sample; $X_{(1)}$ is the smallest and $X_{(n)}$ the largest). This is actually the maximum likelihood estimate when only these k samples are considered to be the tail.

The way The Hill estimator is used is to evaluate it for a sequence of values of k, and then plot $\hat{a}_k$ as a function of k. If the value seems to converge, this is taken as the estimate for a (Figure 5.19).

However, in many cases the Hill estimator does not appear to converge, but rather fluctuates in a certain range or even continues to change monotonically. For example, consider the Unix 1993 file-size data shown in the top left plot of Figure 5.19, which displays only the top 1% of the dataset. Figure 5.20 extends this to the top 50% (i.e., well into the body of the distribution). As the graph shows, the estimate continues to drop as more data points are added, from about 1.33 for the top 0.1%, through about 1.08 when the top 1% are used, and down to a mere 0.66 when a full 50% of the data are used. Although this last value is obviously bogus because it includes too much data, there is no obvious indication of which value in the spanned range is the best to use.

It has therefore been proposed to use a rescaling for k, replacing it by a parameter θ in the range $[0, 1]$, and plotting $\hat{a}_{\lceil n^\theta \rceil}$ as a function of θ (where n is the total number of

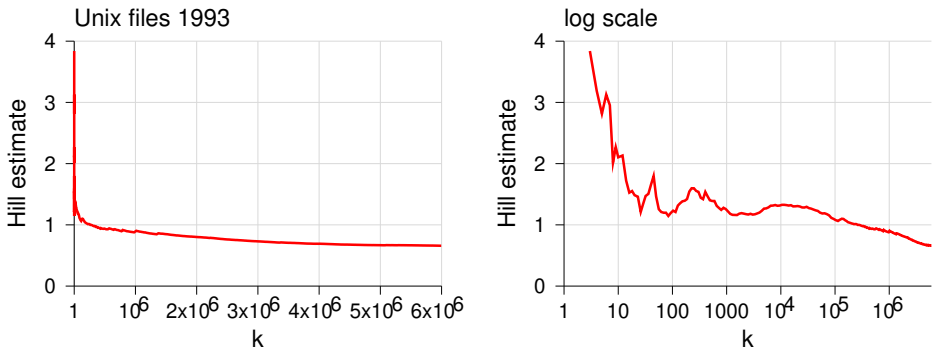


Figure 5.20: *Calculating the Hill estimator for an extended dataset.*

samples, and therefore $n \geq k$). In some cases this makes it easier to identify the plateau in the graph [565]. Another alternative that has also been proposed is to simply use a logarithmic scale, as shown in Figure 5.20.

Estimation for a Truncated Distribution

The above procedures may not work well for data that should be modeled by a truncated Pareto distribution. Recall that the LLCD of a truncated Pareto distribution drops towards the truncation point, rather than retaining the characteristic straight line (Figure 5.23). Thus using a graphical approach may seem to indicate that the correct model does not have a heavy tail at all. Alternatively, when an analytic approach such as the Hill estimator is used, we will get an overestimate of a due to trying to fit the reduced probability for high values at the tail. Adaptations of the Hill estimator that take this into account were developed by Beg [62], Aban et al. [1], and Chakrabarty and Samorodnitsky [115]. Chakrabarty and Samorodnitsky also suggest a statistic to determine whether or not the data corresponds to a truncated heavy tail. In essence it is based on the ratio of the sum of all samples to the largest sample, with a ratio close to 1 indicating an effective heavy tail whereas a large ratio indicates truncation.

In a related vein, it may be that the distribution is in fact not truncated, but the observations are. Of example, this may happen when the cost (or overhead) to obtain tail samples is excessively high. Gomes et al. [292] have developed an adapted estimator for this case.

Comparison of Results

Applying the methods described in this section to the four datasets used throughout yields the results shown in Table 5.1. As we can see, the results obtained by different methods exhibit a reasonable agreement with each other, which raises our confidence in them. Indeed, it is recommended to use several methods and compare the results for cross-validation.

Dataset	*LLCD*	*ML*	*Hill*
Unix files 1993	1.25	1.08	1.1–1.3
HU-CS files 2005	1.31	1.31	∼1.25
Unix runtimes 1994	1.38	1.32	∼1.35
HU-CS runtimes 2005	0.84	0.89	0.88

Table 5.1: *Results of different methods to estimate the tail index of a heavy-tailed distribution, when applied to the top 1% of samples.*

5.4.2 Generalization and Extrapolation

The problem with all the procedures for modeling heavy tails is that data regarding the tail is sparse by definition [295, 115]. For example, a single very large sample may sway the decision in favor of "heavy" [131]. But is this the correct generalization? The question is how to identify the nature of the underlying distribution without having adequate data.

When the LLCD plot is a straight line, this implies a heavy (power-law) tail. For example, Barford and Crovella made such an observation when studying early data regarding request sizes on the world wide web, where the maximal size was about 10^6 bytes. They therefore included a heavy tail as one of the features of their SURGE workload generator [57]. Years later this decision was justified when further data showed that the heavy tail indeed extends at least up to a size of 10^9 bytes [330]. But in principle it could also have been truncated at say 10^7.

Nevertheless, claiming a truly heavy-tailed distribution is unrealistic, because such a claim means that unbounded samples should be expected as more and more data is collected. In all real cases, samples must be bounded by some number. Therefore the correct model, even if the data seems heavy-tailed, is a truncated Pareto distribution. This leaves the question of where to put the bound.

Connection Box: Extreme Value Theory

The need to extrapolate beyond the available data is not unique to computer workload modeling. Such extrapolation is extremely important in finance, insurance, and engineering. For example, engineers may need to design structures that withstand "100-year storms", meaning storms of a force that is experienced on average only once every 100 years. Data about such storms is necessarily very sketchy.

Extreme value theory provides some direction. Given n samples from some process, their maximum is

$$M_n = \max_n\{X_1, X_2, \ldots, X_n\}$$

Extreme value theory deals with the distribution of M_n, and also with the expected number and magnitude of events above a given high threshold.

An interesting example comes from the sinking of the ship *M. V. Derbyshire* during a typhoon in 1980. This was attributed to flooding due to a failure of the hatch cover of Cargo Hold 1 (the foremost of nine holds running the length of the ship). The question then became what was the probability that the typhoon had caused waves to pound on the cover with a force that exceeded the safety standard of the time. A statistical analysis was

conducted to assess this probability, based on a scale model, estimating wave conditions from satellite data, and fitting a generalized Pareto distribution to the extreme values [326, 325]. It concluded that waves large enough to sink the ship were reasonably probable provided the hold cover had already suffered prior damage.

End Box

Truncating the Pareto Distribution

The finite nature of all workload items clearly indicates that we need to postulate a certain upper bound on the distribution. Thus we should actually use the truncated version of the Pareto distribution. But the question remains of where to truncate the distribution. Should it be at the highest observed sample, or higher than that? If so, how much higher?

A lower bound on the truncation point is provided by the largest sample seen in the data. Obviously, if such a sample exists, its probability is larger than 0. The assumption that the appropriate truncation point coincides with the largest sample is implied in those methods that fit a phase-type distribution to heavy-tailed data. The way they fit the heavy tail is to use a set of exponentials, each of which extends the distribution beyond the point at which the previous one decays into oblivion. But nothing extends the distribution beyond the last exponential, which is typically designed to capture the last data points (Figure 5.21).

However, we might not have enough samples. Maybe if we had twice as many samples, or 10 times as many, we would have seen one that is much deeper into the tail. Another option is therefore to assume that everything that is not excluded outright is possible. The truncation point is then chosen to be some wild upper bound. For example, if the total capacity of a storage system is 10 TB, then the maximal file size is set to 10 TB (even though it is unrealistic to expect a single file to really occupy all the available space). If the duration of funding research projects is four years, then the maximal job runtime is set to four years (even though it is unrealistic for a computation to take the full time of a project). In effect, we are using the functional shape of the distribution (the power-law tail) to extrapolate way beyond what we have actually seen.

But what if the range from the largest observed sample to the upper bound is large? If we choose the largest sample we might exclude larger samples that may in fact occur. If we choose a wild upper bound we might introduce large samples that actually cannot occur. In either case there is a danger that this decision will have a large impact on performance evaluation results, because the largest samples dominate the workload.

Figure 5.21 shows how the two models diverge for a real dataset. In the dataset (Unix 1994 process runtimes) there are 184,612 samples, the largest of which is 1573 seconds. If we need more samples for an evaluation, we need to extrapolate and extend the distribution. If we need 10 million samples, for example, and use a hyper-exponential model, the largest sample we may expect to see is about 3700 seconds. But if we choose the Pareto model, we can expect samples as large as 25,000 seconds! If we need even more samples, the difference continues to grow.

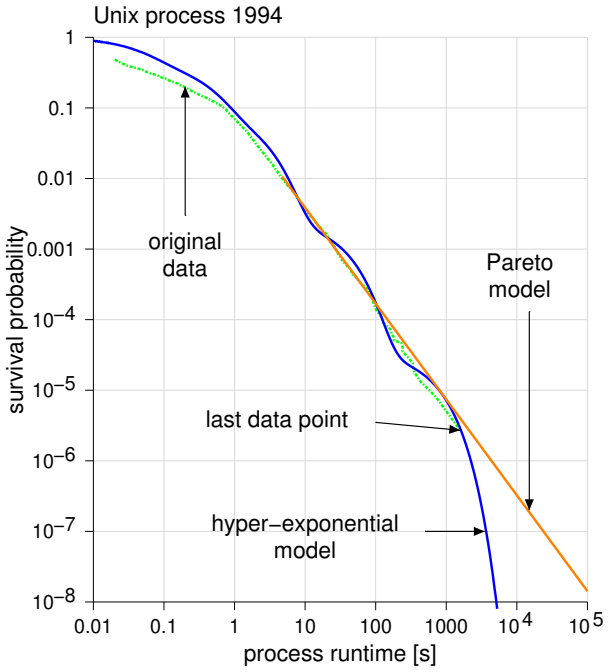


Figure 5.21: *LLCD plot comparing a hyper-exponential model and a Pareto model beyond the range covered by the data.*

Dynamic Truncation

An intriguing option is to set the truncation point dynamically according to need. If we only need a small number of samples from the distribution, there is a negligible chance that we will get one from the tail. In fact, getting a large sample from the tail would not be a characteristic outcome. To prevent this, we can truncate the distribution at a point that depends on how many samples we need. For example, if we need n samples, we can truncate the distribution at a point where $\bar{F}(x) \approx \frac{1}{2n}$ (inspired by Chauvenet's criterion for removing outliers).

Obviously, this approach does not really solve the problem, because we still do not know what is the correct truncation point. But it allows us to define a certain *horizon*, characterized by the number of samples n. With this horizon, we should probably only delve so far into the tail of the distribution, and no more. With a farther horizon, the probability of seeing even larger samples becomes realistic, and we should delve deeper into the tail. Controlling how far we go enables us to qualify the evaluation results, and to claim that they are representative for the given horizon.

Dynamic truncation also allows us to empirically assess the effect of the truncation. We can redo the evaluations using the unbounded Pareto distribution, and a version truncated according to the expected horizon. If the results are consistent with each other, we can probably rely on them, because the tail does not make much of a difference. If they

RNG	*Bits*	*Max*
rand	16	32,768
random	32	~ 2 billion
drand48	48	$\sim 2.8 \times 10^{14}$

Table 5.2: *The characteristics of three random number generators. These figures are for the implementation on a BSD system around the year 2000. Current implementations differ; in particular, rand is often an alias for random.*

are not, we know that our results depend on the largest samples observed. But even so, we can still precisely quantify the probability that the results for the limited horizon are relevant. Similar to this, Greiner et al. employ truncation in the interest of tractability, and compare the results obtained from truncated distributions with those of the limiting heavy-tailed case [302].

Random Number Generator Limits

Pareto random variates are generated in two steps. First, generate a uniform variate u in the range $[0, 1]$. Then calculate $x = \frac{k}{u^{1/a}}$. The result is a Pareto variate with parameters k and a.

Looking closer at this procedure, we note that essentially it uses the reciprocal of a small number. In particular, the large samples from the tail of the distribution are produced from the smallest numbers provided by the random number generator. But because of the finite nature of computer arithmetic, there exists some minimal (nonzero) number that can be produced. This then sets a limit on the maximal value from the tail of the Pareto distribution that can be produced.

Table 5.2 lists the limits of three random number generators. Essentially, the limit is related to the number of random bits produced; this sets the maximal value that can be produced by a Pareto distribution with parameter $a = 1$. In effect, the random number generator may impose an unintended truncation on the distribution. This is naturally undesirable, because the samples from the tail have a dominant role in performance evaluation results.

As a simple example of the effect of such limitations, Figure 5.22 shows the running average of samples from a Pareto distribution generated using the three random number generators. The rand generator is the most limited. As a result the samples do not even begin to approximate a situation in which the mean is infinite; instead, the running average quickly converges to about 11, which is close to $\ln(32{,}768)$.

The random and drand48 generators can create much larger values, so many more samples are needed before their limitations become apparent. However, in the range of billions of samples, the running average of the random generator is also seen to converge. Only the drand48 generator is still usable if so many samples are required.

As a rule of thumb, it is recommended to always use double-precision floating-point variables, as well as the best random number generator you have access to.

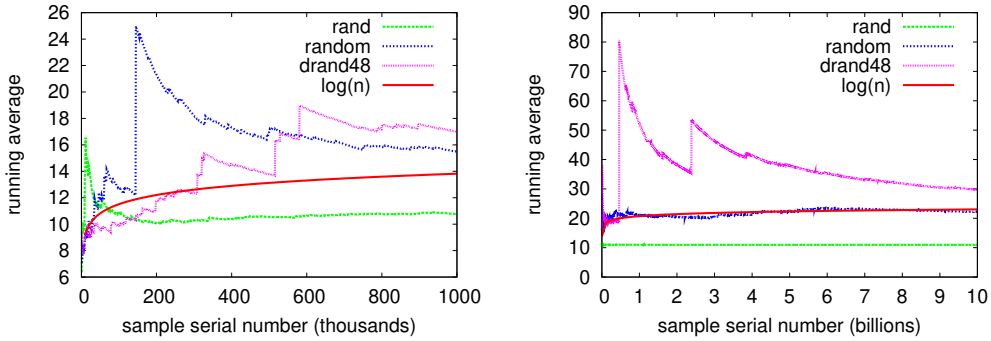


Figure 5.22: *The variability observed in samples from a Pareto distribution depends on the capabilities of the random number generator used.*

Distribution	*Mid-tail*	*Far tail*	*Moments*
truncated Pareto	polynomial	truncated	converge
phase-type	polynomial	exponential	converge
lognormal	near polynomial	sub-exponential	converge
Weibull	near polynomial	sub-exponential	converge
Pareto	polynomial	polynomial	diverge

Table 5.3: *Tail characteristics of alternative model distributions.*

The Lognormal and Other Candidate Distributions

Although heavy tails are by definition Pareto distributed, it is nevertheless sometimes possible to model data that seems to be heavy-tailed using other distributions. This is especially appropriate when the original data does not quite fit a power-law tail (e.g., when the end of the LLCD plot is not completely straight). Such a deviation may indicate that a lognormal, Weibull, or truncated Pareto model may be more appropriate. These models are especially appealing in the context of mathematical modeling, because all their moments converge. The most commonly used alternatives are contrasted in Table 5.3. The choice of which alternative to use may be guided by goodness-of-fit considerations (e.g., using the Kolmogorov-Smirnov test [141], but a test that is more sensitive to deviations in the tail may be advisable).

The truncated version of the Pareto distribution was considered earlier. By also shifting the distribution we obtain a combination of three desirable features (Figure 5.23):

1. It models both the body and the tail.
2. It displays a straight LLCD plot over several orders of magnitude.
3. It has finite moments.

Another possibility is that the tail has several components, so it is not a "smooth" Pareto tail [332].

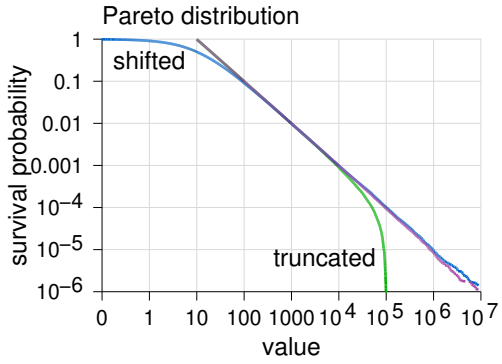


Figure 5.23: *LLCDs of variants of the Pareto distribution.*

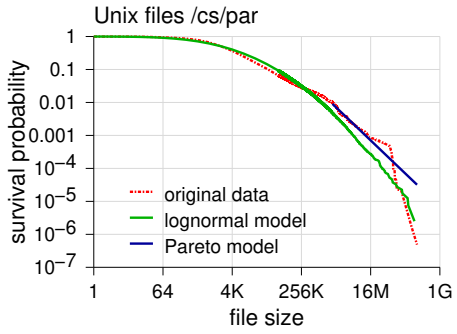


Figure 5.24: *LLCD plot comparing a lognormal model and Pareto model for file size data.*

Applying truncation implies a strict upper bound on the possible values. An alternative is to use a phase-type hyper-exponential distribution that mimics the power-law structure for a certain range of values, but then decays exponentially at the end [253, 208]. The procedures for doing so were described in Section 4.4.3. Importantly, it is also possible to construct mixtures of exponentials to fit distributions with various modes, not only those with a monotonically decaying tail [97, 341, 568, 569].

Another possible model is provided by the lognormal distribution, which has been shown to fit various datasets from different fields of study [445, 196, 189]. Like the shifted and truncated Pareto distribution, this model has the additional benefits of potentially fitting the whole distribution, rather than just the tail, and of having finite moments.

For example, there have been successful attempts to model file sizes using a lognormal distribution rather than a Pareto distribution [188]. An example is given in Figure 5.24, which shows the /cs/par file system dataset, which has about a million files. According to the LLCD plot, this dataset has a tail that does not quite conform to a power law. To create a lognormal model, the mean and standard deviation of the log-

transformed data were computed. While the result does not completely match the quirks of the tail of the data, it does provide a reasonable fit for the distribution as a whole.

Finally, the Weibull distribution has characteristics that are similar to those of the lognormal. Laherrére and Sornette present evidence that this distribution provides excellent fits for many phenomena in nature and society [423]. However, it has so far not been used much in the context of workload modeling, perhaps because fitting the distribution's parameters is more complicated than for other candidate distributions.

The main consideration discussed so far when deciding what distribution to use was the fit to the data. An additional consideration may be a possible underlying model of how the data were generated. This is discussed next.

5.4.3 Generative Models

Given the typical sparsity of data, the choice of which distribution to use is a difficult one. We typically do not have enough data, and the consequences of our choice might be very serious. An alternative approach is therefore not to try to model the data itself, but instead find a convincing model of how the data was generated.

The general framework of generative models is typically as follows. The model is based on a dynamic population, and attempts to describe how this population grows and changes with time. This is done by some growth rule that is applied iteratively: given the population that exists at some point in time, the rule is applied to generate a new member of the population. Using this rule, many new members may be generated one after the other. Importantly, the rule depends on the existing population. Thus *the observed population cannot be considered to be a simple sampling from a given distribution* [627]. Rather, the distribution characterizing the population changes as a result of applying the rule and generating additional members. But in the models the distribution does in fact converge to some asymptotic distribution, which would be reached if we would continue to add members indefinitely. This is the distribution that is generated by the model.

Several models have been shown to lead to skewed distributions and possibly heavy tails, including preferential attachment and multiplicative processes [497, 514]. And they have been rediscovered multiple times, sometimes with slight variations [626]. We review them here, but also note that it is not always clear that they are relevant for workload modeling. For example, they do not seem to be applicable to the question of why process runtimes are heavy-tailed.

Preferential Attachment

Preferential attachment has been proposed as the main mechanism leading to the formation of scale-free networks, such as the network of HTML pages and hyperlinks that form the world wide web. In the simplest version of the model, we start with a population of m items (pages). Then, at each step, we add a new item and link it to m existing items. Preferential attachment is used when selecting the m items to link to: they are selected at random, but with probabilities proportional to their current connec-

tivity [49, 513]. Thus if there are currently n items in the system, and item i already has k_i links, the probability to select it will be

$$\Pr(i) = \frac{k_i}{\sum_{j=1}^{n} k_j} \tag{5.8}$$

In other words, items that already have a relatively high number of links have a higher probability of attracting even more links — a classic "the rich get richer" phenomenon. This is justified in real settings by observing, for example, that sites that are highly ranked by search engines have more visibility, and therefore have a better chance to be linked from other sites, leading to even higher visibility [132]. The model can also be extended to allow for link deletions [255]. Essentially the same model has also been suggested in other domains (e.g., the creation of subdirectories in a file system [18]).

The following argument demonstrates that this process indeed leads to a heavy-tailed distribution of node degrees [50]. Let k_i denote the degree of node i (note that this includes both links from this node to other nodes, which were selected at random when the node was created, and links from other nodes to this one, which were added later). At time t, m additional links will be added to the system. The expected number that will point to node i is $m\,k_i(t)/\sum_j k_j(t)$ (note that it is possible for several parallel links to be created). The denominator of this expression is actually twice the number of links in the system, because each is counted on either end. As m links are added in each time step, we have $\sum_j k_j(t) = 2mt$. Putting this together leads to the following differential equation giving the rate of growth of k_i:

$$\frac{\mathrm{d}}{\mathrm{d}t} k_i(t) = m\,\frac{k_i(t)}{2mt} = \frac{k_i(t)}{2t}$$

The solution of this differential equation is

$$k_i(t) = m\sqrt{\frac{t}{t_i}} \tag{5.9}$$

where t_i is the time step when node i was added; assuming the nodes are simply numbered in the order they are added, $t_i = i$. Thus the number of links to a node grows as the square root of time, but for early nodes time effectively passes faster than for later arriving nodes.

To characterize the tail of the distribution of node degrees, we need to find the distribution function $\Pr(k_i(t) < k)$. By using Equation (5.9) and changing sides, we get

$$\Pr(k_i(t) < k) = \Pr\left(m\sqrt{\frac{t}{t_i}} < k\right) = \Pr\left(t_i > \frac{m^2 t}{k^2}\right)$$

But nodes are added uniformly over the time t, one in each time unit. Therefore

$$\Pr\left(t_i > \frac{m^2 t}{k^2}\right) = 1 - \Pr\left(t_i \le \frac{m^2 t}{k^2}\right) = 1 - \frac{m^2}{k^2}$$

(this is indeed a probability, because k must include the initial m outgoing links, so $k \geq m$). Combining these two equations we finally obtain the distribution

$$\Pr(k_i(t) < k) = 1 - \frac{m^2}{k^2}$$

This is the CDF of the Pareto distribution, with a tail index of 2. A model in which links are created also among existing nodes, and not only from new nodes, leads to a distribution with a tail index of 1 (that is, a heavier tail) [148].

Note that this model has two important elements that are both required to produce the heavy-tailed distribution of node degrees [49, 50]:

1. Preferential attachment is required in order to create nodes with very high degrees. In more conventional random graph models, such as the celebrated model of Erdős and Rényi, each pair is connected with a probability p. If p is above the critical value of $\frac{1}{n}$ in an n-node graph, then the vast majority of nodes will form a single large connected component. Moreover, the probability of having k links will be Poisson distributed with mean pn. The tail of the distribution decays exponentially with k and is not heavy.

   The sequential nature of preferential attachment leads to positive feedback, and allows nodes with a relatively large connectivity to continue to grow faster than the others. This is what causes the distribution to become heavy-tailed.

2. Growth is needed because if edges are just added with no new nodes, eventually all nodes will be highly connected and there will not be enough "small" nodes. An example of this effect was shown in Figure 5.16, where an increasing number of words selected from a finite vocabulary eventually cause a deviation from the observed Zipf distribution.

Connection Box: Evolution, Linguistics, Economics, and Citations

The first to show that using a stochastic process to model the evolution of a population may lead to a distribution with a power-law tail was Udny Yule [748]. Appropriately, this work was done in the context of providing a mathematical model for the theory of evolution. The motivation was to try and explain power-law tails in the distribution of the number of species in different genera.

Yule's work was later extended and updated by Simon [627]. Simon showed that essentially the same model may be applied to diverse situations, including linguistics (explaining the origin of Zipf's law), geography (the distribution of city sizes), and economics (harking back to Pareto's work on the distribution of wealth).

In modern terminology, the basic model can be regarded as a balls and urns model. The urns represent the members of the population, and the number of balls in an urn represent a property of these members; for example, in Yule's original work the urns were genera, and the balls were species in each genus. The modification rule adds balls in the following manner:

1. With probability α, create a new urn and place the new ball in it. In the evolutionary context, this represents a speciation where the change from the original species is so great that the new one is considered a new genus altogether. α is typically a very small constant, close to zero.

2. Alternatively, add a ball to an existing urn. The urn to which the ball is added is selected at random, with probability proportional to the number of balls already in it (exactly as in Equation (5.8)). In the evolutionary context this represents mutations leading to the creation of a new species. Assuming that the new species originated from some existing species, it is natural to assign it to a genus according to the number of species already in that genus.

Applying this rule repeatedly for a long time leads to a heavy-tailed distribution of balls in urns. The intuition is simple. Assuming we start with a single urn, it will accumulate around $\frac{1}{\alpha}$ balls before another urn is created. Even after a competing urn exists, the number of balls in the first urn will continue to grow at a higher pace, because it already has more balls in it. This positive feedback effect means that the initial head start of early urns is essentially impossible to make up, leading to a distribution where the first urns accrue very many balls while late urns mostly remain with a single ball or very few balls.

The following table shows Simon's mapping of this model to different contexts [627]. Note that in the latter two or three examples it makes sense to only consider elements larger than a certain threshold. Doing so requires a variant of the model in which new urns are populated with multiple balls, rather than only with one.

Context	*New urn*	*New ball*
evolution	speciation leading to a new genus	speciation leading to a new species within the same genus, proportional to number that already exist
linguistics	emergence of a new word	use of existing word, based on imitation of or association to existing text
science	appearance of a new author	publication of another paper by an established author
geography	a small town grows enough to become a city	population of city grows naturally and by migration, in proportion to existing population
economics	another person passes the threshold of minimum income for consideration	investment leads to generating new wealth in proportion to existing wealth

Related models have also been studied by others [621, 273, 207], and the issue of word frequencies has been the focus of a fierce debate between Simon and Mandelbrot [464, 628].

Other applications of preferential attachment have since been proposed in additional fields. One example is the study of citations to scientific literature by Price [554]. The premise there is that papers that already have multiple citations have higher visibility, and therefore garner additional citations at a higher rate (and in fact, this can occur simply as a result of citations being copied without the paper actually being read [625]). Obviously this is essentially the same idea as the preferential attachment of web pages noted earlier.

End Box

A different approach suggests that the ubiquity of power-law tails is a result of observing a growing number of individual processes that each grow exponentially [562, 207]. Consider such a process $X(t) = e^{\mu t}$. If we observe it at a certain instant in time T, we will sample the value $X_T = e^{\mu T}$. Now assume that there are many such processes

in parallel, and that each is observed at a random time since its start. A process that is observed a long time after it started thus represents an old process that has been going on for a long time, whereas one that is observed soon after it started is a newly begun process.

Assume the observations are done randomly, with a constant rate; in other words, the sample point can occur at any instance with equal probability. It is therefore similar to a Poisson arrival. The time T until the sample is therefore exponentially distributed, say with parameter ν. We can now calculate the distribution of X_T:

$$\begin{aligned}\Pr(X_T < x) &= \Pr(e^{\mu T} < x) \\ &= \Pr(T < \tfrac{\ln x}{\mu}) \\ &= 1 - e^{-\nu \frac{\ln x}{\mu}} \\ &= 1 - x^{-\nu/\mu}\end{aligned}$$

which is a Pareto distribution with parameter $a = \nu/\mu$. The same result holds also if the sampled process is not deterministic, but grows exponentially in expectation [562]. This model has the advantage of producing a heavy-tailed distribution in the transient state, as opposed to previous models that only produce such distributions asymptotically.

In short, different models can lead to the same end result, provided they combine growth and increasing rates.

Multiplicative Processes

Another type of generative model is the multiplicative process. Such a model for file sizes was proposed by Downey [188]. It starts with a single file of size s_0. It then adds new files by selecting an existing file of size s and a factor f from a given distribution, and adding a new file with size $f \cdot s$. This process is based on the intuition that new files are typically created in one of three ways:

- By being copied from another file (the special case where $f = 1$).
- By some process of translation (e.g., when a program is compiled into an executable).
- By editing an existing file.

Assume now that this process is applied repeatedly many times. The resulting distribution of file sizes is lognormal with a strong mode at the original file size, representing copying and small changes. To see this, consider the following. In step i we pick a file of size s_i and a factor f_i, to generate a new file of size $s_i^{new} = f_i \cdot s_i$. But s_i was generated in the same way in some previous step, so actually s_i^{new} is the product of many factors:

$$s_i^{new} = s_0 \cdot \prod f_j$$

Taking the log gives $\log s_i^{new} = \log s_0 + \sum \log f_j$. Assuming that the f_j are independent and $\log f_j$ has finite variance implies that $\log s_i^{new}$ will have a normal distribution (due to the central limit theorem). Hence s_i^{new} will have a lognormal distribution.

A multiplicative process such as this one can be visualized as a tree: each file is a node, connected to the root by the path of intermediate files that participated in its creation. The branches of the tree are labeled by the factors f, with the size of each file given by the product of the factors along the branch from the root of the tree. Such a view highlights a potential problem with the model: if one of the factors f near the root happens to come from the tail of the distribution, this will have a strong effect on an entire branch of the tree, leading to a more dispersive distribution of sizes. A possible solution is to use a forest instead of a tree, that is, allow multiple initial files [498].

Model Selection and Sensitivity to Details

Although the previously discussed generative models may be used to justify one distribution or another, this does not necessarily mean the choice of distribution is clear-cut.

One problem is that there are cases where different models may be claimed to apply to the same situation. For example, when studying the distribution of city sizes, Gabaix suggested a model leading to a heavy-tailed distribution [273], whereas Eeckhout proposed a model leading to a lognormal distribution [196]. In the context of modeling the world wide web, Barabási and Albert proposed the preferential attachment model including system growth [49], whereas Huberman and Adamic proposed a multiplicative model with different growth rates and system growth [353]. Interestingly these different models lead to the same outcome — a heavy-tailed distribution.

Another problem is that all the models are fragile in the sense that minor changes lead to different behavior. For example, Perline has shown that monkey typing where letters have equal probabilities leads to Zipfian word frequencies (that is, a Pareto distribution), but with unequal letter probabilities the resulting distribution is lognormal [544]. Likewise, the straightforward multiplicative model leads to a lognormal distribution, but it can be turned into a Pareto distribution using any of the following modifications [497, 295]:

- Allow the size of the system to grow with time, so that new items are introduced continuously. This supply of small items offsets the older ones that continue to grow and produce the tail [352].
- Use different growth rates (i.e., different distributions of multiplicative factors) for different elements. Those that grow faster will then create the heavy tail.
- Introduce a normalization step in which after multiplication, the product may be adjusted by comparing with the average of all current values [462].
- Enforce a lower limit, in effect allowing items to grow but not to shrink [462, 497]. This enhances the probability of strong relative growth enough to create a heavy tail.

Thus the heavy-tail and lognormal models are actually closely related. Similar models can also lead to a Weibull distribution [270, 423].

Moreover, the differences between the models may be small in practical situations. Only when very many samples are observed do we see enough data from the tail to

observe the difference between the distributions. In some cases this small difference is good: if a moderate number of samples suffices for our needs, then it does not matter what is the "true" model. But in other cases, notably when using mathematical analysis, the inability to select a model may be very problematic.

To read more: There are many papers on generative models that do or do not lead to power-law tails. A good survey with many references is given by Mitzenmacher [497]; another is by Newman [514]. Preferential attachment is analyzed in detail in [50, 78, 148, 513].

6

Correlations in Workloads

Modeling the distribution of each workload attribute in isolation is not enough. An important issue that has to be considered is possible correlations between different attributes, as well as between different samples from the same distribution.

Correlations are important because they can have a dramatic impact on system behavior. Consider the scheduling of parallel jobs on a massively parallel machine as an example. Such scheduling is akin to 2D bin packing: each job is represented by a rectangle in processors×time space, and these rectangles have to be packed as tightly as possible. Assuming that when each job is submitted we know how many processors it needs, but do not know for how long it will run, it is natural to do the packing according to size. Specifically, packing the bigger jobs first may be expected to lead to better performance [144]. But what if there is a correlation between size and running time? If this is an inverse correlation, we find a win-win situation: the larger jobs are also shorter, so packing them first is statistically similar to using SJF (shortest job first), which is known to lead to the minimal average runtime [419]. But if size and runtime are correlated, and large jobs run longer, scheduling them first may cause significant delays for subsequent smaller jobs, leading to dismal average performance results [450].

6.1 Types of Correlation

When we say that workload attributes are correlated, we mean that they exhibit similar behavior. For example, if parallel jobs with more than the average number of processes also run for longer than the average runtime, we say that size and runtime are correlated. In probability theory such relationships are represented by the notion of dependence: random variable X is said to be dependent on random variable Y if knowing Y tells us something about the value of X. When X and Y are strongly correlated, we can use Y to actually predict the value of X. If the correlation is weaker, knowing Y at least gives us information about the distribution of X. For example, both small and large jobs have similarly wide distributions of runtimes, but for small jobs the distribution is slightly skewed toward lower runtimes.

In the context of computer system workloads, we can identify three main types of correlations that are worthy of study:

1. When successive instances of a workload attribute are correlated with each other, rather than being independently drawn from the same distribution. As successive instances are considered, this is called short-range dependence.

   A well-known example is *locality of reference*. Consider the sequence of addresses in memory that are accessed by a computer program. We can collect all these addresses, create a histogram showing how many times each address is accessed, and use this as a model. But it is well known that computer programs do not access their memory in a random manner. Rather, each access tends to be near a previous access. This phenomenon of locality lies at the base of using memory hierarchies, in which a small fast memory can serve as a proxy for a much larger (but slower) memory space.

   The two classic forms of locality, spatial locality and temporal locality, are discussed in Section 6.2. This discussion is then generalized in Section 6.3, which presents locality of sampling.

2. When cross-correlation exists among distinct workload attributes. For each workload item, if we know the value of one of its attributes, we already know something about its other attributes as well.

   This is the closest to the conventional notion of correlation. We saw an example earlier. parallel jobs have two main attributes: size (number of processors) and duration (runtime). These attributes may be correlated with each other, meaning that large jobs, those that use more processors, also run longer. They may also be inversely correlated, meaning that the larger jobs actually run for less time. It is important to model such correlations because they may have a significant impact on performance evaluation results.

   Such cross-correlation among attributes and its modeling are discussed in Section 6.4. A special case, in which there exists a correlation with time, is discussed in Section 6.5.

3. Long-range dependence leading to the phenomenon of self-similarity. Related to the burstiness of workloads, it essentially states that bursts of activity occur at many different scales.

   Self-similarity was identified as a major element in modeling network traffic in the early 1990s. It was shown that the commonly used Poisson models of network traffic did not capture the burstiness observed in real data. The most important consequence is that when many traffic flows pass through the same link bursts of activity do not average out, as opposed to the predictions made based on Poisson models. This has important implications regarding the sizes of buffers that are needed and the probability of not being able to provide the required service.

   Self-similarity is a large subject in itself, largely separate from the other types of correlation discussed here. We therefore defer its treatment to Chapter 7.

6.2 Spatial and Temporal Locality

Locality is a special case of correlation of a variable with itself over short to medium ranges. It consists of sampling a local part of a distribution, rather than the entire distribution, a concept that is elaborated on in the next section.

Locality is an ubiquitous phenomenon [173]. Examples from computer workloads include the following:

- References to memory addresses by computer programs [95, 140, 306, 476], the first type of locality identified, remain the best-known example. Improving locality has also become one of the goals of algorithmic design, in the interest of better using the memory hierarchy [424, 28, 335]. When considered at the granularity of pages, locality lies at the basis of the working set model and of paging algorithms [171, 174, 172].
- References to files on a file server display locality, in the form of reusing files.
- References to blocks in a single file also display locality, although other regular patterns (e.g. sequential traversal) are also common.
- Accesses to the records of a database display locality, as do accesses to a key-value store, which may be regarded as a simple case of a database [42].
- Communications in parallel computers have locality, which can be exploited when mapping tasks to processors [77, 609, 472].
- Usage of addresses on a LAN or the Internet displays locality: if a message (or packet) was sent to destination X, additional messages to this destination may be expected [369, 366]. The same applies for sources.
- The same also applies at higher levels, e.g., retrieval of documents from servers of the world wide web [23, 381, 267].
- Finally, another type of locality altogether is value locality. It turns out that functions are often called repeatedly with the same parameter [139], and that as few as 10 distinct values occupy more than half of all memory locations [759].

The reason for this prevalence of forms of locality may be related to human work habits. For example, locality of reference has been linked to the "divide and conquer" approach of writing software, and, in particular, to the use of subroutines [173]. Many other types of locality may be linked to the fact that human users of computer systems focus on one thing at a time.

6.2.1 Definitions

The principle of locality was defined by Denning as having three features [174]:

1. There is a nonuniform distribution of references to memory pages.
2. The frequency with which a page is referenced changes slowly.

3. The correlation between the immediate past and immediate future is high, and tapers off as the distance increases.

But this is not a precise and measurable definition.

Locality necessarily deals with multiple samples, or instances, of the same entity. Examples include a sequence of references to memory (as was the case in Denning's work), a sequence of accesses to files, or a sequence of requests from a web server. We focus on the memory reference example for the sake of concreteness. In this case, the possible values come out of an ordered set: the set of possible memory addresses. Each memory reference is a sample from this set. As references happen one after the other, the entire sequence can be represented by a sequence of random variables X_i, where i is the number in the sequence and X_i is the address referenced at step i. We call such a sequence a *reference stream*. We are interested in the relationship between X_i and X_j for js that are "close to i".

Practice Box: Compressing Traces

The way to study reference streams is by tracing all the addresses accessed by a running computer program. The problem with doing this is that the amount of data may be huge, as billions of addresses may be accessed each second. It is therefore necessary to compress the traces, both to save storage space and not to overwhelm the bandwidth of storage devices.

In principle, any lossless compression algorithm can be used. But the best compression is achieved by specialized algorithms that exploit the special properties of address traces. One such algorithm is Stream-Based Compression (SBC) [488]. This algorithm has distinct, special treatments for instructions and data. Instructions are encoded by creating a table of basic instruction streams, which are sequentially executed instructions between branch points (a similar idea is used for instruction caching in trace caches). The full instruction stream can then be encoded by listing the indices of the basic streams that are traversed. Data streams are encoded as the interleaving of multiple access vectors that have a constant stride. Encoding the start address, stride, and number of occurrences, rather than listing every individual address, may save considerable space.

End Box

Temporal Locality

One possible relationship in a reference stream is equality: we may find that $X_j = X_i$. This means that the same address is referenced again after d steps, where $d = j - i$. This is called *temporal locality*. Temporal locality is extremely important in computer systems and forms the basis for all caching schemes. If temporal locality exists, we can cache a requested item (e.g., the contents of a memory cell) and will thus have it at hand when it is requested again.

Spatial Locality

Another possible relationship is nearness. If the distance $s = |X_j - X_i|$ is small, it means that soon after referencing X_i we find a reference to the nearby address X_j. This is called

spatial locality. It is important because it allows for prefetching. For example, upon a request for X_i we may retrieve a whole set of nearby values, under the assumption that there is a high probability that they will be requested too. This is why cache memories store cache lines with several memory values, rather than a single memory value.

A special case occurs when the items in question come from an unordered set. For example, there may be no natural order on the pages served by a web server. In this case there is no natural definition for spatial locality. However, it is possible to identify sets of pages that tend to be requested together, and define such groups as localities.

A variant of spatial locality is *spatial regularity*. This means that successive references can be predicted because they obey a certain pattern, even though they are not close to each other [499]. The most common example is strided access. Consider a matrix that is stored in row-major format, but accessed by column. Successive elements in a column are separated by an entire row, so they may be quite distant from each other. But the access pattern is clearly deterministic, so this pattern can be detected and exploited by sophisticated memory systems.

Popularity

An issue that is closely related to temporal locality is popularity [381]. In most cases, not all items are equally popular. Some are used much more than others. For example, consider the following reference stream:

A B C D A C D B D A C B D C A B C D B A D C A B

Each of the four items — A, B, C, and D — is referenced six times, in random order. There is no appreciable locality. However, if the reference stream is reordered thus,

A A A A A A B B B B B B C C C C C C D D D D D D

significant temporal locality is immediately seen. But now consider a different stream, derived from the first one by identifying C and D with A:

A B A A A A A B A A A B A A A B A A B A A A A B

In this case references to A exhibit a strong temporal locality, but this is due to the fact that A is much more popular than B and appears three times more often. This pattern is in fact a common occurrence. Popularity distributions are typically highly skewed, and this accounts for a large part (but not all) of the temporal locality. The additional locality is caused by correlations among nearby references, as in the second reference stream above.

6.2.2 Statistical Measures of Locality

Locality can be visualized as a 2D probability surface $p(s, d)$, showing the probability that the first time an address s bytes away will be referenced will be in exactly d cycles [306]. Such plots create various patterns: for example, sequential access to a range of

memory locations creates a diagonal ridge of high probabilities. However, it does not attach a numerical value that represents the degree of locality.

To obtain a numerical measurement that represents the degree of locality, we need to analyze the reference stream. The character of the analysis depends on exactly what it is we wish to measure.

Measuring Temporal Locality

The simplest measure of temporal locality looks at the reference stream through a pinhole. The idea is that if we have m possible locations, the probability of seeing any given location (assuming uniformity) is $\frac{1}{m}$. The probability of seeing two consecutive references to a given location is then (assuming independence) $\frac{1}{m^2}$. But this can happen for any of the m locations, so the probability of seeing two consecutive references to *some* location is $\frac{1}{m}$. This probability is easy to measure: just scan the stream of references, and count how many times two consecutive references are the same. If this is significantly more than $\frac{1}{m}$, the stream displays strong local correlation (i.e., locality of reference) [369].

A possible drawback of this process is that m can be very big, but most of these locations are seldom referenced. Therefore the test threshold of $\frac{1}{m}$ will be artificially low. This drawback can be countered by only considering the subset of locations that are the most popular, and thereby account for a large fraction of the total references.

Measuring Spatial Locality

The simplest way to demonstrate spatial locality is by counting unique substrings of the reference stream. The reference stream can be regarded as a very long string, over a very large alphabet: all the items that may be referenced (e.g. all the memory addresses). Denote the number of unique items by m. There are then m^2 possible combinations of two consecutive items, m^3 possible combinations of three, and so on. But if spatial locality is present, some combinations may appear quite often, whereas others do not. Counting the number of unique substrings of each length thereby provides an indication of spatial locality, especially when compared with the same count for a scrambled reference stream in which the same references are ordered randomly [23]. Significantly, this technique also works for items that do not have any natural ordering, such as web pages.

The problem with this approach is that it may be sensitive to the length of the trace — as more and more references are observed, there is a bigger chance of seeing even rare combinations. It is therefore desirable to limit the observations to a certain time window.

Practice Box: Scrambling Data

A simple in-place algorithm to scramble data is as follows. Assume we initially have a sequence of n items.

1. Loop on the items starting from the last one. Denote the index of the current item by i.

2. Select an index j in the range 1 to i uniformly and randomly.
3. Exchange the ith item with the jth item. In effect, this selects the jth item to be the ith item in the scrambled sequence (but note that the current jth item may have arrived at this location in some previous step).
4. Proceed to the next iteration of the loop in Step 1.

End Box

6.2.3 The Stack Distance and Temporal Locality

The most common way to quantify locality is not by statistical measures, but by its effect. We do so by means of a simulation. Given a sequence of requests, we run it through a simple simulation that is sensitive to locality, such as a simulation of caching. The simulation result then provides a metric for the degree of locality found in the sequence. To verify that the result is indeed due to locality, it is often compared to a result based on a scrambled version of the same sequence. In the scrambled sequence the requests are permuted, thus destroying any locality but preserving all the other attributes.

The most popular way to quantify temporal locality is by using a simulation of an LRU (least recently used) stack [475, 649, 648, 38, 271, 23]. The stack is maintained using a move-to-front discipline [66]: each item that is referenced is moved to the top of the stack, and its previous location (called its stack distance or reuse distance) is noted. If temporal locality exists, items will often be found near the top of the stack, and the average stack distance will be small. If there is no temporal locality, the average stack distance will be half the size of the stack, that is, linear in the size of the address space.

More precisely, the procedure is as follows:

1. Initially the stack is empty.
2. Loop on the reference stream. For each one,
   (a) If the referenced item is not in the stack, because this is the first time it is seen, place it on the top of the stack and proceed to the next reference.
   (b) If it has already been seen, find it in the stack and note its stack distance (i.e., how far it is from the top).
   (c) Move the item to the top of the stack.
3. The distribution of stack distances observed characterizes the degree of temporal locality.

To read more: The most time-intensive step in this procedure is finding each reference in the stack. The simplest approach is to perform a linear search. More efficient tree-based structures reduce the complexity from $O(n\,m)$ to $O(n \log m)$ [667, 22], where n is the length of the reference stream and m is the number of different addresses (note, however, that due to locality the cost of the simple algorithm is typically significantly lower than $n\,m$). Ding and Zhong give an approximate algorithm that reduces the time to $O(n \log \log m)$ and the space to $O(\log m)$ [182].

Note that the distribution of stack distances immediately tells us how well an LRU cache will perform on this reference stream. By definition, an LRU cache of size s holds

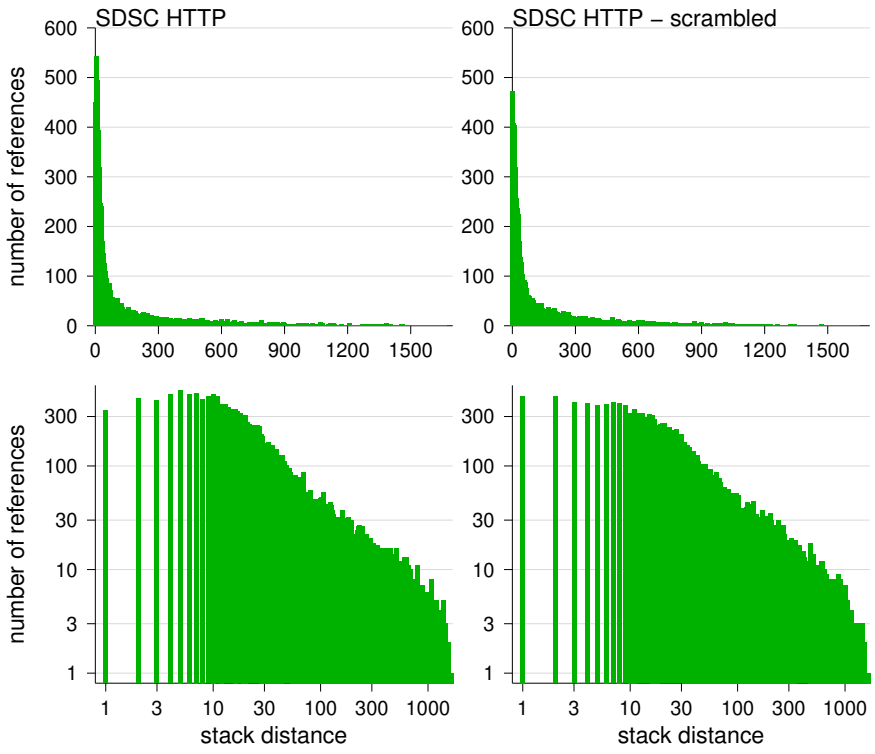
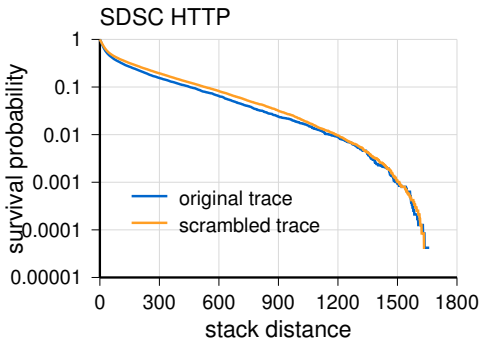


Figure 6.1: *Example of the stack distance distribution from a stream of requests to an HTTP server. The bottom plots present the same data in log-log axes.*

the s least recently referenced items. Thus references with a stack distance of up to s will be found in the cache, and those with a larger stack distance will cause a cache miss (to correctly count misses, insertions of new items at the top of the stack in step 2a should be counted as if they were found at distance ∞). Remarkably, a single simulation run generating the distribution of stack distances provides all the data needed to find the miss rate for any cache size s [475].

An example is shown in Figure 6.1, which displays the distribution of stack distances calculated for the SDSC HTTP trace available from the Internet Traffic Archive. This trace contained 25,430 successful requests to 1680 unique files, which were logged on 22 August 1995.

The figure shows a strong concentration at low values, indicating that small stack distances are much more common than large ones (top-left plot). However, the same qualitative effect remains if we scramble the reference stream, and calculate the stack distances for the same requests when they come in a random order (top-right plot) [381]. This is because of the highly skewed nature of the request stream. Some files are much more popular than others and have very many requests, so the previous request is never



Stream	*Average*	*Median*
original	146.7	40
scrambled	173.9	53

Figure 6.2: *Difference between the distributions of stack distances and their statistics, for the original and scrambled logs.*

far away. The effect of correlation between nearby references is actually the difference between the two plots.

To better see the difference, we plot the survival functions of the stack distances on a logarithmic scale (Figure 6.2). This shows that the main difference occurs for stack distances in the range from 50 to 1000. For example, the 90th percentile (0.1 probability of survival) of the original stack distance distribution is 453, and in the scrambled distribution it is 530. This means that in the original distribution the low values account for more of the total weight. This can also be seen by calculating the median and average stack distance. Interestingly, the tails of the original and scrambled distributions are essentially identical. This reflects those documents that are accessed very rarely in a random manner.

A possible way to eliminate the strong effect of popularity on the distribution of stack distances is to focus only on documents that are requested the same number of times [381]. However, this only utilizes a small fraction of the available data. It also raises the question of which part to focus on.

An additional set of examples is given in Figure 6.3, which shows the stack distance histograms from several SPEC 2000 benchmarks. The data comes from monitoring the execution of the benchmarks on Pentium III systems, and was available from the Brigham Young University Trace Distribution Center. Note that these graphs show the histogram on log-log axes, as in the bottom plots in Figure 6.1. They show that the distributions may have various shapes, and some may have a heavy tail. They may also have relatively sharp peaks like those in perl_makerand. Such peaks may indicate repeated access to sets of addresses in the same sequence.

An alternative to the distribution of stack distances is the distribution of inter-reference distances [22]. In other words, count how many other references occurred between two consecutive references to the item in question, rather than only counting *unique* references. This number is easier to calculate, and has the advantage that the results for each item are independent of how we deal with other items [267]. For example, considering these two reference streams again:

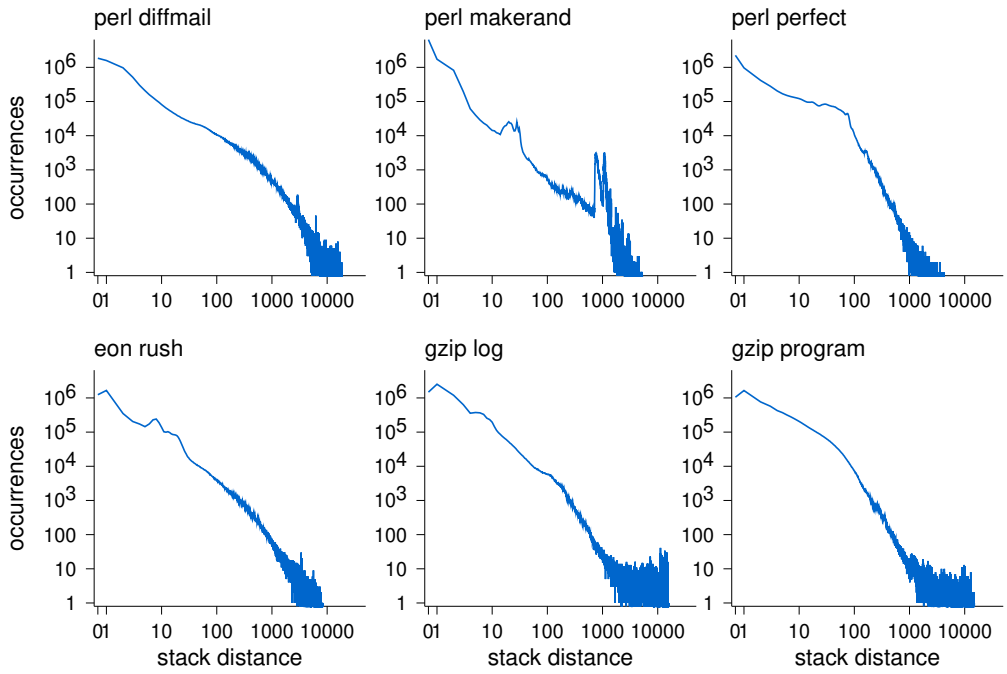


Figure 6.3: *Examples of stack distance distributions from SPEC 2000 benchmarks, using 4 KB pages.*

A B C D A C D B D A C B D C A B C D B A D C A B

and

A B A A A A A B A A A B A A A B A A B A A A A B

Note that B appears exactly at the same places in both of them. Therefore the inter-reference distances for B would be the same. However, the stack distances for B would be different in the two cases.

6.2.4 Working Sets and Spatial Locality

Spatial locality is measured by the size of the working set [471]. Intuitively, this counts how many different addresses are being referenced in the current phase of the computation.

The definition of a working set has two parameters: the time t and the time window τ, both of which are typically measured in machine cycles rather than in seconds. The working set $W(t, \tau)$ is then defined to be the set of addresses accessed in the interval $(t - \tau, t)$ [171]. Note the use of the notion of sets: if an address is accessed many times, it only appears once in the working set, and there is no requirement of order of the addresses. Given this definition, the size of the working set is a measure of locality: the smaller the working set (for a given time window), the stronger the locality.

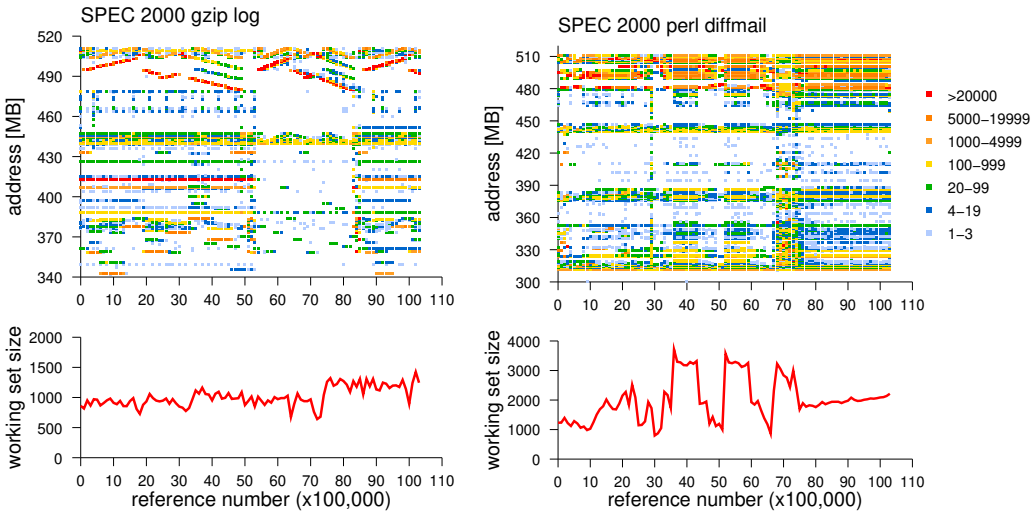


Figure 6.4: *Examples of memory accesses and working set sizes, from SPEC 2000 benchmarks.*

A couple of examples are shown in Figure 6.4. The data is again from the Brigham-Young repository of address traces of SPEC 2000 benchmarks. The top panels show access maps, in which each small square corresponds to 1 MB of address space in the vertical dimension, and 100,000 references in the horizontal dimension. In the bottom, the working set size is approximated by the number of unique 4 KB pages accessed within this window of 100,000 references.

Note that the definition of a working set includes all the addresses (or pages) accessed within a given window of time. But as we noted, some of these addresses are much more popular than others. It has therefore been suggested that one should focus on the "core" working set, defined to be those pages that are accessed repeatedly a large number of times. The size of this select group is, of course, smaller, and may be claimed to better reflect the locality of the reference stream [448, 216, 218].

A more elaborate approach is to simulate how a cache would handle the reference stream, and to count the cache misses. Simulating a cache that only caches single items can only capture temporal locality, and provides the base case. Simulating a cache with a cache line of several items captures spatial locality as well, and the reduced miss-rate can be used to quantify it. However, this has the disadvantage of depending on the parameters of the cache being simulated. Also, it can only be applied for ordered items, in which the neighborhood of each item is well defined.

6.2.5 Measuring Skewed Distributions and Popularity

As noted earlier, a large part of locality is often just the result of a skewed popularity distribution. It is therefore of interest to quantify how skewed is the popularity distribution.

The traditional metric for skewed popularity is the information-theoretic entropy. Assume a set A of m items (memory addresses or whatever), identified by an index i. Given a stream S of references to these items, we can find the fraction p_i of references to each item i. The empirical entropy of S is then defined to be

$$H(S) = -\sum_{i=1}^{m} p_i \log p_i \tag{6.1}$$

(note that $p_i < 1$, so $\log p_i < 0$, and the minus sign makes the entropy positive). The entropy is 0 when all the references are to only one of the items (i.e., the distribution is completely skewed). It is $\log m$ when the references are distributed uniformly (i.e., when $p_i = \frac{1}{m}$). This has led to the suggestion of using the *normalized* entropy, defined to be the entropy divided by $\log m$, making its value independent of the size of the set [266].

Measuring the normalized entropy of network addresses on the world wide web leads to a clear distinction between different types of nodes [266]. Clients have a relatively low entropy (around 0.65), indicating a rather skewed distribution: many requests are being sent to the same address in sequence. Proxies, in contrast, have a high entropy (in the range 0.8 to 0.9), indicating a much more uniform distribution. This is due to filtering out repeated requests, and to merging many different request streams. The entropy is reduced again (to around 0.75) the closer we get to the specific servers being accessed.

Another possible measure of skewed popularity is the Zipf exponent. Recall that, in the generalized Zipf distribution, when numbering the items according to their popularity, we have $p_i \propto \frac{1}{i^\theta}$ (Equation (3.40)). If the distribution is uniform rather than being skewed, $\theta = 0$. Conversely, the larger θ, the more skewed the distribution.

Considering the fact that the Zipf distribution is a special case of a heavy-tailed distribution, another option is to apply the metrics for describing mass-count disparity defined in Section 5.2.2. An example of doing so is shown in Figure 6.5. The joint ratio is found to be 17/83, meaning that 17% of the addresses account for 83% of the references, and vice versa. The less referenced 50% of the addresses account for only 4.2% of the references, whereas a full 50% of the references go to only 1.4% of the addresses. For comparison, the figure also shows the mass and count distributions that would be obtained if the same number of references were distributed uniformly among the same number of addresses. These are essentially identical normal distributions centered around the average number of references per item, which happens to be 15.1.

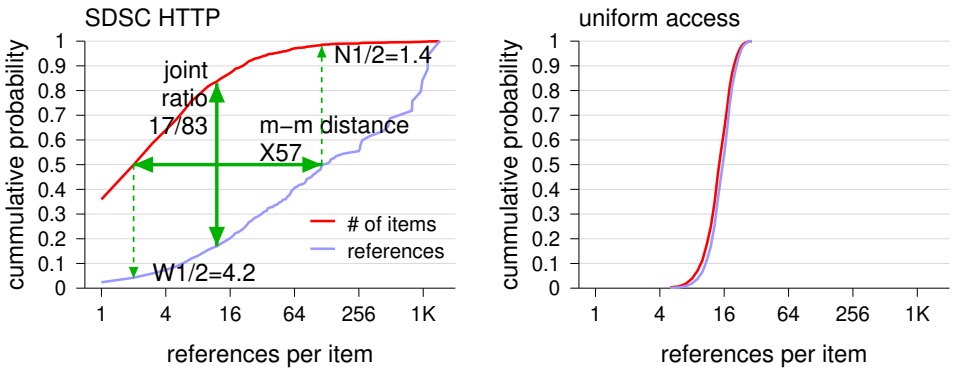


Figure 6.5: *Example of applying metrics for mass-count disparity to popularity data.*

6.2.6 Modeling Locality

Although the use of specialized metrics allows us to assess the degree of locality in a stream of references, this is not enough. In a model we want to be able to *generate* a stream of references that exhibits locality.

Independent Reference Model

The simplest way to generate a stream of references is to define a distribution that specifies the probability of referencing each address. Sampling from this distribution generates the desired stream. Note that the samples are done independently, meaning that one sample does not affect another. This is then called the independent reference model (IRM).

If the distribution is uniform, the generated reference stream will, of course, not display any locality. But as we noted earlier, distributions of popularity are generally highly skewed, and, in particular, follow a Zipf distribution [95, 57, 85]. When some addresses are much more popular than others, they will be selected much more often, leading to significant locality.

An example of the reference stream produced by such a model when using a Zipf distribution for address popularity is shown in Figure 6.6. Obviously most references are to the highly popular addresses, here drawn at the middle of the range. The degree of concentration actually depends on the exponent θ (see page 132). If we use a Zipf-like distribution with a lower exponent, for example $\theta = 0.5$, the references are spread out more evenly; the extreme case in which $\theta = 0$ is the uniform distribution.

In addition to the Zipf-like model of popularity, some other simple models are sometimes used too, especially in the context of networking. One such model is a combination of uniform traffic and hotspots. This means that most of the traffic is spread uniformly across all destinations, but a certain fraction is directed at one specific destination (or a small set of destinations) [547, 433, 17, 163, 546, 133]. A generalization of this model is based on mass-count disparity: for example, 80% of the traffic can be directed at 20%

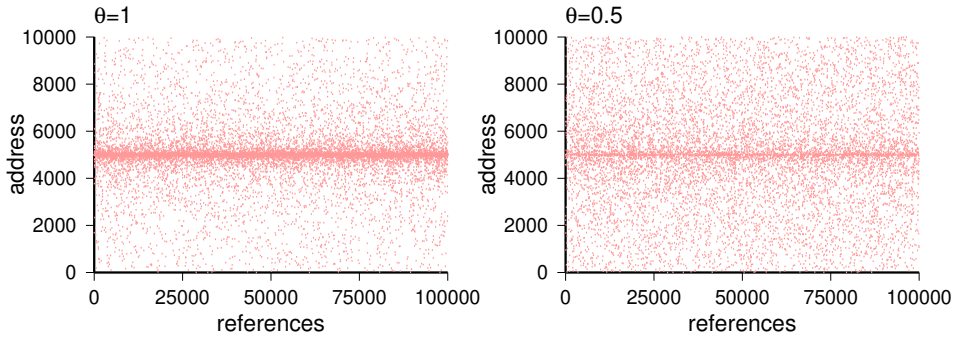


Figure 6.6: *Examples of an address trace generated by the independent reference model, for a Zipf distribution of address popularity. The most popular addresses are drawn in the middle for better visibility.*

of the destinations, while the other 20% of the traffic is distributed across the remaining 80% of the destinations [133]. Or, 90% of the I/O operations can refer to 10% of the files, while the remaining 10% of operations are spread across the remaining 90% of the files [578].

Performance evaluation studies reveal, however, that real reference streams often have more locality than that generated by the skewed popularity alone [23, 381]. This model is therefore not recommended in general.

LRU Stack Model

Simulating an LRU stack is not only useful for quantifying locality, as described in section 6.2.3. It can also be used as a generative model to create a reference stream with locality.

The basic idea is that a list of references and a list of stack distances are actually equivalent. We already saw one direction: given a reference stream, you can simulate an LRU stack and generate a list of stack distances. But it also works in the other direction: given a list of stack distances, we can generate the corresponding reference stream. This is done as follows.

1. Initially the stack contains all the addresses.
2. Loop on the stack distance list. For each one,
   (a) Find the address that appears at this distance into the stack, and use it as the next address to be referenced.
   (b) Move this address to the top of the stack.

To turn this process into a model, we just need to decide on which distribution of stack distances to use. We then select stack distances from this distribution, and use them to generate a reference stream. In order to have significant locality, small stack

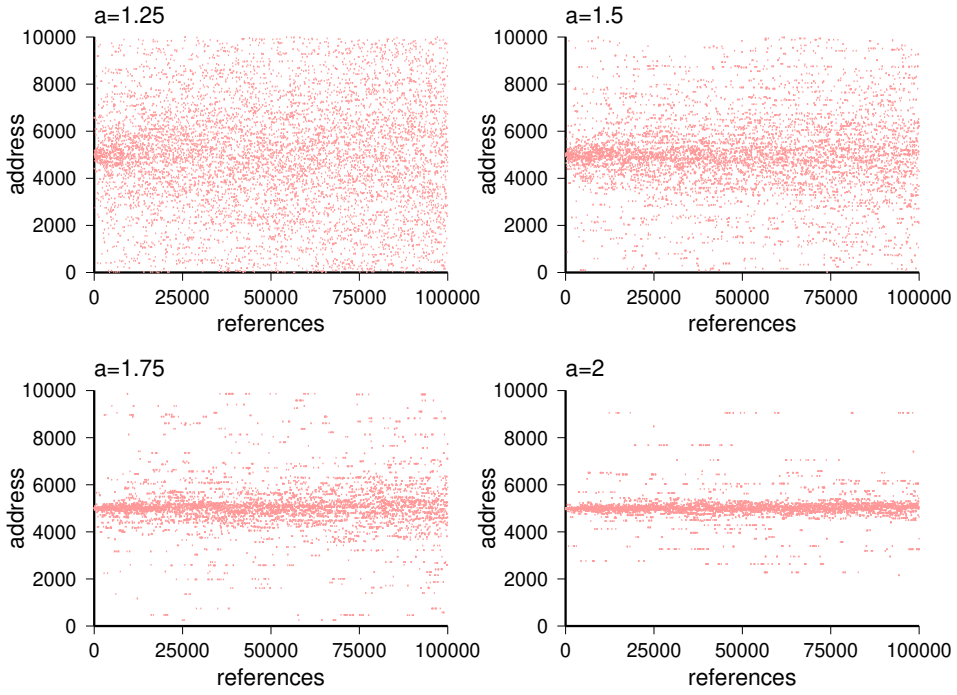


Figure 6.7: *Example of an address trace generated by the simple LRU stack model for a Pareto distribution of stack distances. The stack is initialized with addresses from the middle of the range at the top for better visibility.*

distances need to be much more common than big ones. Spirn suggests using a Pareto distribution with parameter $a = 2$ as an approximation [648].

Examples of reference streams generated by this model, for Pareto distributions with different values of the parameter a, are shown in Figure 6.7. When a is relatively low, the distribution of stack distances has a heavy tail. Therefore "deep" addresses have a non-negligible probability of being selected and moved up to the top of the stack. As a result there is relatively little locality. When a is large (e.g. $a = 2$), the tail is much lighter. In this case there is a much smaller probability of selecting an address from deep down in the stack, and the locality is largely retained at the addresses that were initially near the top. Use of this model comes with a problem of coverage: most of the address range is never visited. But coverage actually depends on the length of the reference stream: with a long enough stream, we will eventually visit more addresses and create a stronger mixing, losing the focus on the addresses that were initially on top.

Observing Figure 6.7, and especially the top plots where $a = 1.25$ or $a = 1.5$, might lead one to believe that such a reference stream displays a rapid mixing of addresses and a loss of locality. It is important to understand that this is in fact not the case. What we are observing is a continuous and gradual shift of locality, where occasionally a new address is added and others are pushed one step further out. The reason that we do not

see locality at the right-hand edge of the plot, after 100,000 references, is not that it does not exist, but that the addresses are not ordered in a manner that brings it out. If we were to order the addresses differently, such that the addresses that happen to be near the top at the end are grouped together, the plot would look as if locality was being generated out of a random mixture.

The stack distance model has two main deficiencies. One, which it shares with the IRM model, is the lack of structure in the reference stream. Real reference streams contain structures such as the serial traversal of large data structures, which are not generated by these random models. The other is that it leads to a sort of crawling phase transition as described above. In real workloads, application phase transitions cause a sharp shift in locality, in which many addresses become popular and many others become unpopular at the same time. Still, the LRU stack model is better than the independent reference model which has no shifts in locality at all.

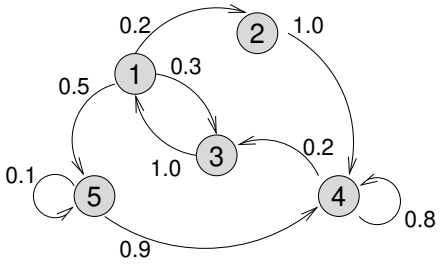
Markov Reference Model

A more realistic model is obtained by considering localities explicitly, and modeling the transitions from one locality to another. Thus as long as we are in the same phase, we want successive references to be to nearby addresses. But once in a blue moon we want to perform a transition to a different phase, with a different locality. This can be achieved by a Markovian model.

Background Box: Markov Chains

Markov chains are used to model stochastic processes, i.e. processes where things change dynamically with time.

At any moment, the process is in a certain *state*. The states are denoted by numbers: 1, 2, 3, and so on (this implies the assumption that they are enumerable). Given that the process is in state i, it has a certain probability $p_{i,j}$ to move to another state j. This can be represented graphically, with states as circles and arrows denoting possible transitions. Equivalently, it can be represented as a matrix of the $p_{i,j}$ values (called the transition matrix):



	1	2	3	4	5
1	0.0	0.2	0.3	0.0	0.5
2	0.0	0.0	0.0	1.0	0.0
3	1.0	0.0	0.0	0.0	0.0
4	0.0	0.0	0.2	0.8	0.0
5	0.0	0.0	0.0	0.9	0.1

The main characteristic of Markov processes is that these probabilities do not depend on history. Whenever the process is, say, in state 4, it has a probability of 0.2 to move to state 3, and a probability of 0.8 to stay in state 4. It does not matter whether it arrived at state 4 from state 2, or from state 5, or whether it has been continuously in state 4 for the last 13 steps.

The formal way to express the fact that knowing the history is immaterial is by using conditional probability. Saying that "there is a probability of 0.2 to move from state 4 to

state 3" is the same as saying that "the probability of being in state 3 at the next step, given that we are in state 4 now, is 0.2". This is written as

$$\Pr(s_{t+1} = 3 \mid s_t = 4) = 0.2$$

where s_t denotes the state at step t. To indicate that the history does not matter, we say that the probability given that we know the full history is the same as the probability given that we know only the last step:

$$\Pr(s_{t+1} = x \mid s_t = x_t, s_{t-1} = x_{t-1}, \ldots, s_0 = x_0) = \Pr(s_{t+1} = x \mid s_t = x_t)$$

This property is often interpreted as being "memoryless". Note, however, that we may actually know quite a bit about the process's history (e.g., if we are in state 2 the previous state must have been state 1). It is just that this knowledge is already encoded in the state, so knowing it explicitly does not add any information.

One of the most interesting properties of Markov chains is that they have a limiting distribution. This means that if the process goes on and on and on for a very long time, the probabilities of being in the different states will converge. Moreover, the limiting probabilities, denoted π_i, are stationary. This means that if we continue to apply transitions the probabilities do not change. They can therefore be found by solving the following equations:

$$\forall i : \pi_i = \sum_{j=1}^{n} \pi_j \, p_{j,i}$$

$$\sum_{i=1}^{n} \pi_i = 1$$

The first equation simply says that each step of the process maintains the same distribution, and the second says that it is indeed a distribution.

In truth, not all Markov chains have this property. The requirement for having a limiting distribution is that the Markov chain be ergodic. This means that there is a path from every state to every other state, and that states are not periodic (i.e., it is not true that they can only be visited on multiples of some number of steps).

The importance and consequences of ergodicity are subtle. Consider a given Markov chain, defined by its transition matrix. When we talk of the "probability of being in state i" we typically mean the average over many independent processes. This means that we may perform a large number of repetitions of the process, using different random number seeds, and then look at the probability of being in a certain state after, say, 100 steps. But if the process is ergodic, we will get the same result if we only look at one process, and calculate the probability of passing through state i as it evolves. Ergodicity means that averaging across independent processes at the same time step is the same as averaging on time steps within a single process.

To read more: Markov processes are covered in many probability texts. For example, good introductions are provided by Ross [581] and Trivedi [691].

End Box

The idea is to use a Markov chain in which the states correspond to memory addresses. The evolution of the Markov process generates a stream of references to the

corresponding addresses: being in state s_i generates a reference to address i. A transition from state s_i to state s_j corresponds to a reference to address j that follows a reference to address i. Having higher probabilities of moving to nearby states will lead to a reference stream that exhibits locality.

In particular, the Markov reference model typically uses what is known as a "nearly completely decomposable" transition matrix. This means that addresses are arranged in blocks [152, 172]. The transition probabilities within a block are high, which represents a phase of the computation in which we see repeated references to addresses in the same locality. But there are also low probabilities of moving to other blocks. If such a transition occurs, we become "trapped" in the new locality. Thus we will stop seeing references to the previous locality, and instead we will see many references to the new one. In this way Markovian models can capture different phases of computation, in contrast to the independent reference model and LRU model.

It is also relatively easy to construct a Markov chain that directly models a given address trace. Obviously, the states simply correspond to the addresses that appear in the trace. The transition probabilities are computed by counting how many times each address appears after each other address and normalizing. This will also lead naturally to the observed popularity distribution. The drawback of this model is that the state space is huge, because we need a state for each address. And the transition matrix is even bigger. This may be alleviated by modeling accesses to pages, rather than to individual addresses.

Although the Markov reference model is good at capturing the division of a program's execution into phases, it cannot model phases that have overlapping working sets. Another deficiency of a simple Markov model is that it implies that runlengths of the same item are geometrically distributed. Consider a state (representing an address) s, which has a self-loop with probability p_s. This means that the probability of seeing a single visit to s between visits to other states is $1 - p_s$. The probability of seeing exactly two consecutive visits to s is $(1 - p_s)p_s$, three consecutive visits $(1 - p_s)p_s^2$, and in general i consecutive visits $(1 - p_s)p_s^{i-1}$. Thus the probability of long runlengths drops off exponentially.

There are two simple ways to get around this deficiency. One method to obtain an arbitrary desired distribution instead of the exponential distribution is to use a hidden Markov model (HMM) [557]. This severs the direct link between visits to a state and references to the address represented by this state. Instead, a visit to a state generates a sequence of references that come from the desired distribution. When this approach is used, no self-loops are needed. The other method is to use a high-order Markov chain, in which the next state depends on the last k states, rather than only on the last state. For example, Phalke and Gopinath proposed a k-order Markov chain to model the inter-reference gap distribution for a given memory address [548].

Fractal Model

An intriguing approach that combines both spatial and temporal aspects of locality, and also characterizes it by a single number, is to measure the fractal dimension of the access

pattern [712, 683]. This is based on the postulate that the memory reference pattern of programs has a self-similar structure, i.e. often referenced addresses are grouped into clusters, within which there are sub-clusters, etc. A more detailed treatment of self-similarity will be presented in Chapter 7. For now, it is enough to know that it can be characterized by a single number.

More precisely, the model is that a reference stream with locality may be viewed as a fractal generated by a one-dimensional random walk. The "one dimensional" part is easy — the memory addresses are ordered from 0 to some maximal possible address, say m. A random walk is just what its name implies: start from some address r_1, and then move randomly to another, r_2, and so on. Recording the addresses visited by the random walk generates the desired reference stream: $r_1, r_2, r_3, \ldots$.

The crux of the model is how we choose the next address to visit. To get a self-similar fractal, the jumps from one address to the next must be scale invariant. This is achieved by using a (power-law) Pareto distribution [683, 684]. Thus most of the jumps will be small, leading to nearby addresses. But occasionally we will get a big jump and move to another part of the address space, and sometimes we will get really big jumps. How often this happens depends on the parameter a of the Pareto distribution. In general, the relevant range for a is $1 < a < 2$. The closer a is to 1, the larger the jumps, and the less locality we will see in the produced reference stream. The closer it is to 2, the more infrequent the large jumps become, and we will get a reference stream with significant locality. In effect, a is a parameter we can use to tune the desired degree of locality. An example is given in Figure 6.8.

The procedure to create a reference stream is as follows. Start with some arbitrary address, e.g. $r_0 = m/2$. Given that you have r_i, generate r_{i+1} as follows:

1. Create a Pareto random variable. As shown in Section 3.2.11, this is done by following these two steps:
   (a) Select a uniform random variate u from the range $[0, 1)$. This is what random number generators typically give.
   (b) Calculate the jump: $j = 1/u^a$. Successive js will then be Pareto distributed with parameter a.
2. With probability $\frac{1}{2}$, make j negative (so that jumps go both ways).
3. Multiply j by the desired unit. For example, if you only want word addresses, multiply by 4 (alternatively, it is possible to divide m by 4 and consider the whole range as composed of word addresses).
4. Calculate the next address: $r_{i+1} = r_i + j$.
5. Ensure that it is in the range $[0, m]$ by taking the modulo of $m + 1$.

Repeat this to generate as many references as desired.

As seen in Figure 6.8, the problem with this model is again one of coverage. When a is close to 2, the references display a high degree of locality. As a result they tend not to spread out across the address space. With the random number generator seed used

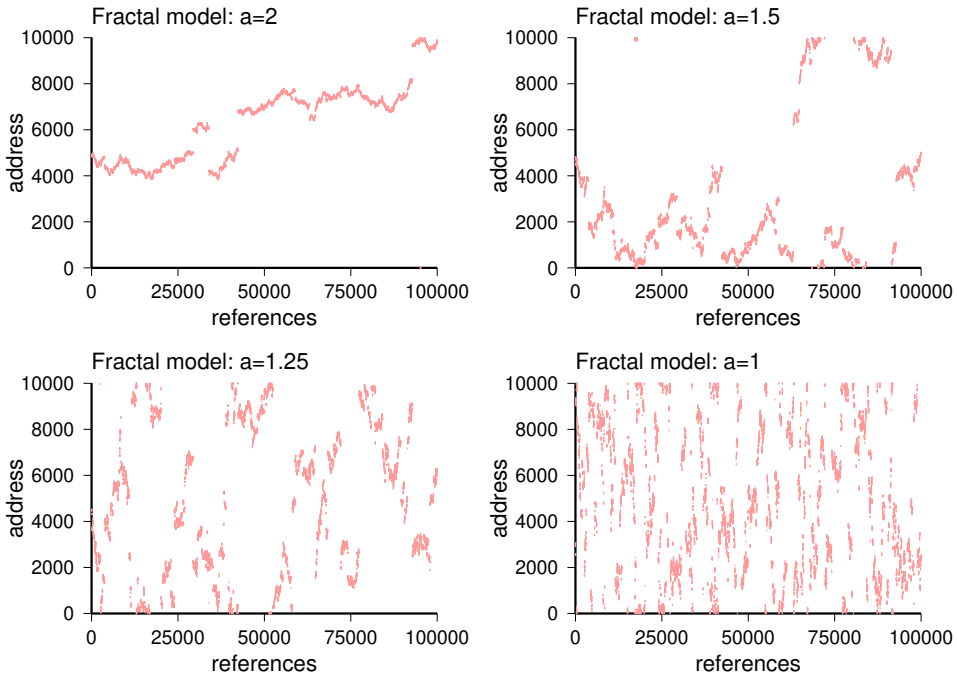


Figure 6.8: *Examples of address traces generated by the fractal model, for different values of the parameter a. The range of addresses is $M = 10,000$, and the unit is 2. Different random number generator seeds would produce different reference streams.*

in the example, this caused the range from 0 to about 3000 not to be visited even once in the first 100,000 references. When using the model, one should consider whether it is important to ensure that the different versions with the different degrees of locality indeed cover the same range. If it is important, rather long reference streams may be required.

The major deficiency of the fractal model is that it may violate the desired popularity distribution. In fact, on a very long run, the observed distribution of the number of times that different addresses have been referenced will tend to become uniform. Nevertheless, the model is able to predict the dependence of cache misses on the cache configuration under various circumstances [684, 631].

6.2.7 System Effects on Locality

It is important to realize that patterns of locality are not necessarily an intrinsic property of various workloads. Rather, they may be affected by the system. Care must therefore be taken to model the locality correctly at the source, and not to assume that the same levels of locality apply elsewhere.

A case in point is the difference in locality observed by client and server caches in a distributed file system [271]. The locality that is present in the original client request

streams allows for efficient LRU caching at the client caches. But the request streams from the clients to the server have much less locality, because many of the repetitions have been filtered out by the client caches. Moreover, the merging of multiple client streams that converge on the server destroys what little locality existed in each one of them. As a result LRU is not suitable for the server cache. LFU (least frequently used), which essentially caches more popular items, is better, because popularity survives better than locality. Similar effects occur in the world wide web, where the patterns in which caches are connected to each other are more complicated [572, 267, 266].

6.3 Locality of Sampling

Locality of sampling is a generalization of both temporal and spatial locality [234, 239]. It refers to the fact that workloads often display an internal structure: successive samples are not independent of each other, but rather tend to be similar to each other. This applies to all workload attributes, and not only to those that denote location, such as memory addresses.

This section demonstrates this effect and shows how to quantify and model it.

6.3.1 Examples and Visualization

Perhaps the simplest way to visualize locality of sampling is by plotting the sequence of values in question. For example, Figure 6.9 shows the sizes of files requested from two web servers, in the order that they were served. Due to the skewed distribution and the high variability a logarithmic scale is used, and the values are shown relative to the median: higher values are shown as upward pulses, and lower values as downward pulses. Thus sequences of requests for similarly sized files lead to wider blocks that are either above or below the median line. Such blocks are especially prevalent in the bottom log, from the University of Saskatchewan.

But a better approach is to actually plot the distribution as it is observed within limited time slices [190]. This is shown in Figure 6.10. The data is the distributions of job sizes on the LANL CM-5 in successive weeks. This machine used only 5 sizes, which were powers of two between 32 and 512 processors. Obviously the distribution of submitted jobs changes considerably from week to week, so they cannot all be similar to "the distribution of job sizes" — despite the fact that each week is represented by many hundreds of samples.

Looking at the distributions can be generalized by using scatterplots in which the X axis is time, and the Y axis is a workload attribute. Such a plot shows how the distribution of values of this attribute changes with time: a vertical slice of the plot shows the distribution at the time (position along the X axis) of this slice. This allows more data to be observed at once.

An example of such scatterplots is given in Figure 6.11, which shows the distributions of job runtimes and sizes on two parallel supercomputers, and how they change over a period of up to two years. Significantly, these plots use cleaned versions of the logs (without flurries). The concentrated blobs appearing in various locations testify that

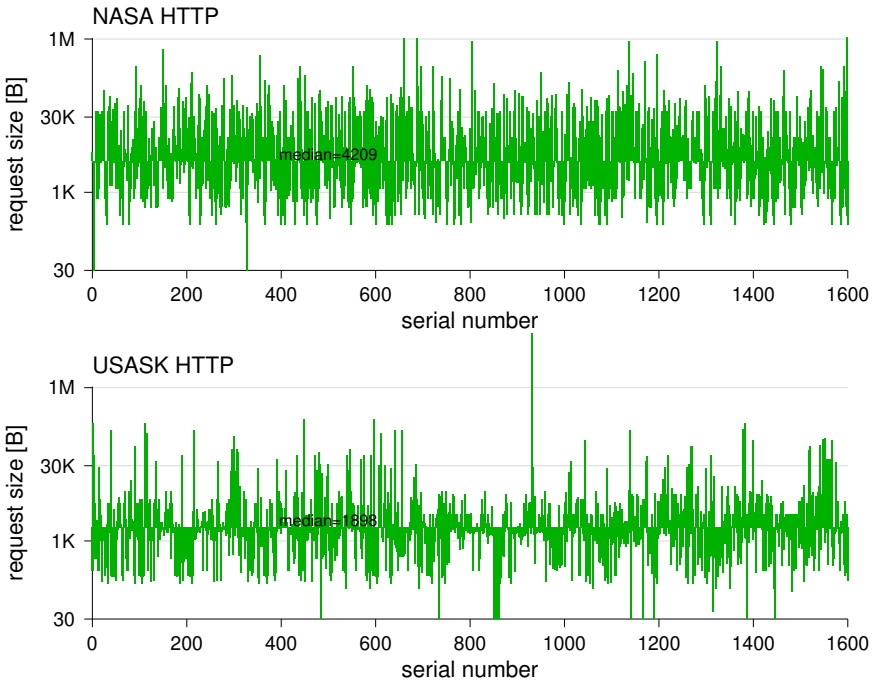


Figure 6.9: *The sequence of requested file sizes from two web servers. In both, requests for similarly sized files seem to be clustered together, but the effect is stronger in the bottom plot.*

a certain runtime or size value was very popular at that specific time. Horizontal streaks indicate that a value was popular throughout the duration of the observation; this is especially common for powers of two in the size distribution. However, enlarged blobs appear at certain times, as do gaps, which indicate that the value is absent at a certain time. The vertical streaks in the graphs show periods of especially intense load, but do not indicate anything about the distribution.

To better observe the concentration of certain values when sampling in a restricted span of time, one can compare the given data with a scrambled version. Figure 6.12 shows an example. The panels on the left show the original data, with some prominent concentrations of values (or gaps) marked with circles. The panels on the right are scrambled, or randomized. This means that some random permutation was applied to the jobs in the log (but keeping the original arrival times). As a result, jobs that used to be next to each other may now be distant, and jobs that originally were unrelated are now next to each other. The effect on the scatterplots is that the dependence on time is lost: at every time, we now observe a random sample from the global distribution. In particular, the concentrations of very many samples of related values are spread out uniformly in the horizontal dimension, and gaps in existing horizontal streaks are filled in. The fact that

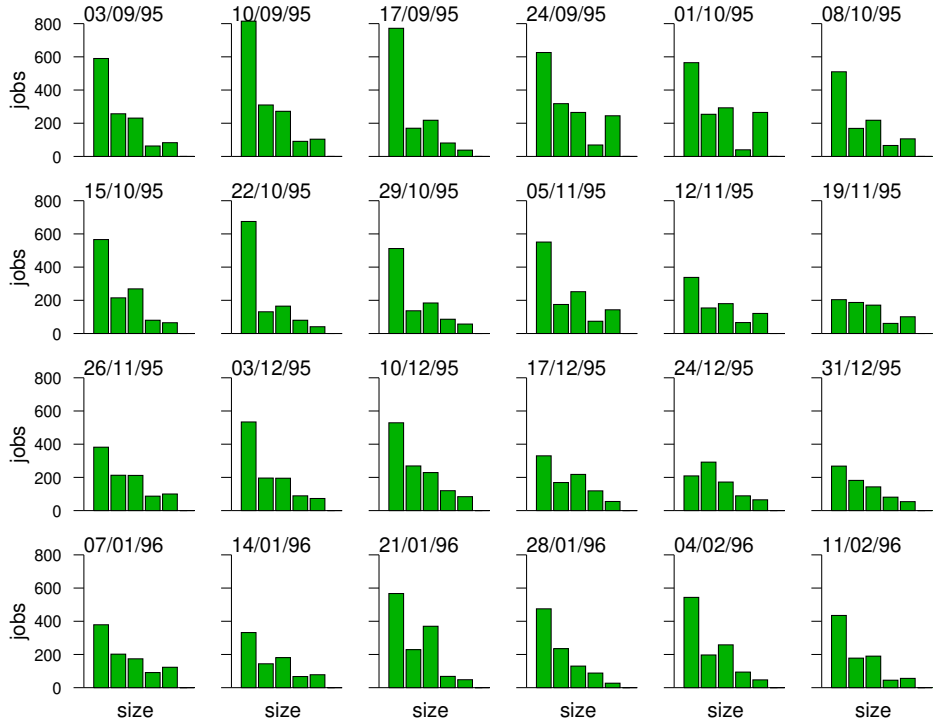


Figure 6.10: *Distributions of job sizes on the LANL CM-5 in successive weeks.*

this is different from the original plots testifies to the fact that the original ones exhibit locality of sampling.

While the above examples focus on the attributes of web requests and parallel jobs, locality of sampling exists in various other domains as well. For example, the instruction mix in different basic blocks of a program may differ [387]. This difference is important because basic blocks with an emphasis on a particular instruction type may not be able to use all the CPU's functional units effectively, thus reducing the achievable instruction-level parallelism.

Another example is the requests made of a memory allocator. In many cases the same sizes of memory blocks are requested over and again. This is especially common in applications written in object-oriented languages, in which new objects are created repeatedly at runtime. Such behavior has important implications for memory management. In particular, it makes sense to cache freed blocks in anticipation of additional requests for the same size [734].

A third example comes from scheduling a sequence of jobs. It has been shown that the relative performance of different scheduling schemes changes significantly for different levels of correlation between successive jobs [314]. Thus to ensure optimal performance the system should learn about the correlations in its workload, and select the appropriate scheduler to use.

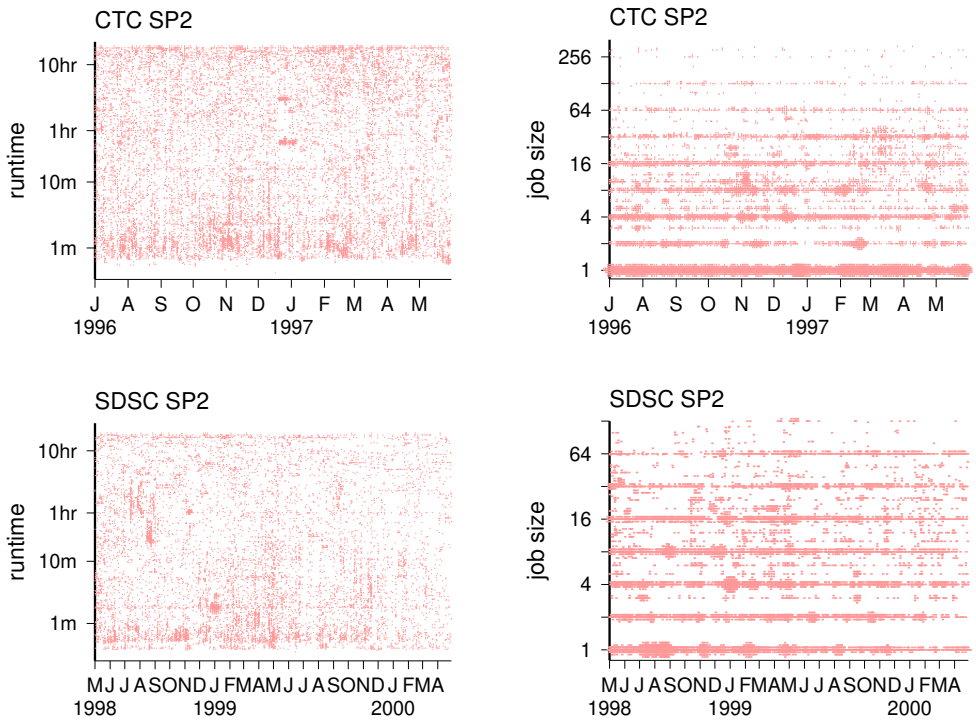


Figure 6.11: *Scatterplots with time as the X axis. It seems that at different times different parts of the distribution are emphasized.*

6.3.2 Quantification

The essence of locality of sampling is that, if we look at a short time scale, we see only part of the global distribution. The distribution at a short time scale is different from that at longer time scales: it is less diverse in the values that it contains, or in other words, it is more modal. Moreover, this modality is time dependent: at different times we observe different values.

The metrics for quantifying the degree of locality of sampling try to formalize these intuitions. This formalization has to be done with care, because, at extremely short time scales, a lack of diversity is expected: a single sample cannot represent the entire distribution.

Workload Diversity at Different Time Scales

Locality of sampling implies that during short intervals we do not observe all the possible different values. Note, however, that this is different from saying that the variance will be low: the values can still be far apart, leading to a high variance. The point is that there will be few distinct values, so that the *diversity* of values will be low.

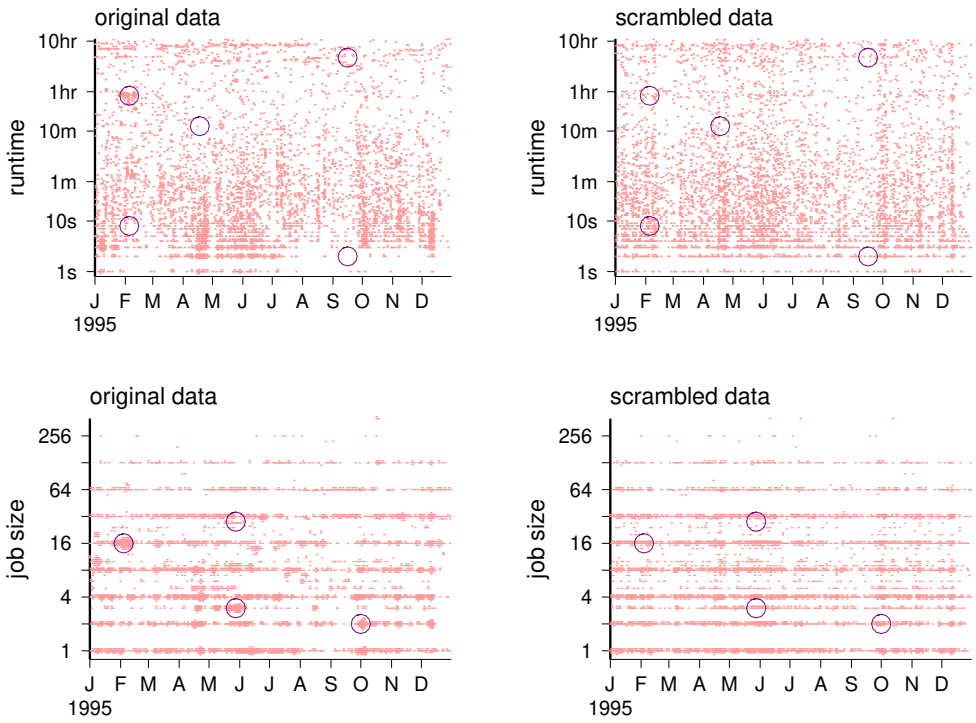


Figure 6.12: *Verification of the existence of locality of sampling, by comparing the original data with scrambled data after a random permutation along the time axis. Workloads typically do not look like a random sampling from a global distribution. Data from the SDSC Paragon.*

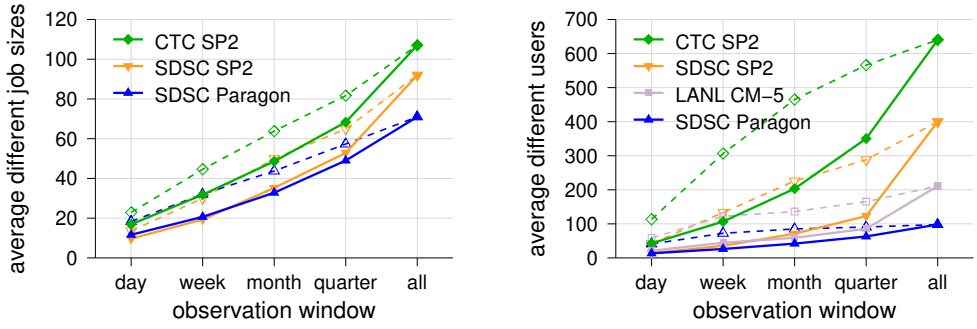


Figure 6.13: *Examples of workload diversity as a function of the observation interval. With longer intervals, more values are observed. Dashed lines show the same data when the log is scrambled, and locality is destroyed. In both cases, weekends are excluded for the "day" data point.*

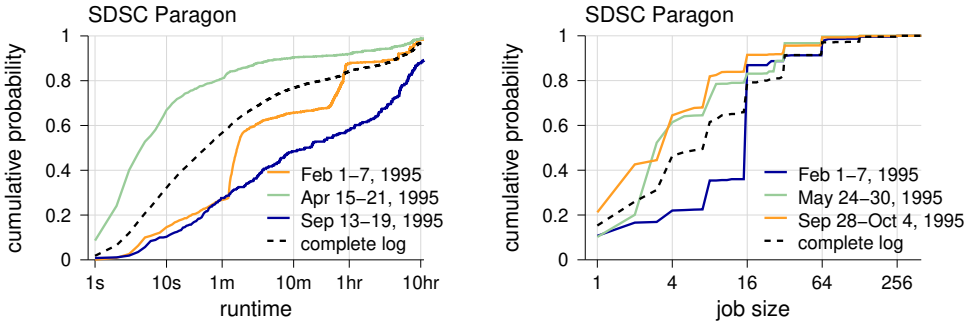


Figure 6.14: *The distributions of job runtimes and sizes on select weeks tend to be modal and different from each other and from the distribution of the entire log. Selected weeks correspond to markings in Figure 6.12.*

In the case of discrete distributions with few possible values, the diversity is measured by the number of different values observed in a certain interval of time — similar to the size of the working set of a process, which is the number of distinct pages accessed [171]. This is demonstrated in Figure 6.13 for the average number of job sizes and the average number of distinct users seen in a time interval, as a function of the length of the interval. In short intervals the number of distinct values is much smaller than in longer intervals or in the entire log.

But, of course, with small intervals we also expect to see less distinct values even if there is no locality of sampling. To check that the results are indeed significant, we therefore repeat the measurement on a scrambled log. The scrambled log contains exactly the same jobs and the same arrival times, but the association of a specific job to a specific arrival time has been permuted randomly. The results, shown with dashed lines, is that the number of distinct values still depends on the observation interval. However, it is always much higher than the number observed in the original log. Thus the original log has much less diversity at short time scales, and much more locality.

Slice vs. Global Distributions

For continuous distributions we need to come up with other metrics for diversity. The inspiration comes from data such as shown in Figure 6.10. We start by dividing the timeline into equal-duration slices, and find the distribution of workload items when considering each slice independently. We call these *slice distributions* because they are limited to a slice of time. The metrics are based on direct measurement of the difference between the slice distributions and the global distribution.

Figure 6.14 shows distributions for three selected weeks of the SDSC Paragon log. Because of the locality of sampling, these distributions tend to be different from each other, different from the global distribution, and also much more modal than the global distribution, as reflected by their more steplike shape. For example, the data for February

1–7 indicates a preponderance of 16-node jobs, running for either a couple of minutes or about one hour.

Based on this observation, we can propose an actual measure of the divergence of the weekly (or other short-range) distributions from the global distribution. This uses a combination of the χ^2 test described in Section 4.5.4 and the Kolmogorov-Smirnov test from Section 4.5.2. The difference is that here we use the tests in reverse: we want to show that the distributions are different from each other, and then quantify how different they are.

The χ^2 test divides the range of possible values into subranges with equal probabilities, and verifies that the number of samples observed in each are indeed nearly equal. To quantify locality we modify this to divide the overall range into subranges that have equal probabilities *according to the global distribution*, and observe the maximal probability for a single range *according to the slice distributions*. Doing so measures a mode that is present in the slice distributions, but not in the global one. We use the maximal deviation, as in the Kolmogorov-Smirnov test, because we are interested in the deviation between the global and slice distributions.

In order to be meaningful, there should be at least a few samples in each subrange. This places constraints on how the measurement is done. Assume the whole log contains a total of n samples (e.g. parallel jobs). If the length of the log is d days, there are n/d jobs per day on average. This number has to be large enough to apply the χ^2 test using enough subranges. If it is too small, we need to consider a larger basic time unit. Using the 1995 SDSC Paragon log as an example, it contains 53,970 jobs (in the cleaned version) and spans a full year, for an average of 147.9 jobs per day. This should be enough for a resolution of more than 20 subranges. However, due to fluctuations in activity, there will be much fewer jobs on some days. It may therefore be better to use a somewhat longer time unit, say three days.

Given that we have selected a time unit t and a resolution r, the calculation proceeds as follows [239].

1. Create a histogram of the complete (global) data, and partition it into r equally likely ranges. This defines the boundary points of the ranges.
2. Partition the log into d/t successive slices of t time each (this need not be measured in days as in the previous example — the time unit should match the type of data being considered).
3. For each of these slices of the log (indexed by i), do the following:
   (a) Find the number of workload items n_i in this slice of the log.
   (b) Create the slice histogram of these n_i workload items, and count how many of them fall into each of the r ranges defined in step 1. Denote these counts by $o_1, \ldots, o_r$.
   (c) By construction, the expected number of items in each range (assuming the global distribution) is $e_i = n_i/r$. We are interested in the deviations from this, and, in particular, in the maximal relative deviation. Therefore we compute

Log	M'	M
LANL CM-5	0.059	0.039
SDSC Paragon	0.098	0.073
CTC SP2	0.063	0.046
KTH SP2	0.085	0.056
SDSC SP2	0.101	0.072
Blue Horizon	0.069	0.052
DataStar	0.085	0.068

Table 6.1: *Results of measuring the degree of locality of sampling for the runtime distributions in different logs.*

$$m_i = \frac{\max_{j=1..r} \left\{ |o_j - e_i| \right\}}{n_i - e_i}$$

This is slightly different from the conventional expression used in the χ^2 test. First, we use the max, rather than a sum, to emphasize the concentration of values in a subrange. Second, we use the absolute value rather than the square to ensure that the result is positive, thereby avoiding the distortion that results from squaring. Finally, we divide by $n_i - e_i$ rather than by e_i. This normalizes the result to the range $[0, 1]$, because the maximal value for any o_j is n_i, which occurs if *all* the samples appear in the jth subrange.

4. Given all the m_i, calculate the final metric as their median:

$$M' = m_{(d/2t)}$$

The maximum could also be used, but it is more sensitive to details of the measurement, such as the precise duration of the log or how it is divided into slices [239].

Applying this procedure to the SDSC Paragon log, when using 24 subranges and slices of three days, yields the following results. The observed range of results is from 0.031 to 0.470. This means that, in one of the slices, nearly half of the jobs were concentrated in a single subrange, rather than being equally dispersed among all 24 subranges. The median is 0.098. Additional results are given in Table 6.1.

But are these results significant? Obviously, even if the slice distributions are identical to the global one, some deviations are to be expected in a random sampling. We therefore need to validate our result by comparing it to one that would be obtained via random sampling. This is done using the bootstrap method [205, 206, 178, 212].

The bootstrap method is very simple: we just repeat the measurement using a random sampling from the global distribution a large number of times (say a thousand), and find the distribution of the results. In our case, each experiment creates n samples from the global distribution, partitions them into slices with sizes dictated by the n_is, creates their histograms, and finds the median of the resulting m_is. Each repetition thus produces a single data point (the value of M') valid for a specific sampling from the global

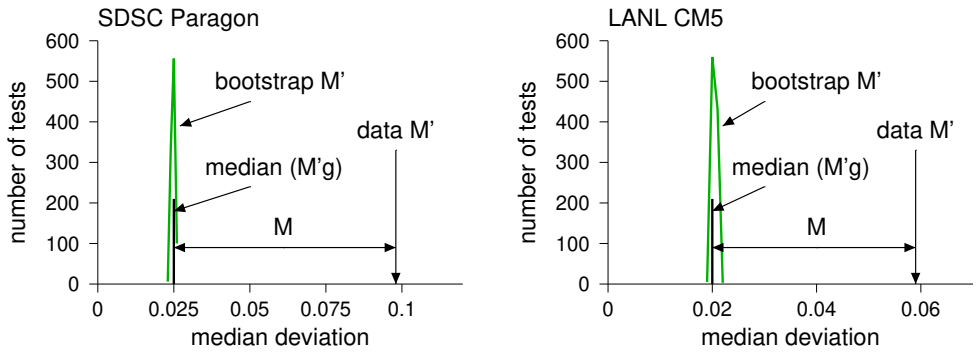


Figure 6.15: *Results of 1000 random tests of the median deviation observed when samples come from the global distribution, compared to the median deviation observed in the slice distributions.*

distribution. Repeating this a thousand times allows us to approximate the distribution of such results, that is, the distribution of M' for random sampling. We then check where the result for the *real* slice distributions falls in this distribution of results. If it is at the extreme end of the range, it is unlikely to have occurred by chance. The outcome of following this procedure is shown in Figure 6.15. Obviously, the actual result for the log is way out of the scale of results that are obtained for random sampling from the global distribution. We can therefore claim that it is significant.

Note, however, that the absolute value of M' as computed above is not of interest. Rather, locality of sampling is measured by the difference between M' and the value that would be obtained by chance. We define the latter simply as the median of the distribution of results obtained by the bootstrap method. Denote this median value by M'_g. Our final metric for locality of sampling is therefore

$$M = M' - M'_g$$

This is demonstrated in Figure 6.15 and tabulated in Table 6.1.

Locality of Sampling and Autocorrelation

An alternative, much simpler methodology to measure locality of sampling is by using autocorrelation. If workload items are independent, there will be no correlation between them. But if we often see repetitions of the same type of work, the successive items will appear correlated to each other.

Correlation and autocorrelation are explained in detail in subsequent sections of this chapter. The idea is to check whether items deviate from the mean in a similar way. For example, if successive items tend to be either both above the mean or both below the mean, but not one above and one below, then they are correlated. Technically, this illustrates an autocorrelation at a lag of 1, because we are looking at successive items. Larger lags refer to correlations between items that are more distant from each other.

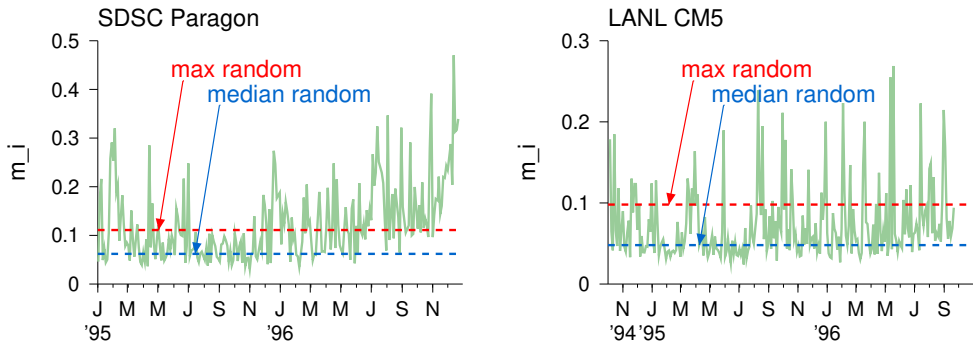


Figure 6.16: *Fluctuations in m_i across the duration of the logs indicate a lack of uniformity.*

Although locality of sampling and autocorrelation are closely related, they are not in fact equivalent. It is possible to construct sequences that exhibit significant locality of sampling as quantified earlier, but still have zero autocorrelation [239].

6.3.3 Properties

Locality of sampling actually has four distinct properties: depth, focus, length, and uniformity.

The *depth* is a measure of how different the slice distributions are from the global ones. It is the maximal value M that can be computed by the procedure outlined above, using different (reasonable) choices of t and r. Depth relates to the effect that locality of sampling may have on performance evaluation results: the bigger the depth, the bigger the potential impact.

The *focus* is the degree to which the slice distributions are focused into a narrow range of values; in other words, it is a measure of how modal they are. This can be measured by the number of subranges that produce the maximal depth. If the maximum occurs for a small r, the distributions are actually rather dispersed. If it occurs for a large r, they are focused. Focus is related to the degree of predictability that can be expected from workloads during short intervals.

The *length* is the span during which the slice distributions stay different; it corresponds to the time unit t in the earlier derivation. Specifically, the length is the t for which the metric is maximized. Length relates to the characteristic time scale affected by locality of sampling. This time scale influences the degree to which we can exploit the existence of such locality to make predictions about the future workload.

The *uniformity* is the degree to which different spans display the same locality of sampling. This corresponds to the distribution of m_is in the above derivation. If they all have similar values, the data is considered uniform. But if high values only occur in a small subset of ranges, and at other times the slice distribution is similar to the global one, the data is non-uniform (Figure 6.16). Uniformity can be measured by the distri-

bution of m_i values measured for disjoint spans that all have the characteristic length. It relates to the stability of performance evaluation results: if the workload is highly non-uniform, there is a greater danger that different results would be obtained if we use different parts of the workload data.

6.3.4 Importance

As noted earlier, the commonly used approach to the generation of synthetic workloads is to sample from a distribution [427, 367]. In fact, such sampling is also implied in mathematical analyses (e.g., using queueing networks) that accept the workload distribution as an input parameter. But real workloads tend to have an internal structure, in which job submittal patterns are not random [258]. One type of pattern is the repetitiveness that results from locality of sampling: on short time scales workloads have much less diversity than on long time scales, leading to a much more modal distribution. It is the conjugation of many such modal distributions that leads to the more continuous distributions seen on long time scales.

This effect is potentially very important because it means that, on a short time scale, workloads are relatively regular and predictable. This justifies the common assumption that recent behavior is indicative of future behavior, and can be exploited to our advantage. For example, we may be able to make predictions about resource utilization and availability, as is done in the Network Weather Service [739], and when predicting queueing times [87], job runtimes [695], and user clicks on web search results [551].

Moreover, real systems operate in an online manner, and must contend with the workload as it appears on a short time scale. Thus a model based on random sampling from a global distribution subjects the system to a very different workload than one based on localized sampling from the same distribution. In terms of system behavior it is possible to envision schedulers that exploit the short-range regularity of localized workloads, and adapt their behavior to best suit the current workload [243, 756, 669, 757, 695]. This is similar to recent approaches used in memory allocators, which cache freed blocks in anticipation of future requests for the same block sizes [734].

On the flip side, it seems that repetitive workloads result in performance degradations [439], and also cause performance evaluations to be more sensitive to small variations in conditions [696]. Thus workloads with significant locality of sampling may be harder to handle, and require more work to get reliable results.

An example of why this may happen is shown in Figure 6.17, which compares the moving average of runtimes of parallel jobs generated by two workload models. The Feitelson model creates locality of sampling, as explained later. In the Jann model, jobs are independent. The results are that in the Feitelson model there is higher variability, and one occasionally sees large fluctuations that are the result of a burst of many very long jobs. Such large fluctuations in the workload are obviously hard to handle by a scheduler, and also affect the stability of the results. In a model where jobs are independent, in contrast, the probability of a sequence of extraordinary jobs is negligible, and effects tend to cancel out.

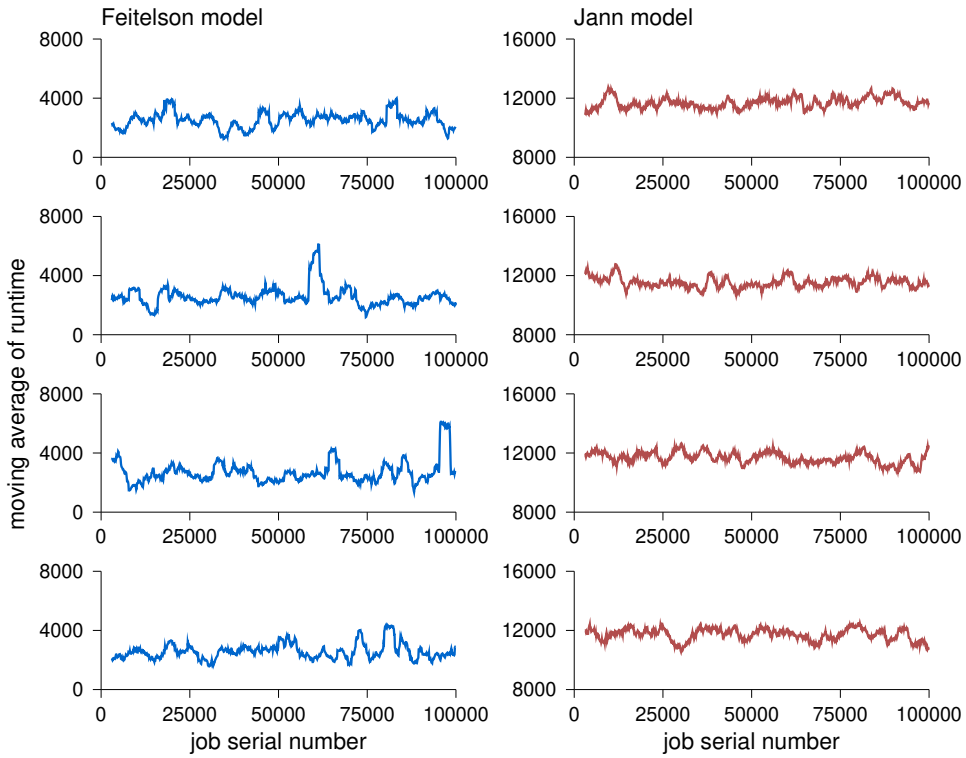


Figure 6.17: *Moving average of runtime samples from the Feitelson model, which has locality of sampling, and the Jann model, which does not, for different random number generator seeds.*

6.3.5 Modeling

A simple way to generate data that displays locality of sampling is to use repetitions. Initially, we simply sample from the global distribution. But then, instead of using each variate once, we repeat it a number of times. With enough repetitions we will get a sequence of samples that has a modal slice distribution.

The justification for this approach comes from the original workloads. Analyzing repetitions in workload logs leads to results such as those shown in Figure 6.18. In this analysis, we scan the workload data and partition it into separate streams of jobs submitted by different users. We then look for runs of equivalent jobs, defined to be jobs that execute the same application and use the same number of nodes. The distribution of runlengths shows that many jobs are independent or are part of a short run, but some runs are very long. Plotting the data on log-log axes suggests a Zipf-like distribution, with a power-law tail.

This intuition may be formalized as follows [230, 239]. We are given a global distribution described by the pdf $f(x)$. In addition, we need the distribution of repetitions, which we denote by $f_{rep}(r)$. The procedure can then be described as follows:

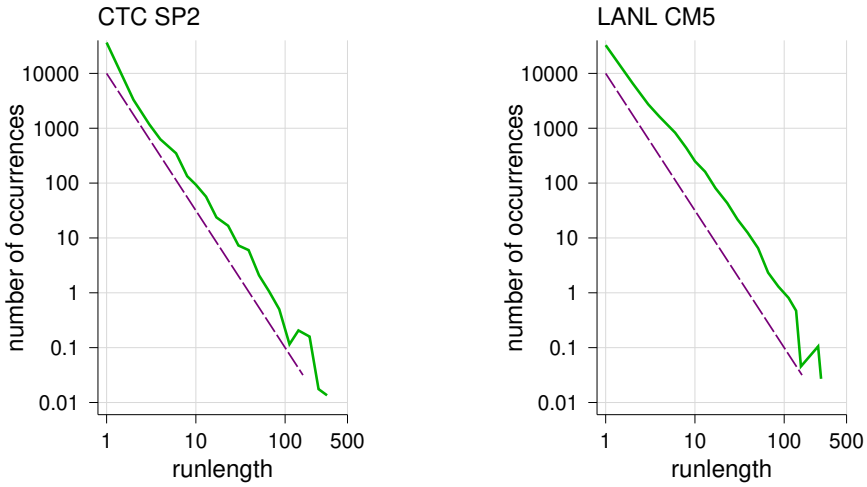


Figure 6.18: *Histograms of runlengths of similar jobs in production workloads from parallel supercomputers. Note the use of logarithmic axes; the dashed line used for reference has a slope of −2.5.*

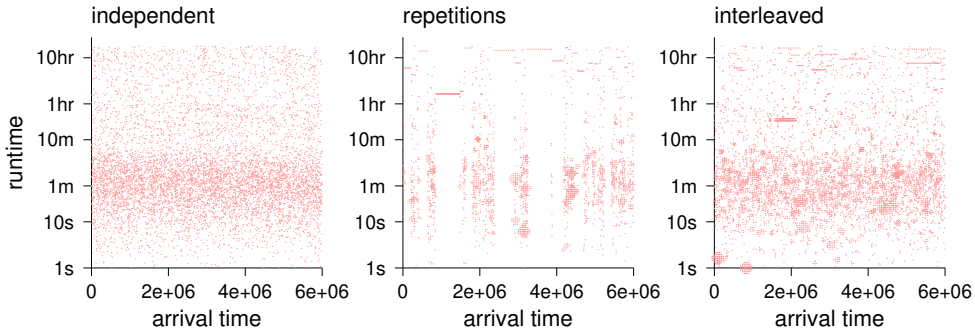


Figure 6.19: *Modeling locality of sampling by repeating workload items.*

1. Select a sample X from the distribution $f(x)$.
2. Select a repetition factor R from the distribution $f_{rep}(r)$.
3. Repeat the X variable R times. This distorts the distribution locally.
4. Return to step 1 until the desired number of samples have been generated.

With a large enough number of samples, the number of times we will see a value of x will be proportional to $f(x)$ (i.e., according to the global distribution, as desired). But these samples will come in bursts rather than being distributed evenly (Figure 6.19).

As a concrete example, consider modeling the arrivals of jobs with a certain distribution of runtimes. Using an independent sampling model leads to results such as those

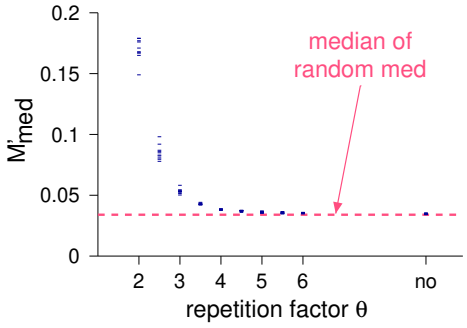


Figure 6.20: *The measured locality of sampling as a function of the parameter θ of the distribution of repetitions.*

shown in the left of Figure 6.19: a smear of runtimes that matches the distribution. By using repetitions we get the results shown in the middle. In this case the runlengths were taken from a Zipf-like distribution with parameter $\theta = 2.5$, chosen according to the data in Figure 6.18. This means that the probability of a runlength of r is proportional to $r^{-2.5}$.

To generate a more realistic workload, it is necessary to incorporate time and interleave the different sequences. Assume that each workload item is associated with an arrival time, and has some duration. We then sample items as above, assigning them arrival times. But the repetitions of each item are assigned arrival times that are staggered according to each one's duration. Assuming the durations are large enough relative to the interarrival times, the result will be a more realistic interleaved workload (right-most graph in Figure 6.19). It is also possible to add think times between the repetitions, thereby modeling the case of manual repetitions, rather than repetitions generated by a script.

Even further realism is obtained if the repetitions are not exact replicas of each other. For example, Li et al. suggest that job attributes be represented by multivariate Gaussian distributions, based on clustering the original job data. The top level of the sampling (step 1) then selects the cluster to use, and the bottom level (step 3) creates multiple samples from the respective distribution [440].

A nice feature of this modeling technique is that it affords control over the degree of locality of sampling in the generated workload. The locality results from the repetitions, which in turn come from a Zipf-like distribution with parameter θ. By modifying this parameter we can change the distribution of the lengths of repeated sequences[1]. This is illustrated in Figure 6.20, in which small θ lead to very long runlengths, whereas larger θ cause the distribution of runlengths to decay quickly, producing few repetitions if any.

Modeling locality of sampling by using job repetitions as suggested here has two important advantages: it is parsimonious, and it is generative.

Sampling with repetitions is as simple as a model can be, because it only requires

[1]This actually has a similar effect to the samples permutation procedure suggested by Li et al. [440].

the distribution of repetitions, which is described by a single parameter — the slope of the histogram (Figure 6.18). Other models for locality are typically more complex. For example, Shi et al. find that the best model for a distribution of stack distances is a mixture of a Weibull distribution and a Pareto distribution, so five parameters are needed. Locality of sampling can also be achieved by a user behavior graph [258] or a hidden Markov model (HMM) [557, 643]. However, this complicates the model because it requires a description of the complete dynamics and what workload items correspond to each state. For example, when using an HMM we need to define the transition matrix among the states, as well as the output distribution for each state; the number of required parameters is at least linear in the number of states. Sampling with repetitions is much simpler, albeit this simplicity may come at the price of not capturing potential non-repetition sequencing properties.

The fact that the model is generative is even more important than parsimony. The alternative to a generative model is a descriptive one, which just describes a certain situation, without explaining its mechanics. Thus descriptive models do not provide any clues about how the model should change under different conditions. For example, consider what may happen when the load on a system changes. If a (descriptive) stack model is used, the same stack depth distribution would be used for all load conditions. But a repetitions-based generative model shows that this is probably wrong. When the load is extremely low, there is little if any overlap between repeated sequences of jobs, so stack distances should be very small. But when the load is high, more other jobs intervene between repetitions, leading to higher stack distances. With a generative model we only need to create more sequences to increase the load, and the modification of the locality follows automatically.

A more sophisticated approach is to use a hierarchical workload model: the top level selects what part of the distribution to sample now, and the lower level does the actual sampling from the designated region. For example, this can be based on modeling the behavior of the user population. Such models are discussed in Chapter 8.

6.4 Cross-Correlation

Locality means that successive samples of the same random variable are correlated. Here we deal with another situation, in which samples of distinct random variables are correlated.

6.4.1 Joint Distributions and Scatterplots

It is not always obvious whether a correlation exists between two variables. We therefore start by considering ways to describe the joint behavior of two variables.

Formally, the way to describe the relationship between two or more variables is to use their joint distribution. The joint distribution specifies the probability of seeing various *combinations of values*, just as each variable's distribution specifies the probability of seeing various values of that variable in isolation.

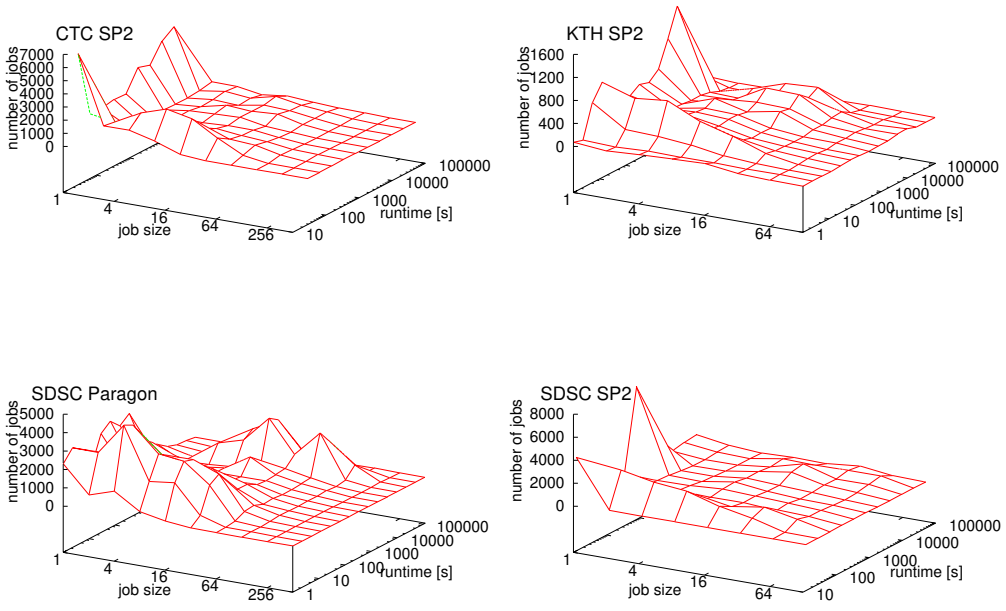


Figure 6.21: *Joint distributions of job size and runtime from different parallel supercomputers.*

All the issues discussed in Section 3.1.1 regarding distributions of a single variable are directly applicable to joint distributions as well. Assuming the variables are called X and Y, their joint CDF is

$$F(x,y) = \Pr(X \leq x, Y \leq y) \tag{6.2}$$

The joint pdf is the double derivative of the joint CDF

$$f(x,y) = \frac{\partial^2 F(x,y)}{\partial x \, \partial y} \tag{6.3}$$

or, in the case of a discrete distribution, the probability that they are assigned specific values:

$$p(x,y) = \Pr(X = x, Y = y)$$

In short, these are 2D functions, and can be visualized as a 2D surface. An example is given in Figure 6.21, which shows the joint distribution of the size (number of processors) and runtime of parallel jobs. Combinations that occur more commonly cause peaks in the distribution. This can of course be generalized to higher dimensions, but that is harder to visualize.

An important concept related to joint distributions is the marginal distribution. This is the distribution of each of the variables by itself. The idea is that we can use the joint

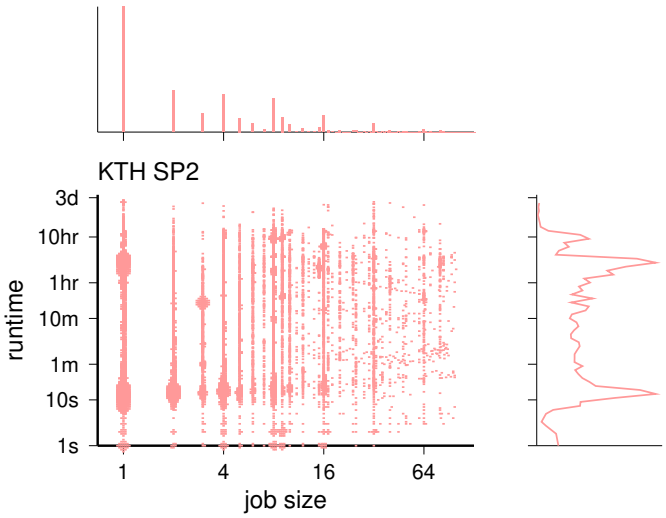


Figure 6.22: *Example of a scatterplot and the related marginal distributions.*

distribution of X and Y to find the distribution of, say, X, by summing over all possible values of Y. Specifically, the probability that X will have a value of x is

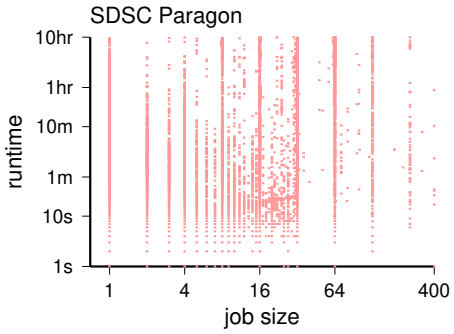
$$p_X(x) = \sum_y p(X = x, Y = y) \tag{6.4}$$

Recall that X and Y form the axes of a 2D area. Summing over all values of Y, for a given value of X, means that we are summing along a line parallel to the Y axis. This gives one value. Repeating this for all values of X yields the entire distribution of X. In effect, we are summing on the whole area and producing the results along the X axis, on the margin of the summation area. Hence the name "marginal distribution".

Given sampled data, one can represent the joint distribution using a 2D histogram. But a more common alternative is to use a scatterplot. In scatterplots the axes are the possible values of X and Y, as in a joint distribution. Each sample is represented by a dot drawn at the coordinates of the sample's X and Y values. This allows for a very detailed observation of the data. The marginal distributions are then simply the 1D histograms of the values assumed by X and Y (Figure 6.22). If the distributions are skewed, the axes can be logarithmically scaled.

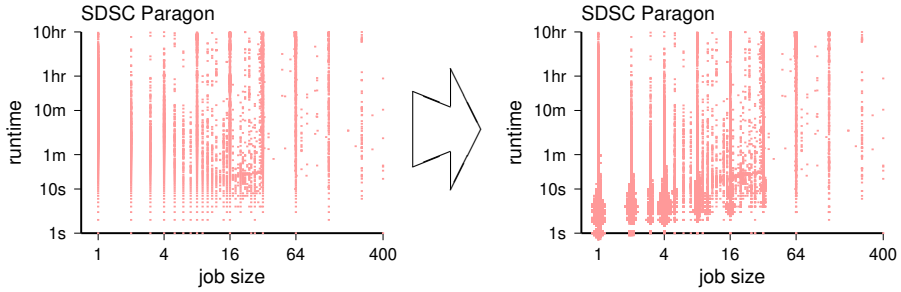
Practice Box: Scatterplots with Lots of Points

In its basic form, a scatterplot represents each data point by a small dot. For example, a scatterplot of parallel job sizes and runtimes may look like this:



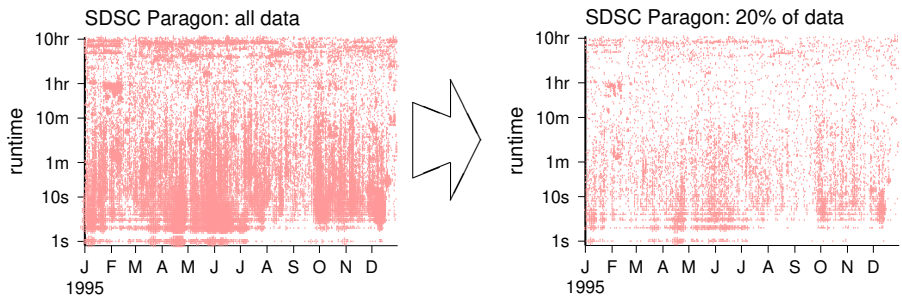
The problem with such a rendering is that in many cases several data points fall on the same location, and this information is lost. In other words, a dot may represent a single job or a thousand jobs, and we will not know the difference. It is therefore necessary to weight the dots according to the number of data points they represent.

One way of doing so is to enlarge the dots, so that their area corresponds to the number of data points represented. This immediately guides the eye to those places where the weight is concentrated:



This gives a good feel for the spread of the data, but may run into trouble in crowded areas. In particular, some fine detail may be lost as enlarged dots merge with each other and subsume nearby smaller dots.

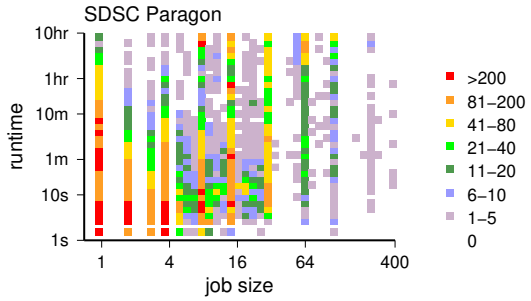
In some cases there are so many data points that the whole space seems to be filled. It may then be necessary to use sampling to reduce the clutter. This means that instead of using all the data points we use a random subset of, say, 20% of them. This has the additional benefit of reducing the size of data and graphic files. Although removal of the majority of dots results in the loss of data, it serves to bring out the main patterns:



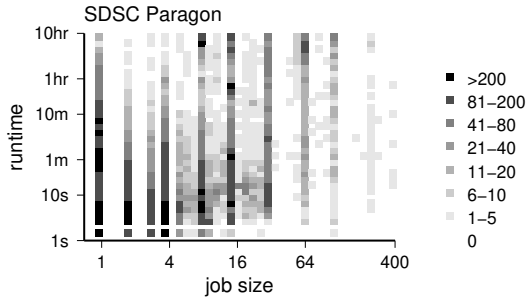
Note that data is removed randomly, so that crowded areas retain their distinction. Re-

moving repetitive data points is counterproductive, because it specifically eliminates the information about where weight is concentrated that we are seeking.

Another alternative is to use color coding. In this scheme the plotting area is pre-partitioned into small squares, and each is colored according to the number of data points that fall in it. The result is called a "heat map":



Depending on the size of the squares, this approach may also lose resolution relative to the raw scatterplot, and a legend is required in order to specify what each color means. Because it is hard to distinguish among too many colors, this approach may lead to some loss of discrimination, and care should be taken not to use colors that are indistinguishable for the colorblind. An alternative is to use a gray scale:



This is less colorful, but might be clearer and easier to interpret.

To read more: The perception of color and its use for encoding quantities are subjects for applied psychology, and have been studied by Spence [645] and others. A detailed account is provided by Ware [721, chap. 4].

End Box

As an example, consider the scatterplots in Figure 6.23. The two variables plotted here are the size and runtime of parallel jobs. The question is whether these two variables are correlated: are large jobs, which use many processors, generally longer than small jobs? Are they generally shorter? Although the scatterplots contain lots of information (specifically, they show the sizes and runtimes of tens of thousands of jobs), it is still hard to give any definite answer. In the following sections, we consider ways of extracting an answer from the data.

One way to avoid the question is to simply use a bidimensional (or, in general, a multidimensional) empirical distribution function, which simply describes the joint dis-

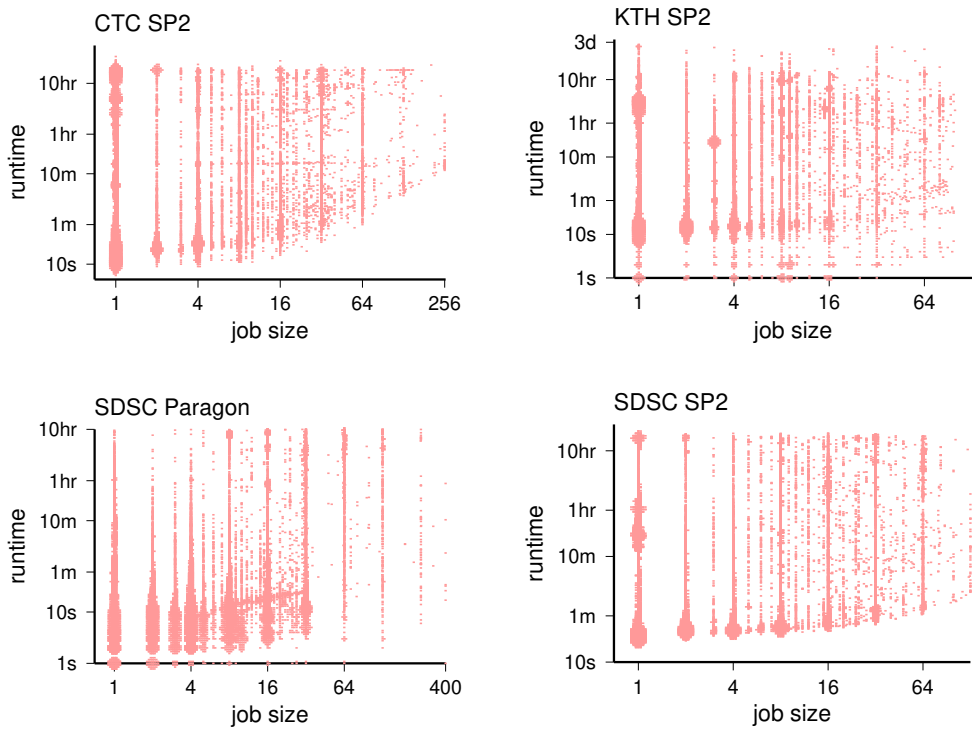


Figure 6.23: *scatterplots drawn in an attempt to assess the correlation between job sizes and runtimes on parallel supercomputers.*

tribution of the various attributes [654]. But this typically requires a very large number of parameters. It is therefore more common to try and find a good model based on the patterns observed in the scatterplot. In particular, there are two situations in which a scatterplot can lead to the immediate identification of a good model. One is when a functional relationship exists between the two variables (e.g., one is linearly dependent on the other). The other is when the data points tend to cluster together in clumps that are separated from each other.

6.4.2 The Correlation Coefficient and Linear Regression

A common way to measure the correlation between two variables is to compute the correlation coefficient. The correlation coefficient is defined as the covariance divided by the standard deviations. Let us see what this means.

Covariance and Correlation

Given n samples of X and of Y, the empirical *covariance* is calculated as

$$\mathbb{C}\text{ov}(X,Y) = \frac{1}{n-1}\sum_{i=1}^{n}(X_i - \bar{X})(Y_i - \bar{Y}) \tag{6.5}$$

Note that each term in the sum has two factors: the difference between the X sample and the average of all Xs, and the difference between the Y sample and the average of Ys. Samples in which both X and Y are larger than their respective averages contribute to the covariance. Samples in which both are smaller than their respective means also contribute to the covariance. Samples in which one is larger and one is smaller detract from the covariance. In short, if X and Y consistently behave in the same way relative to their averages, there will be a positive covariance. If they consistently behave differently, the covariance will be negative. If they are inconsistent (i.e., sometimes behaving the same and sometimes not) the covariance will be close to zero.

It is instructive to consider the covariance as a weighted sum, where the terms are the differences of Y's samples from their average, and the weights are the differences of X's samples from their average. If large deviations in Y correspond to large deviations in X, we find that the larger values get the higher weights, further increasing the covariance. If there is no such correspondence, the absolute value of the covariance will be small.

Naturally the covariance (and correlation) can also be defined for any joint distribution on two variable, without resorting to sampling. The definition is then

$$\begin{aligned}\mathbb{C}\text{ov}(X,Y) &= \mathbb{E}[\,(X - \mathbb{E}[X])\,(Y - \mathbb{E}[Y])\,] \\ &= \iint f(x,y)\,(x - \mathbb{E}[X])\,(y - \mathbb{E}[Y])\,\mathrm{d}x\mathrm{d}y\end{aligned}$$

where the expectation of X is calculated irrespective of Y, and vice versa:

$$\mathbb{E}[X] = \iint x\, f(x,y)\,\mathrm{d}x\mathrm{d}y$$

Practice Box: Calculating the Covariance

The covariance, just like the variance, can be calculated in one pass over the data. This is based on the identity

$$\begin{aligned}\mathbb{C}\text{ov}(X,Y) &= \frac{1}{n-1}\sum_{i=1}^{n}(X_i - \bar{X})(Y_i - \bar{Y}) \\ &= \frac{1}{n-1}\sum_{i=1}^{n} X_i\, Y_i - \bar{X}\,\bar{Y}\end{aligned}$$

which is easily seen by opening the parentheses and collecting terms.

End Box

Dividing the covariance by the standard deviations of X and Y normalizes it to the range between –1 and 1. This gives the *correlation coefficient*,

$$\rho = \frac{1}{n-1}\frac{\sum_{i=1}^{n}(X_i - \bar{X})(Y_i - \bar{Y})}{\mathbb{S}(X)\,\mathbb{S}(Y)} \tag{6.6}$$

(also called Pearson's correlation coefficient or the product moment correlation coefficient). In effect, the correlation coefficient measures the degree to which X and Y are

linearly related. A correlation coefficient of 1 indicates a linear relationship. A coefficient of −1 indicates an inverse relationship. More generally, correlation coefficients with an absolute value near 1 indicate that when X grows, Y grows proportionally. Small correlation coefficients indicate that no such linear relationship holds.

To understand what it means to "measure the degree to which X and Y are linearly related", note that the correlation coefficient is also a measure of the quality of a simple linear model obtained by regression. In fact, the well-known R^2 metric for regression is simply the square of the correlation coefficient. Note, however, that a relatively high correlation coefficient does *not* necessarily mean that a linear model is the correct one. It is imperative to always look at the data.

Background Box: Linear Regression

The simplest model of a relationship between two variables is a linear[2] one:

$$Y = a \cdot X + b$$

If such a relationship exists, then given a value of X one is able to predict the value of Y.

If we have a set of samples, each an (X_i, Y_i) pair, we can try to fit such a linear model. Linear regression is the process of finding the "best" values of a and b for the given samples. The metric used to determine which values are the best is that the prediction error of the regression model will be minimal. This implies an asymmetry among the variables: X is given, and Y needs to be predicted. The error for sample i is therefore not its shortest distance from the regression line, but rather the *vertical* distance from the line:

$$\text{err}_i = Y_i - (a \cdot X_i + b)$$

We are looking for the a and b that will minimize all the errors at once. Note, however, that each error may be either positive or negative. To prevent situations in which large errors in both directions cancel out, leading to a seemingly good result, we minimize the sum of the squares of the errors. The result is therefore sometimes qualified as being a *least squares* model. This model has the additional benefit that the model line will tend to pass "in the middle" between the data points, because any distance that is unduly large will cost a quadratic price.

To find the a and b that yield the minimum, we differentiate the sum of squared errors with respect to these two parameters and equate with zero:

$$\frac{\partial}{\partial a} \sum_{i=1}^{n} \left(Y_i - (a \cdot X_i + b)\right)^2 = 0$$
$$\frac{\partial}{\partial b} \sum_{i=1}^{n} \left(Y_i - (a \cdot X_i + b)\right)^2 = 0$$

Performing the differentiation yields

$$-2 \sum_{i=1}^{n} X_i \left(Y_i - (a \cdot X_i + b)\right) = 0$$
$$-2 \sum_{i=1}^{n} \left(Y_i - (a \cdot X_i + b)\right) = 0$$

[2]Linear appears twice here, in different contexts. In one we are talking about a linear model: $Y = a \cdot X + b$. In the other, we are using linear regression, which means that the model parameters a and b are found using a set of linear equations.

The second equation is the simpler one. By opening the parentheses and dividing by n, we find that

$$\bar{Y} = a\bar{X} + b$$

From this we extract an expression for b, and plug it onto the first equation. The somewhat messy result is

$$\sum_{i=1}^{n} X_i Y_i - \sum_{i=1}^{n} aX_i^2 - \sum_{i=1}^{n} X_i \left(\bar{Y} - a\bar{X}\right) = 0$$

from which we can extract an expression for a:

$$a = \frac{\sum_{i=1}^{n} X_i Y_i - \sum_{i=1}^{n} X_i \bar{Y}}{\sum_{i=1}^{n} X_i^2 - \sum_{i=1}^{n} X_i \bar{X}} = \frac{\sum_{i=1}^{n} X_i Y_i - n\bar{X}\bar{Y}}{\sum_{i=1}^{n} X_i^2 - n\bar{X}^2} = \frac{\mathbb{C}\mathrm{ov}(X,Y)}{\mathbb{V}\mathrm{ar}(X)}$$

and, given a, we can calculate b as

$$b = \bar{Y} - a\bar{X}$$

The main point to understand regarding the formulas for a and b is that they are just that: formulas. Given any set of samples $\{(X_i, Y_i)\}$, plugging the data into the formulas will yield values for a and b. This does not mean that a linear model is in any way appropriate.

Verifying that the model is meaningful is based on analysis of variation. The variation in question is the sum of the squares of the differences between the Y values and their mean, called the "sum squares total":

$$SST = \sum_{i=1}^{n} (Y_i - \bar{Y})^2$$

This sum reflects a situation of no prior knowledge regarding each sample, so the best we can do is assume that it is represented by the mean. But the regression model claims that we can do better: if we know X_i, we can predict Y_i based on the formula $\hat{Y}_i = aX_i + b$. The difference between Y_i and $\bar{Y}$ can therefore be partitioned into two parts: one part equaling $\hat{Y}_i - \bar{Y}$ that is explained by the linear model, and the rest. Assuming we believe in the model, the residual reflects the error made by the model.

The quality of the regression depends on how well the X values enable us to predict the Y values. The model is good if a large part of the variation may be attributed to the regression, and only a small part to the error. The metric is therefore the ratio between the sum squares of the regression and the sum squares total:

$$R^2 = \frac{\sum_{i=1}^{n} (\hat{Y}_i - \bar{Y})^2}{\sum_{i=1}^{n} (Y_i - \bar{Y})^2} \tag{6.7}$$

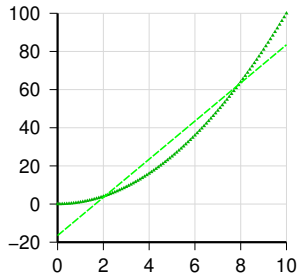
This is often called the coefficient of determination.

The interesting thing is that $\hat{Y}_i$ can be expressed as a function of X_i, a, and b, and a and b in turn can be expressed as functions of the X_is, Y_is, and their means. Therefore R^2

can be expressed directly as a function of the original samples. Doing so involves some tedious algebra. The end result is that R^2 equals the square of the correlation coefficient of X and Y. In this sense, the correlation coefficient reflects the degree to which a linear relationship exists between X and Y.

Incidentally, this also implies that the roles of X and Y may be reversed. If we use Y to predict X, we will typically get a different linear model (because we find a and b values that minimize the horizontal distance from the regression line, rather than the vertical distance). However, this alternative line will have the same quality as the first as measured by R^2.

Given the mathematics, one should also realize its limitations. High R^2 values do not necessarily mean that the data indeed conforms to a straight line. For example, if we perform a linear regression on the 101 points of the form (x, x^2) with x going from 0 to 10 in jumps of 0.1, we get a correlation coefficient of 0.97 and $R^2 = 0.94$. Although the line is indeed close to the data, this is obviously not the best model. It is crucial to always look at the data first [31].



End Box

It is important to also note what the correlation coefficient does *not* measure. It does not measure the slope of the regression line. A correlation coefficient of 0.7 does not imply that the slope is steeper than when the correlation coefficient is 0.5. A relatively flat sequence of points that lie close to a line will have a high correlation coefficient, but a low slope. Conversely, an oblique cloud of points can have a low correlation coefficient, but a high slope. The slope is given by a and can vary from $-\infty$ to ∞. The correlation coefficient measures the quality of the linear model, and varies from –1 to 1.

However, a relationship between the slope and the correlation coefficient does exist. Recall that the slope of the linear regression line is

$$a = \frac{\sum_{i=1}^{n} X_i Y_i - n\bar{X}\bar{Y}}{\sum_{i=1}^{n} X_i^2 - n\bar{X}^2} = \frac{\mathbb{C}\text{ov}(X,Y)}{\mathbb{V}\text{ar}(X)}$$

The correlation coefficient can be written as

$$\rho = \frac{\sum_{i=1}^{n} X_i Y_i - n\bar{X}\bar{Y}}{\sqrt{\sum_{i=1}^{n} X_i^2 - n\bar{X}^2}\sqrt{\sum_{i=1}^{n} Y_i^2 - n\bar{Y}^2}} = \frac{\mathbb{C}\text{ov}(X,Y)}{\mathbb{S}(X)\mathbb{S}(Y)}$$

Note that the numerators in both expressions are the same, and the difference is only in the denominator, which is the variance of X in the first one, and the product of the two standard deviations in the second. Therefore the relationship is simply

$$a = \frac{\mathbb{S}(Y)}{\mathbb{S}(X)}\rho$$

This also explains why the sign of the correlation coefficient corresponds to the sign of the slope.

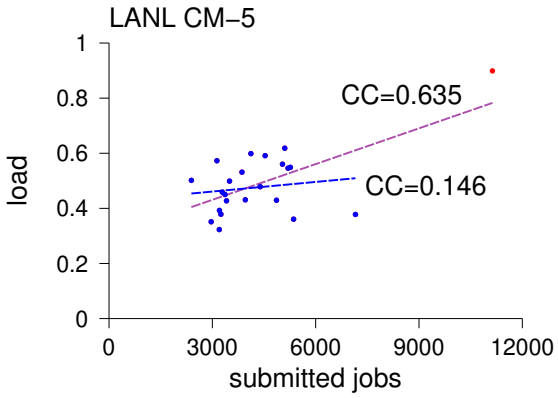


Figure 6.24: *Outliers can have a very strong effect on regression results.*

To read more: An extensive treatise on the correlation coefficient and its meaning, starting with intuition and ending with alternative computation methods, was written by Rummel [588].

The correlation coefficient has two main drawbacks. One is that it may not be sensitive enough, especially to forms of correlation that cannot be expressed as a linear relationship. The other is that it is overly sensitive to outliers.

Sensitivity to Outliers

The sensitivity to outliers is demonstrated in Figure 6.24. This scatterplot was drawn to investigate the relationship between the number of jobs submitted and the system load. Data of the two-year log from the LANL CM-5 is partitioned into months, and for each month we compute two variables: the number of jobs submitted during that month, and the system load, measured as a fraction of capacity. The results for all the months but one form a rather diffuse cloud. Performing a linear regression leads to a line with a very shallow slope, and the correlation coefficient is also rather low at 0.146.

But the data also contains one outlier month, in which both the number of jobs and the load are much higher. When this additional data point is included, the slope of the regression line becomes steeper, and the correlation coefficient jumps up to a respectable 0.635. Taken at face value, these figures indicate that indeed there is a significant correlation between jobs and load, and therefore differences in load may be modeled by differences in the number of jobs. But this conclusion rests mainly on the effect of the single outlier month.

The reason that a single outlier has such an effect is the quadratic nature of the correlation coefficient and the regression analysis. The farther away a data point is from the rest, the larger its effect. In particular, when there is a single outlier, the regression line will tend to connect this outlier with the center of mass of the rest. The distribution of the other data points will only have a small effect on the results.

Incidentally, the sensitivity to outliers also affects the use of linear regression for

predictions. The common way to perform linear regression, described in the box on page 268, is based on minimizing the squares of the vertical deviations from the fitted line. Doing so emphasizes the effect of the points with the biggest distance. The alternative is to use the absolute values of the deviations in the optimization. Because it is more natural to use the absolute value of the deviations as the metric of quality, the second method leads to a better fit and better predictions [662].

The Rank Correlation Coefficient

The limitation of only measuring a linear relationship is reduced by using the *rank correlation coefficient*. In this approach, instead of using the data values directly, we first sort them, and than calculate the correlation between their ranks instead [644]. For example, consider an item for which $x = 13.2$ and $y = 25.7$. If the rank of this x value is 19th out of all the xs of the different items, and the rank of this y value is the 15th of all ys, then the item will be represented by the pair $(19, 15)$, instead of the pair $(13.2, 25.7)$. Calculating the correlation coefficient on the ranks then gives a measure of the degree to which X can be used to predict Y. In particular, a full correlation is obtained for all cases where the relationship between them is monotonic; it does not have to be linear.

The formula for the rank correlation coefficient is as follows. Denote the rank of X_i by r_i^X, and the rank of Y_i by r_i^Y. Because the ranks are actually the integers from 1 to n, the average ranks are $\frac{n+1}{2}$ for both X and Y. The rank correlation coefficient (also called Spearman's rank correlation coefficient), which is simply the correlation coefficient of Equation (6.6) applied to the ranks, is then

$$r = \frac{\sum_{i=1}^{n} \left(r_i^X - \frac{n+1}{2}\right)\left(r_i^Y - \frac{n+1}{2}\right)}{\frac{1}{12}(n^3 - n)} \tag{6.8}$$

The denominator is simply the variance of the uniformly distributed ranks (it can be derived using the equality $\sum_{i=1}^{n} i^2 = \frac{1}{6}\, n\,(n+1)(2n+1)$).

A simpler way to actually compute the rank correlation coefficient is the following equivalent expression. For each item, calculate the difference between its X rank and its Y rank, $d_i = r_i^X - r_i^Y$. The rank correlation coefficient is then

$$r = 1 - \frac{6\sum_{i=1}^{n} d_i^2}{n^3 - n}$$

The equivalence can be demonstrated by expressing the squared difference as $d_i^2 = \left((r_i^X - \frac{n+1}{2}) - (r_i^Y - \frac{n+1}{2})\right)^2$, applying straightforward algebraic manipulations and comparing with Equation (6.8), and using formulas for the sum and sum of squares of the ranks, which are simply all the numbers from 1 to n.

Note that special care must be taken if several workload items have the same value. This often happens for workload attributes that are discrete in nature, such as the size of a parallel job, which is an integer. A set of items with the same value will then

cover a range of ranks, but in reality they should all have the same rank. The common approach is to use the middle of the range to represent all of them. Note, however, that the denominator has to be adjusted too, because it is no longer the variance of uniformly distributed ranks, but rather the variance of ranks that come in groups.

An alternative to Spearman's rank correlation coefficient is Kendall's τ [399]. Consider any pair of elements i and j. We call them concordant if they are ranked in the same order according to both X and Y; that is, if either $r_i^X < r_j^X$ and $r_i^Y < r_j^Y$, or else $r_i^X > r_j^X$ and $r_i^Y > r_j^Y$. Otherwise we call them discordant. Denote the number of concordant pairs by C and the number of discordant pairs by D. Kendall's τ is the normalized difference between the number of concordant and discordant pairs, namely

$$\tau = \frac{2(C-D)}{n(n-1)}$$

where the normalization factor is the number of possible pairs, which is $\frac{1}{2}n(n-1)$. An important advantage of this coefficient over Spearman's is that it has an intuitive meaning: it is the difference between the probability that a pair of observations appear in the same order and the probability that they appear in different orders [518].

As with Spearman's coefficient, here too one may need to deal with ties. Items with the same X or Y values are neither concordant nor discordant, so they are not counted in C or D. Therefore the normalization has to be done differently. The simplest solution is to nevertheless use the sum of C and D, which would actually be correct if there were no ties. This leads to

$$\hat{\tau} = \frac{C-D}{C+D}$$

An alternative is to use the geometric mean between the number of pairs after subtracting the pairs with ties in X and in Y. Denote the number of different values of X by n_X, and the number of occurrences of the jth value by t_j^X, and do so similarly for Y. The expression for τ is then

$$\tau_b = \frac{C-D}{\sqrt{\left(\frac{n(n-1)}{2} - \sum_{j=1}^{n_X} \frac{t_j^X(t_j^X-1)}{2}\right)\left(\frac{n(n-1)}{2} - \sum_{j=1}^{n_Y} \frac{t_j^Y(t_j^Y-1)}{2}\right)}}$$

Given that all pairs of values need to be compared, calculating Kendall's τ takes quadratic time. However, efficient algorithms requiring only $O(n \log n)$ time have been given by Christensen [134].

A variant of this metric is Kendall's concordance, which measures the agreement between m rankings rather than just two [400]. As ranks range from 1 to n, the mean rank is $\frac{n+1}{2}$. Denote the jth ranking of the ith item by r_i^j, and consider the sum of all ranks received by a certain item. The deviation of this sum from m times the mean is

$$d_i = \sum_{j=1}^{m} r_i^j - \frac{m(n+1)}{2}$$

		Rank CC		
System	*CC*	*Spearman*	*Kendall*	*Dist CC*
KTH SP2	0.039	0.250	0.185	0.876
CTC SP2	0.057	0.244	0.186	0.892
SDSC Blue	0.121	0.411	0.373	0.993
SDSC SP2	0.146	0.360	0.276	0.962
NASA iPSC	0.157	0.242	0.198	0.884
SDSC Paragon	0.280	0.486	0.378	0.990

Table 6.2: *Correlation coefficients of runtime and size for different parallel supercomputer workloads (using the cleaned versions).*

The concordance is the normalized sum of the squares of these deviations:

$$W = \frac{12 \sum_{i=1}^{n} d_i^2}{m^2(n^3 - n)}$$

Comparison

We demonstrate the behavior of all these correlation coefficients by calculating the correlation of the data shown in the scatterplots of Figure 6.23. The results for these and additional datasets are given in Table 6.2. For the Pearson correlation coefficient the results range from essentially zero (i.e., no correlation at all) to about 0.3 — a weak positive correlation. The results for the rank correlation coefficients are somewhat higher, in one case nearly 0.5. In general, Spearman's coefficient is slightly higher than Kendall's τ.

But looking at the scatterplots, it is hard to find any correspondence between these results and the spread of the data. Moreover, sorting the systems by the rank correlation coefficient would lead to a different order than the one used in the table, which is sorted by the conventional correlation coefficient. For example, the NASA iPSC system has a higher correlation coefficient than the SDSC Blue system, but a much lower rank correlation coefficient.

For this type of data, we need the metric shown in the fourth column, which is developed in the next section.

6.4.3 Distributional Correlation

The conventional correlation coefficients measure the degree to which one random variable can be used to predict another. This can be generalized by using one variable to predict the *distribution* of another.

Conditional Distributions

As we have seen, establishing whether or not a correlation exists is not always easy. The commonly used correlation coefficient only yields high values if a strong linear

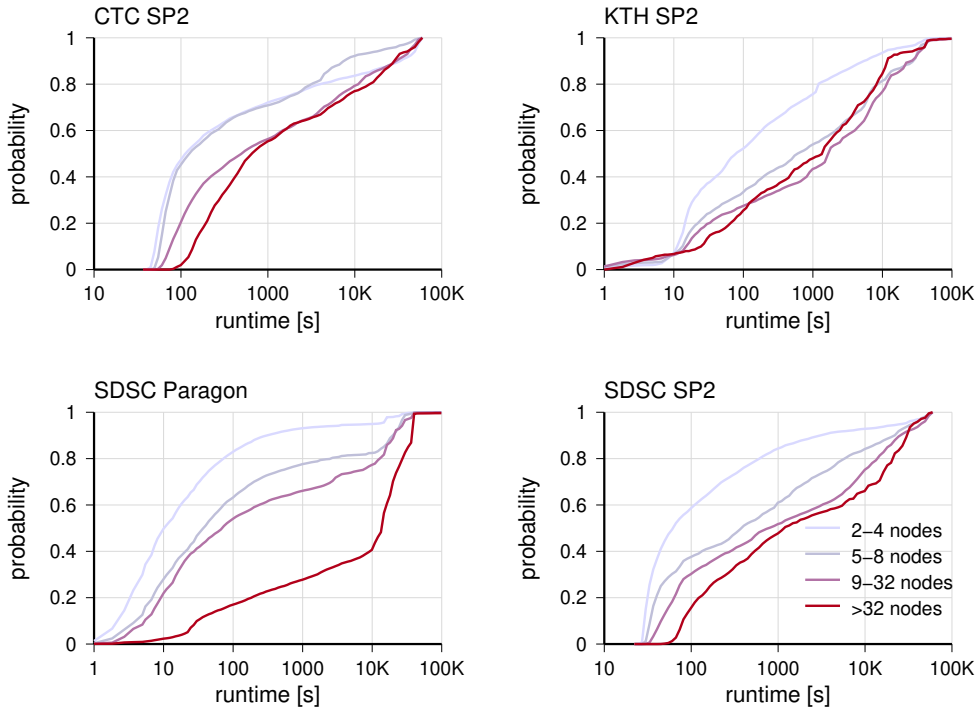


Figure 6.25: *Distributions of runtimes drawn for jobs with different degrees of parallelism expose a weak correlation when serial jobs are excluded.*

relationship exists between the variables. In the examples using the size and runtime of parallel jobs the correlation coefficients are typically rather small (Table 6.2), and scatterplots show no significant correlations as well (Figure 6.23).

However, these two attributes are sometimes actually correlated with each other in the following sense. Consider the distributions of runtimes of small jobs and of large jobs separately. These distributions cover essentially the same range, from a few seconds to several hours. But the distribution of weight across this range is different. For small jobs, much of the weight is concentrated at short runtimes. For large jobs, more of the weight is spread across longer runtimes. Thus *the distribution of the runtime depends on the size*.

This principle is demonstrated in Figure 6.25 for the four datasets used earlier. The jobs in each log are partitioned into five groups according to size. The distribution of runtimes is then plotted for each size independently. These are actually conditional distributions: each one is a distribution of the runtimes, conditioned on the size being in a given range. In general, the runtime distributions belonging to larger jobs tend to be to the right and below those of smaller jobs. This means that there are more longer runtimes and fewer short runtimes. The main deviations from this pattern are due to serial jobs, which tend to be longer than small parallel jobs.

(Although this discussion indeed seems to be a good description of the correlations

observed on parallel machines, there are exceptions. To illustrate this metric we focus here on those examples that display distributional correlation. A more complete picture regarding parallel workloads is given in Section 9.6.2.)

The Distributional Correlation Coefficient

To turn these observations into a quantified metric, we can suggest the following procedure [236]. First, partition each workload log into just two equal parts: the half with the smaller jobs, and the half with the larger jobs (where "small" or "large" relates to the number of processors used). Then plot the cumulative distribution functions of runtimes for the jobs in the two sets. If one CDF is consistently above the other, we say that a distributional correlation exists. If the two repeatedly cross each other, there is no such correlation.

Background Box: Stochastic Dominance

Distributional correlation is closely related to stochastic dominance, or, more precisely, first-order stochastic dominance. In simple terms, dominance means that one random variable tends to be bigger than the other. More formally, this is defined as follows. Consider two random variables, X_A from distribution A and X_B from distribution B. We will say that A dominates B if for every value x the probability that $X_A \geq x$ is at least as high as the probability that $X_B \geq x$. This will be denoted by $X_A \succeq X_B$. If in addition there is some x where the probability that $X_A \geq x$ is strictly larger than the probability that $X_B \geq x$, then we will denote this by $X_A \succ X_B$.

Note that this definition is actually a relationship between the distributions from which the two random variables come. The CDF of a distribution specifies the probability of being smaller than a value x. So if the probability of being larger than x is higher, the CDF should be lower. Thus the CDF of the dominating random variable should be lower and to the right of the CDF of the dominated random variable. In mathematical notation, if $X_A \succeq X_B$, then $F_A(x) \leq F_B(x)$ for every x, and if $X_A \succ X_B$, then for some x we also have the stronger condition that $F_A(x) < F_B(x)$.

Dominance as defined here is rather strong in the sense that the CDFs may never cross: one should always be above the other (or they may overlap). If the CDFs cross each other then neither distribution dominates the other. As a result this is not a full order. For example, normal distributions with different standard deviations do not dominate each other regardless of how far away their means are, because asymptotically the tails of the one with the lower standard deviation will tend to zero faster, and at least one of them will cross the tail of the distribution with the higher standard deviation.

Our notion of distributional correlation is less strict. Essentially, it can be understood as quantifying how close we are to dominance, by estimating for what fraction of the values (as weighted by their probability) one distribution dominates the other.

End Box

For the specific case of parallel job sizes and runtimes, example graphs are shown in Figure 6.26. They show essentially the same data as in Figure 6.25, except for the fact that the data is partitioned into only two sets (serial jobs are again excluded). Obviously, this results in a clearer picture, and it is easy to see that one CDF is nearly always above the other.

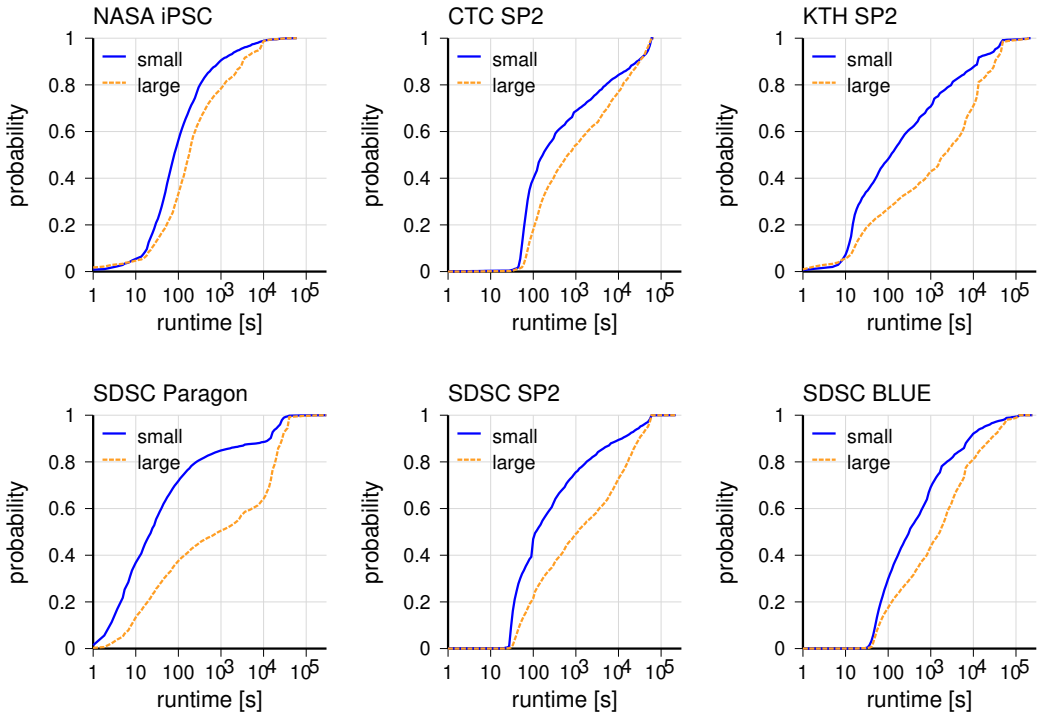


Figure 6.26: *Data for calculating the distributional correlation coefficient. Serial jobs are excluded.*

More precisely, the procedure to calculate the distributional correlation proceeds as follows. We start with a set of n samples of pairs of variables, which we denote by $P = \{(X_i, Y_i)\}$.

1. Find the median of the X_is, and denote it by $X_{(.5)}$.
2. Partition the data into two equal subsets: one including those X_i that are smaller than $X_{(.5)}$, and another in which they are larger. Samples for which $X_i = X_{(.5)}$ are assigned at random to both subsets in a way that ensures that the subsets have equal sizes.
3. Find the empirical distribution of Y_i for each part of the data. The distribution of Y_i conditioned on X_i being smaller than its median will be denoted $F^{\perp}(y) = |\{Y_i \mid Y_i \leq y \wedge X_i \leq X_{(.5)}\}|$. Similarly, $F^{\top}(y) = |\{Y_i \mid Y_i \leq y \wedge X_i \geq X_{(.5)}\}|$. Note that formally these are not really distributions but counts of samples, because they are not normalized to sum to 1.
4. Calculate the distributional correlation coefficient as

$$distCC = \frac{1}{|P|} \sum_{i \in P'} \mathrm{sgn}\Big(F^{\perp}(Y_i) - F^{\top}(Y_i)\Big) \tag{6.9}$$

where $|P| = n$, and sgn is the sign function that gives each sample one vote:

$$\mathrm{sgn}(x) = \begin{cases} 1 & \text{if } x > 0 \\ -1 & \text{if } x < 0 \end{cases}$$

The set of samples P' over which the sum in the last step is taken is important. In principle it should be the set P of all sample points. But this choice is susceptible to high values that occur by chance. To prevent this, one has to limit the sum to points where the CDFs have some minimal distance from each other. Specifically, it is recommended to use

$$P' = \left\{ i \;\middle|\; \left| F^{\perp}(Y_i) - F^{\top}(Y_i) \right| > 2\sqrt{|\{Y_j \mid Y_j \le Y_i\}|} \right\}$$

The motivation for this expression stems from considering the two CDFs when samples are randomly assigned to the small and large groups (thus simulating the case where the two CDFs are actually samples from the same distribution). We start from the smallest values and move upward. Both CDFs obviously start at zero. Each new sample has a probability of 0.5 to either increase the CDF of the small group or to increase the CDF of the large group. Thus the difference between the CDFs behaves like a random walk with equal probabilities of taking a step in either direction. It is well known that the expected distance that such a random walk covers after n steps is $\sqrt{n}$ (as explained on page 325). Therefore fluctuations in the difference between the CDFs up to the square root of the number of samples we have seen so far do not indicate any systematic divergence. Adding a factor of 2 provides a safe margin, ensuring that only significant points are used.

This definition of a distributional correlation coefficient leads to values in the range $[-1, 1]$, with 1 indicating the strongest correlation, –1 indicating an inverse correlation, and values near 0 indicating little correlation — the same as for other coefficients. However, it has the drawback of not being symmetric, as opposed to conventional correlation coefficients that are symmetric. The lack of symmetry is the result of choosing one variable to split the observations into two, and then plotting the distributions of the other variable. It is possible that different results would be obtained if we would choose the variables the other way around [236].

The results of computing this metric for several parallel job logs are shown in Table 6.2, and compared with the conventional correlation coefficients. Serial jobs are again excluded because of their unique characteristics. Obviously the distributional coefficients are quite close to 1, as one would expect based on the distributions shown in Figure 6.26.

6.4.4 Modeling Correlations by Clustering

A coarse way to model correlation, which avoids the problem altogether, is to represent the workload as a set of points in a multidimensional space and then apply clustering [19, 103]. For example, each job can be represented by a tuple including its runtime, its I/O activity, its memory usage, and so on. By clustering we can then select a small number of representative jobs, and use them as the basis for our workload model; each

such job comes with a certain (representative) combination of values for the different attributes [99].

Clustering is especially popular in the context of queueing analysis. In its simplest form, queueing analysis assumes that all jobs come from the same class, and are governed by the same distributions. The next step toward realism is to assume a multiclass workload, in which jobs from several classes are present, and each class has distinct distributions. Clustering is used to identify and characterize the different classes [104].

There are many different clustering algorithms, but they share a general framework [365]. The essence of clustering is to find a partitioning of the data into disjoint sets, such that the points within each set are similar to each other, while the sets themselves are dissimilar. The different algorithms use different definitions of what is similar or dissimilar. They also differ in whether they find the appropriate number of clusters automatically, or need this number to be specified in advance.

To read more: Clustering is treated in many textbooks on performance or data analysis, e.g. Jain [367, sect. 6.8] and Berthold et al. [71, chap. 7]. A good review is provided by another Jain [365]. There are also several book-length treatments devoted exclusively to clustering, e.g. Everitt et al. [222].

Preprocessing for Clustering

Before applying a clustering algorithm, some preprocessing is typically required. The main reason for this is that different attributes may have completely different units and scales. To compare distances in the different dimensions, a common scale is needed. The classical way to do scaling is to subtract the mean of each attribute and divide by the standard deviation:

$$Z = \frac{X - \bar{X}}{\mathbb{S}(X)} \tag{6.10}$$

Using this result, the deviations of all attributes from their respective averages are measured in units of their respective standard deviations (it is so common to denote this by Z that it is known as the "Z score"). However, it has the disadvantage of being oblivious to scale [556].

An alternative is to use a logarithmic transformation, which actually measures orders of magnitude. A logarithmic transform is especially suitable for skewed distributions and heavy tails, because it maintains the details of the myriad small values, while reducing the huge variance caused by the few large values. Without it, there is a danger that the clustering will group all the small values together and leave each of the large values in a cluster by itself. If all attributes are log-transformed no scaling is needed, because orders of magnitude serve as the common scale.

The k-means Clustering Algorithm

The simplest clustering algorithm is the k-means algorithm. This assumes that the number of desired clusters, k, is known in advance. Assume there are n data points, $p_1, \ldots p_n$. Each data point has two attributes, corresponding to the X and Y variables

discussed earlier: $p_i = (X_i, Y_i)$. The procedure generalizes naturally to more than two dimensions. The k clusters will be designated C_1 to C_k. The algorithm works as follows.

1. Select k points at random and designate them as the initial centroids. Note that it is undesirable just to use the first k points, because these may be very close to each other if the data displays locality of sampling.
2. For each of the n points, find the cluster centroid that is nearest to it, and assign the point to that cluster.
3. Calculate the new centroid of each cluster. This is the "center of mass" of the points assigned to this cluster. Thus the coordinates of the centroid (c_j^X, c_j^Y) of cluster C_j will be the averages of the coordinates of these points:

$$c_j^X = \frac{1}{|C_j|} \sum_{p_i \in C_j} X_i, \qquad c_j^Y = \frac{1}{|C_j|} \sum_{p_i \in C_j} Y_i$$

4. If no point has been reassigned in step 2 (i.e. they were all already assigned to the nearest cluster), this is it. Otherwise, return to step 2.

An alternative to the first step is to assign the n points to the k clusters at random, so that each cluster has approximately the same number of points, and at least one point, and then calculate their centroids. However, for large datasets, this runs a greater risk of having all centroids next to each other and to the center of the whole dataset.

Details Box: Measuring Distance

Step 2 in the k-means algorithm assigns each data point to the "nearest" cluster. This actually depends on how we measure distance.

The most common approach is to use the Euclidean distance. This is what we commonly use in geometry. For d dimensional data, the definition is

$$||X - Y||_2 = \sqrt{\sum_{i=1}^{d} (X_i - Y_i)^2}$$

(note that here X and Y are samples and the index i refers to dimensions; thus $X = (X_1, \ldots X_d)$ and similarly for Y.) This is sometimes called the "2-norm" or the "ℓ_2 distance" due to the use of the power of two in the expression; the double vertical bars are similar to the notation for an absolute value.

A generalization of this idea is to use a power other than two. This leads to

$$||X - Y||_p = \sqrt[p]{\sum_{i=1}^{d} \left|X_i - Y_i\right|^p}$$

(note that the absolute value of the differences is used to ensure that the result is always positive.) In particular, when the power p is 1, we are actually summing the differences along the different dimensions, so

$$||X - Y||_1 = \sum_{i=1}^{d} |X_i - Y_i|$$

This is sometimes called the Manhattan distance, based on the analogy to traveling in Manhattan, where we can only move along streets and avenues but not in a diagonal. At the other extreme, when $p = \infty$, the largest difference dominates all the rest. We therefore get

$$||X - Y||_\infty = \max_{i=1..d} |X_i - Y_i|$$

These distance metrics assume that the attributes characterizing each element are numerical. But they can also be binary (interactive work vs. background work) or categorical (jobs belonging to user group A, user group B, and so on). In such cases distance is usually taken to reflect the degree of (dis)similarity, e.g. the fraction of attributes that are different [222].

In addition to defining the metric, we also need to decide how to apply it. The description of the k-means algorithm stated that distance is measured from each point to the centroid of each cluster. In hierarchical algorithms that operate on clusters rather than on individual points (described later) there are several options for measuring the distance between two clusters: the distance between the nearest two elements, the average distance between all pairs of elements, the distance between the centroids, and so on. For categorical data, one can consider the set of items in each cluster as a whole. The distance is then a function of the differences in the fraction of items that occupy each level of each attribute. Using different approaches will lead to the construction of different clusters.

End Box

An example of the results obtained by using the k-means algorithm is shown in Figure 6.27. The data is the sizes and runtimes of jobs run by user 8 in the SDSC Paragon log. In three of the clusters the centroid indeed corresponds to a concentrations of jobs that have the same job sizes and similar runtimes. The fourth is between two groups of jobs with different numbers of processors.

This example also exposes one of the subtle points of clustering. When using clustering, the clusters are often represented by their centroids. As shown in Figure 6.27, this may not be appropriate. In the parallel job data of this example, job sizes are typically powers of two. But the centroids, which are an average of many jobs with different sizes, may have sizes that are not powers of two, and even may not be integers. This may have a strong impact on performance evaluations, because jobs that are not powers of two are harder to pack efficiently. Therefore a size that is a power of two should perhaps be chosen to represent each cluster, even if it is not exactly at the centroid.

Another issue with k-means clustering is the effect of the starting points. Unless the points have a clear natural partitioning into k clusters, there is a strong probability that the results will depend in some way on the initial points used. A good criterion for selecting the initial points is that they provide a good coverage of the data. This can be achieved by using synthetic scrambled midpoints rather than actual data points [573]. This method works as follows:

1. For each dimension, find its range (that is, the minimal and maximal values observed in the data).

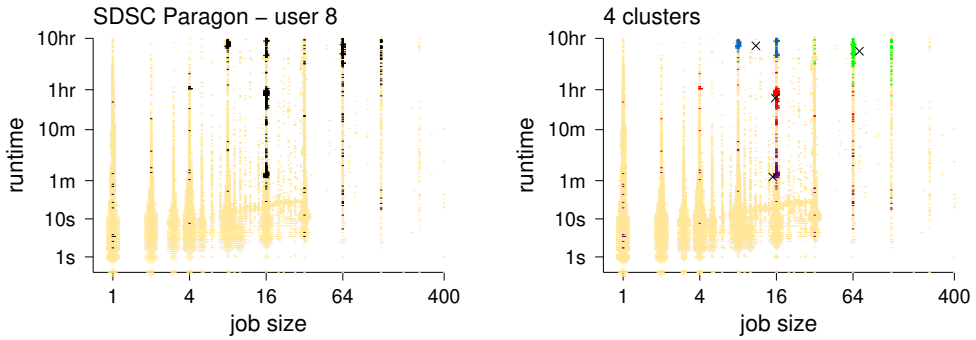


Figure 6.27: *Left: jobs of user 8 in the SDSC Paragon log, shown against a backdrop of other jobs in the log. Right: results of clustering these jobs using the k-means clustering algorithm for four clusters, after a logarithmic transform. Cluster centroids are indicated by $\times$s.*

2. Divide each dimension range into k equal partitions, and find their midpoints.
3. Create the first starting point by selecting one partition midpoint at random from each of the dimensions.
4. Repeat this k times to create k initial starting points.

Hierarchical Clustering

The k-means algorithm is "flat" or "partitional", in that it partitions the data into k disjoint clusters that all have equal standing, and k must be known in advance. The alternative is to use hierarchical clustering, in which clusters subsume each other in a nested structure. There are two basic approaches used for hierarchical clustering: bottom-up and top-down. The bottom-up (agglomerative) approach starts with all data points as individual clusters, and iteratively unites existing clusters to create larger clusters. The top-down (divisive) approach starts with all data points in one big cluster, and successively divides clusters into two.

A simple variant of the bottom-up approach proceeds as follows:

1. Initially each point is defined to be a cluster by itself.
2. Find the two clusters that are closest to each other. For example, this can be defined to be the pair of clusters that have the closest points to each other (the so-called "single link" algorithm), or the pair of clusters with the closest centroids.
3. Unify the two clusters. This means that a new cluster is formed, that contains all the points in the original two clusters.
4. Find the centroid of the new cluster if needed, and return to step 2.
5. Continue the procedure until a single cluster is formed.

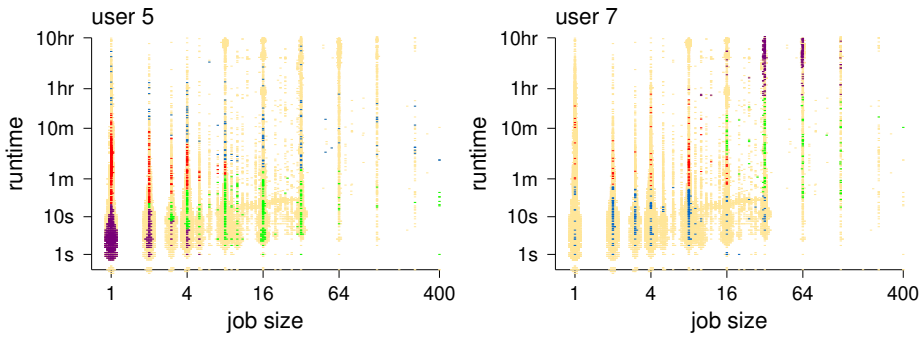


Figure 6.28: *Applying clustering to the activity of users 5 and 7 in the SDSC Paragon log is much less convincing than that shown in Figure 6.27 for user 8.*

A simple variant of the top-down approach is based on using the k-means algorithm as a subroutine, with $k = 2$. The clustering algorithm is then as follows:

1. Initially all the points are in a single cluster.
2. Apply the k-means algorithm with $k = 2$ to partition this cluster into two smaller clusters.
3. Repeat this step recursively until each cluster is divided into individual points.

Both of these procedures create a binary tree above the data points. The leaves of the tree are single points. Each internal node in the tree represents a cluster including all the points in the leaves of the subtree rooted at this node. Because the tree is binary, this is the unification of two smaller clusters. The result is a set of clusters with inclusion relationships. It is best represented by a dendogram, where the branches of the tree are L-shaped and the vertical part represents distances. It can then be cut at a certain level to obtain the desired number of clusters.

Applicability of Clustering

Blindly applying a clustering algorithm to data is guaranteed to produce a result. The question is whether this result is meaningful. In many cases the distributions of workload attribute values are not modal but continuous. Likewise, the combinations of different attributes may not cluster naturally. For example, the scatterplots of parallel job sizes and runtimes (Figure 6.23) do not seem to contain significant clusters.

If clustering is nevertheless applied in such situations, it will produce results that are not representative. First, the clusters will not represent the full workload because the full workload will contain many more combinations of values. Second, the clusters themselves will be sensitive to the exact procedure used to find them; small variations in procedure may lead to completely different clusters.

Clustering may work better when workload distributions are modal. This tends to be the case when the workload generated by a single user is considered, because users

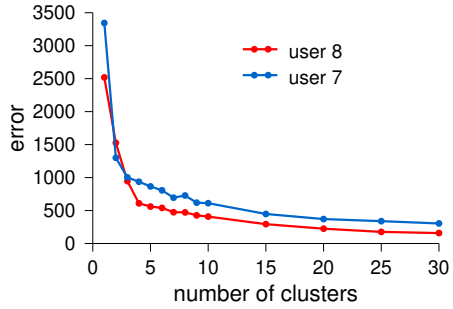


Figure 6.29: *Error as a function of the number of clusters, for the examples of Figures 6.27 and 6.28.*

typically repeat the same work over and over again. It may therefore be possible to apply clustering to the work of a single user, and to create a user behavior graph. The example in Figure 6.27 was of this type, and showed how clustering identifies concentrations of similar jobs. However, it should be remembered that this is not always the case, as demonstrated in Figure 6.28. The unique properties of workloads generated by different users are the topic of Chapter 8, and user behavior graphs are discussed in Section 8.3.3.

To assess whether clustering is applicable, it must be validated. The simplest approach is visual validation (the "eyeball method"): create a scatterplot of the data, and verify that clusters indeed stand out. If they do not, as is the case in Figure 6.28, clusters that are identified by automatic means will probably not be meaningful. One can also partition the data randomly and apply the clustering to each part separately. Consistent results bear witness to the validity of the clustering.

A more mechanical method to assess the quality of clustering is as follows. Given a clustering of data, we can measure its quality by the "errors" involved. For example, we can define the sum of the distances of the data points from the centroids of their respective clusters as a metric of the error. We can then create different clusterings, with different numbers of clusters (e.g., by running the k-means algorithm with successive values of k). Plotting the error as a function of the number of clusters will generally lead to a decreasing function: when all the points are in one cluster, the distances from the centroid are large, but when each is in its own cluster the distances are zero. If this function initially decreases rapidly, and then levels off at a low value (that is, the graph has an L shape with an "elbow"), the clustering is considered effective, and the number of clusters to use is the number at the elbow.

An example of using this scheme is given in Figure 6.29. The result for the clustering of the user 8 data from Figure 6.27 is indeed very nice, clearly indicating that four clusters should be used. Regrettably, the result for the user 7 data from Figure 6.28 is not much worse, despite looking much less convincing in the scatterplot. The moral is that one should always verify that tests indeed distinguish between "good" and "bad" samples of the data at hand.

An alternative formalism uses the explained variance as the metric, rather than the

errors. When the complete dataset is viewed together, its variance can be calculated based on the distances of all the points from the center of gravity. But when the points are assigned to clusters, the variance can be partitioned into two: one part that represents the distances between the clusters, and another that represents the residual variance that remains within each cluster. The difference between them is the variance that is explained by the clustering. The fraction of the total variance that is explained grows from 0 when all the points are in one cluster to 1 when each point has its own cluster. Again, if initially the explained variance grows rapidly as the number of clusters is enlarged, and then it levels off, the knee identifies the number of clusters to use.

The problem with visual (or even statistical) validation is that it is oblivious to the goals of workload modeling. Given that we create models for performance evaluation, good models are those that lead to reliable evaluations. This implies that clustering should be validated by comparing evaluation results for the original workload and the cluster-based model of the workload [99, 556]. If the results are compatible, the clustering is valid.

To read more: The formal framework for studying clustering is provided by graphs, where the vertices are the items and the edges represent the distances. Clustering is then a partitioning of the vertices such that each partition is dense, but connections among partitions are sparse. An exposition along these lines, comparing metrics for the quality of the clustering, is given by Gaertler [274].

Workload Modeling with Clusters

The pure form of clustering as described earlier implies a partitioning of the data: each item belongs to one and only one cluster. This strict requirement is the source of some of the problems with clustering, because in many cases the items being clustered do not have modal distributions. Therefore the data does not partition naturally into disjoint clusters.

But in workload modeling, it is often actually OK to have clusters that overlap (e.g., in relation to models based on using a mixture of distributions, a concept we reviewed in Section 4.3). In particular, a mixture of multinormal distributions is often used.

The multinormal distribution (or multivariate normal distribution) is the joint distribution of multiple normal random variables. Thus if a workload is characterized by k attributes, a "cluster" of workload items can be characterized by a set of k normal distributions, each specifying the mean and standard deviation of one attribute. Workload items in this cluster are those that have a combination of attribute values that fits the specified distributions. The set of k normal distributions defines a k-dimensional multinormal distribution.

Returning to mixtures and clusters, by using multiple multinormal distributions (that is, multiple sets of normal distributions of attribute values) we can describe multiple clusters of workload items. The way to do this is as follows.

1. Starting with workload data, cluster it using your favorite clustering algorithm (e.g., one of those described earlier).

2. Use each cluster to define a multinormal distribution. To do so, iterate on the workload attributes. For each attribute, fit a normal distribution to the attribute values of jobs in this cluster. This combination of normal distributions (each with its mean and standard deviation) defines the desired multinormal distribution.
3. Use the EM algorithm to reassign workload items to clusters (or rather, to multinormal distributions) and to adjust the parameters of these distributions. (The EM algorithm is described in Section 4.3.2.)

Note that if the resulting clusters are close enough in all the dimensions, the distributions may overlap. This reflects a more continuous and less modal workload, where the clusters do not stand out so much.

A 1D example was given in Figure 4.6 in Section 4.4.2. The attribute in question was job runtimes. The distribution of job runtimes was multimodal, with some very prominent peaks (many jobs with practically the same runtime) and some wider swells (a collection of jobs with similar runtimes). Using the EM algorithm this distribution was modeled as a mixture of normal distributions.

Modeling a workload based on sampling from a multinormal distribution representing clusters of workload items has the advantage of combining several desirable properties [440]:

- The marginal distributions of the different workload attributes are maintained by the combinations of normal distributions representing each attribute in the different multinormal distributions.
- The common combinations of attribute values that should appear are maintained by using multinormal distributions that dictate specific combinations of distributions for the different attributes. This leads to the correct correlations, which is what we started from.
- It is also possible to incorporate locality of sampling, by controlling the lengths of sequences of samples from the same multinormal distribution.

However, this approach is not very good if workload attributes have continuous distributions rather than modal ones.

6.4.5 Modeling Correlations with Distributions

As noted earlier, many workloads do not cluster nicely — rather, attribute values come from continuous distributions, and many different combinations are all possible. This motivates the use of complete distributions rather than representative points. But the distributions of the different attributes must be correlated in some way.

The direct way to model a correlation between two attributes is to use the joint distribution of the two attributes. This approach suffers from two problems. One is that it may be hard to find an analytical distribution function that matches the data. The other is that, for a large part of the range, the data may be very sparse. For example, most parallel jobs are small and run for a short time, so we have a lot of data about small, short jobs. But

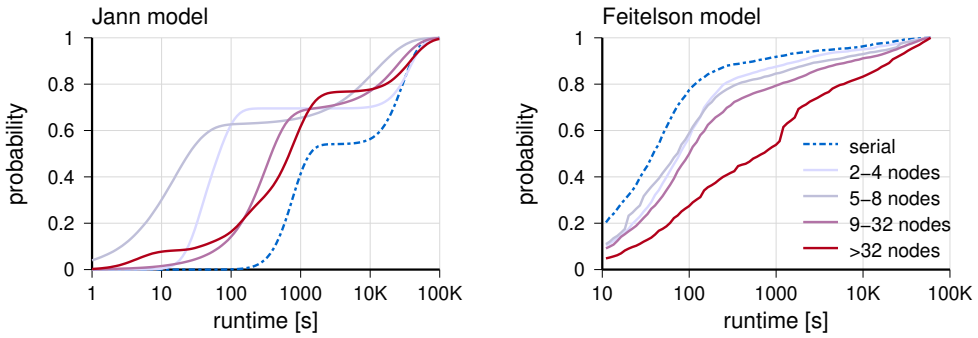


Figure 6.30: *Modeling the dependence of runtimes on job size by using different distributions for different sizes.*

we may not have enough data about large long jobs to say anything meaningful about the distribution — we just have a small set of unrelated samples.

A possible solution is therefore to divide the range of one attribute into subranges, and model the distribution of the other attribute for each such subrange. For example, the Jann model of supercomputer workloads divides the job size scale according to powers of two, and creates an independent model of the runtimes for each range of sizes [370]. These models are completely independent and turn out to be quite different from each other (Figure 6.30 left).

An alternative is to use the same model for all subranges, and define a functional dependency of the model parameters on the subrange. For example, the Feitelson model first selects the size of each job according to the distribution of job sizes; it then selects a runtime from a distribution of runtimes that is conditioned on the selected size [230]. Specifically, the runtime is selected from a two-stage hyper-exponential distribution, and the probability of using the exponential with the higher mean is linearly dependent on the size:

$$p(n) = 0.95 - 0.2(n/N)$$

Thus, for small jobs (the job size n is small relative to the machine size N) the probability of using the exponential with the smaller mean is 0.95, and for large jobs this drops to 0.75. The result is shown on the right of Figure 6.30.

An important advantage of modeling correlation with distributions is that this approach can handle categorical data, which are especially common when the workload is actually a mixture of several workload classes. For example, consider Internet traffic that is generated by multiple applications. Each of these applications has its own distribution of packet sizes, leading to a correlation between the packet sizes and the protocol being used. As the protocols are categorical, Internet traffic is best modeled by simply multiplexing the distributions that characterize the different applications and protocols.

6.4.6 Dynamic Workloads vs. Snapshots

Workload items typically have finite durations. Applications running on a workstation or parallel jobs on a parallel supercomputer have their runtimes. Requests from a web server have their service times. Flows on the Internet also exist only for a certain time. Even files may be deleted after some time.

An important implication of correlations between the duration and any other workload attribute is that the distribution of such attributes depends on how we observe it. The distribution of newly arriving items is different from the distribution of items currently in the system [190].

For example, consider the relationship between runtime and size in parallel jobs. If we look at a job log directly, we see one distribution of sizes. But if we simulate the execution of the jobs, and look at the distribution of sizes during the simulation, we will see a somewhat modified distribution (Figure 6.31). The reason is that long jobs tend to be bigger, and they hang around in the system longer, so they have a higher chance of being observed. In effect, the sizes are weighted by their average runtimes.

Feller calls such effects "waiting time paradoxes" [254, sect. I.4]. Consider buses that arrive according to a Poisson process with exponential interarrival times that have a mean θ. You arrive at the bus stop at some arbitrary time, that is uncorrelated with the last bus. How long should you expect to wait? One answer is that due to the memoryless property of the Poisson process, you can consider your moment of arrival as the beginning of a wait, and therefore expect to wait θ time. But you actually arrived during some interval that comes from a distribution with mean θ, so another answer is that by symmetry you may expect to wait only $\theta/2$ time. The solution is that the second answer is wrong, because the intervals are not chosen uniformly: if you arrive at an arbitrary time, you have a higher chance to fall in a long interval than in a short one. With appropriate weighting of the intervals, their mean turns out to be 2θ, and the symmetry argument then indicates that you should again expect to wait θ time.

The consequence of all these considerations is that one needs to be careful when collecting data. Accounting logs and activity logs, which contain data about each workload item, are more reliable than sampling of active systems. If sampling must be used, then weighting due to different durations should be factored out.

6.5 Correlation with Time

A special type of correlation is correlation with time. This means that the workload changes with time: it is not stationary.

At human time scales, the most commonly encountered nonstationary phenomenon is the daily work cycle. In many systems, the workload at night is quite different from the workload during the day. Many workload models ignore this feature and focus on the daytime workload, assuming that it is stationary. However, when the workload includes items whose duration is on the scale of hours (such as parallel jobs), the daily cycle cannot be ignored [248]. Also, in some cases it is wrong to assume that the daytime workload is more important; for example, in web search workloads the activity of home

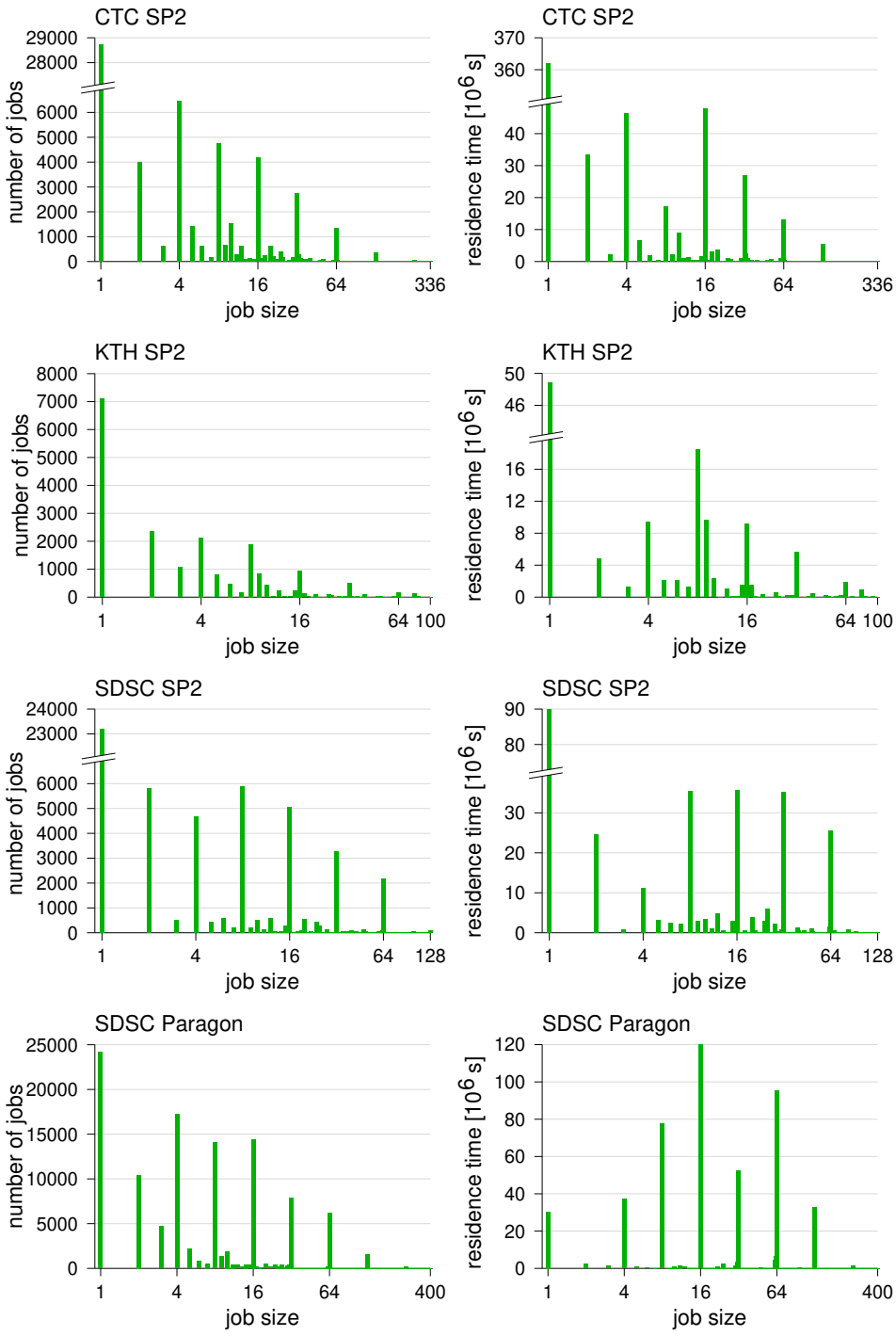


Figure 6.31: *The distribution of job sizes (left) changes if jobs are weighted by their runtime (right).*

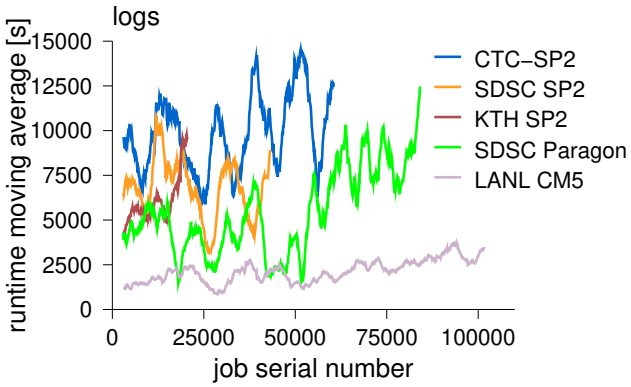


Figure 6.32: *Nonstationarity in parallel job logs, as exemplified by the moving average of job runtimes. The average is computed over a window of 3000 jobs.*

users in the evening and at night is no less important than the activity of professionals during the day [63].

Over long ranges, a nonstationary workload can be the result of changing usage patterns as users get to know the system better. It can also result from changing missions, e.g. when one project ends and another takes its place. Such effects are typically not included in workload models, but they could affect the data on which models are based. In business applications, seasonal effects may also change the workload.

In mathematical notation, nonstationarity is expressed by parameters that are a function of the time t. For example, in a stationary model the arrival rate is denoted by λ, which is a constant. In a nonstationary model we have $\lambda(t)$ (i.e., λ changes as a function of t). Two common types of change are as follows:

- Periodic: the value of λ changes in a cyclic pattern and repeats itself after a period T. An especially simple example is a sinusoidal pattern

$$\lambda(t) = \lambda_0 + \lambda_\delta \sin\left(\frac{2\pi t}{T}\right)$$

  Here λ_0 is the average rate, and λ_δ is the modulation (which should satisfy $\lambda_\delta \leq \lambda_0$ in order to avoid negative values). In general, any periodic function can be used instead of the sinus. In particular, it can be hand-carved to describe a single cycle, as exemplified in the next section.

- Drift: the value of λ grows (or shrinks) with time. For example, a linear drift would be expressed as

$$\lambda(t) = \lambda_0 + \lambda_\delta\, t$$

  where λ_δ is the change per unit time.

Of course, real systems do not limit themselves to simple mathematical models like these. For example, Figure 6.32 shows the moving average of job runtimes in several

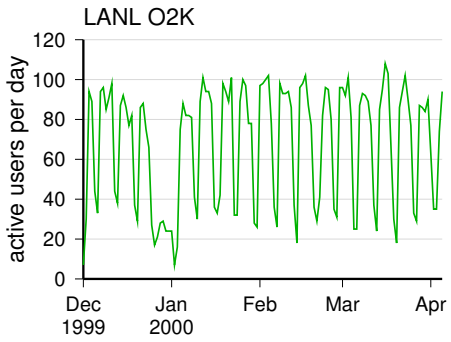


Figure 6.33: *The daily number of active users on a parallel supercomputer shows clear weekly cycles, as well as a yearly effect in the holiday season from Christmas to New Year's Day.*

large parallel supercomputer logs. Some of these display large fluctuations, indicating that the workload is nonstationary. However, they do not follow any clear pattern.

To read more: Correlation with time is the subject of time-series analysis. There are many books on this subject. A concise introduction is given by Chatfield [121]; Another readable exposition is Shumway and Stoffer [619].

6.5.1 Periodicity and the Diurnal Cycle

Cycles are well known in statistics. Perhaps the most common is the seasonal cycle, which as its name suggests reflects yearlong weather patterns. It also relates to many economic endeavors: retail chains may make most of their money in December, whereas seaside resorts make theirs in July and August. For computer workloads, daily and weekly cycles seem to be more important, but yearly effects may also be observed occasionally (Figure 6.33).

The diurnal cycle is a well-known and widely recognized phenomenon, which appears in practically all workloads (e.g., [327, 321, 525, 63, 763, 747, 758, 382, 192, 42]). An example from parallel supercomputers is shown in Figure 6.34. The cycle typically shows higher activity during the day and lower activity during the night. However, details may differ among different types of workloads. For example, in work-related activities one often sees a dip in activity during lunchtime, but in recreational activities such as watching video-on-demand there may be a peak of activity during lunchtime [747]. Also, the peak activity may be in the morning, in the afternoon, or in the evening — as happens for Internet activity at some ISPs, where the peak activity occurs when people return home after work [747, 382].

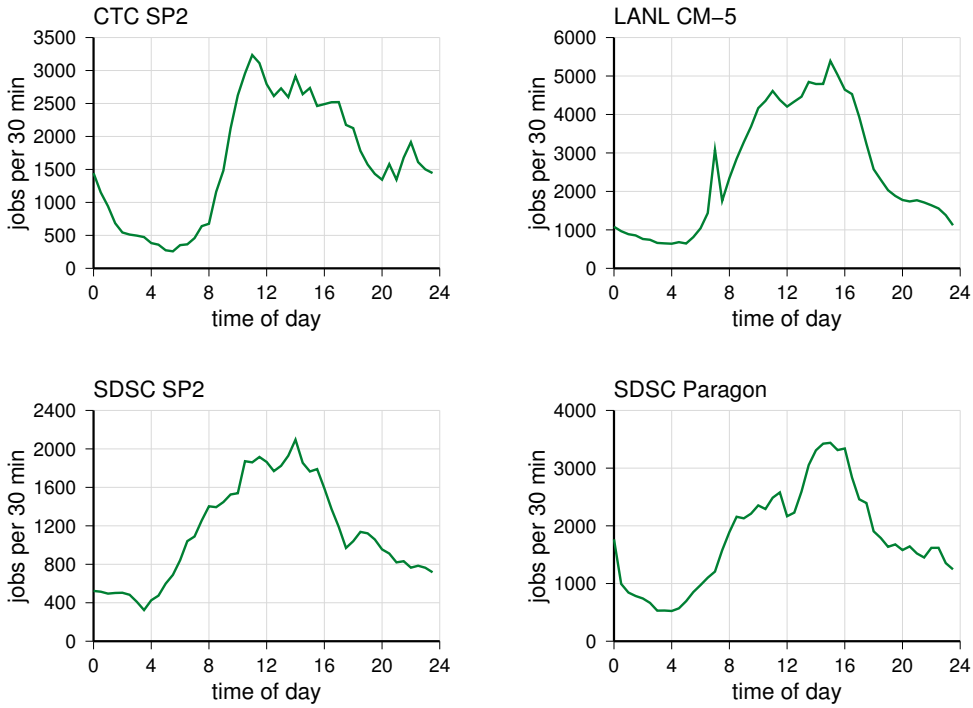


Figure 6.34: *Arrival patterns at four large-scale parallel machines.*

Detecting Periodicity

But what about periodicity in general? Workloads may also contain other periods that are not necessarily related to human activity. Such phenomena may be identified using the workload's autocorrelation.

Background Box: Time Series and Autocorrelation

A time series is a sequence of samples from a random variable, where the index denotes time: $X_1, X_2, X_3, \ldots$. For example, X_1 can be the number of packets that arrived in the first second, X_2 the number of packets that arrived in the next second, X_3 the number in the third second, and so on.

Given such a sequence, its autocorrelation is its correlation with itself, as a function of the relative lag.

Let's do this more slowly. To measure the correlation between two sequences, we want to quantify how similar they are. If the sequences are correlated, they should have large values at the same positions, as well as small values at the same positions. A good way to measure this is as follows. First, center the data. This means that we calculate the mean of the whole series, and subtract it from each sample. We now have a new sequence $Z_i = X_i - \bar{X}$, with values that are both positive and negative. The mean of this new sequence is 0, because

$$\sum_{i=1}^{n} Z_i = \sum_{i=1}^{n} (X_i - \bar{X}) = \sum_{i=1}^{n} X_i - n\bar{X} = 0$$

Second, we perform an element-wise multiplication of the two centered sequences, and find the average of the products. The idea is that, if the sequences are correlated, large values will be multiplied by each other, leading to extra-large products. Because of the centering, small values will be negative, and, when multiplied, will also lead to large (positive) products. Thus the sum (and average) of all the products will be large. But if the sequences are uncorrelated, we will get a mixture of products that are positive (both values are large or small) and negative (one value is large and the other is small). As a result, they will cancel out, and the sum will be close to zero. This is exactly the covariance of the two series, as defined already on page 267. The correlation is the covariance divided by the product of the standard deviations of the two series; this normalizes it to the range $[-1, 1]$.

To derive the *auto*correlation of a sequence, we simply perform this procedure using the same sequence twice. But this is relatively uninteresting, because large values are guaranteed to match up with themselves, and small values with themselves, and the correlation is identically 1. It becomes interesting only if we introduce a lag: one of the copies is shifted by a certain number of locations k. The ith product is then not Z_i^2 but rather $Z_i Z_{i+k}$. The autocovariance at a lag of k is therefore[3]

$$\gamma(k) = \frac{1}{n-k} \sum_{i=1}^{n-k} (X_i - \bar{X})(X_{i+k} - \bar{X})$$

(where we have written the full expression for the Z_is).

The autocorrelation at a lag of k is the same expression divided by the product of the two standard deviations. But the two sequences are one and the same, so this product is just the variance:

$$r(k) = \frac{\frac{1}{n-k} \sum_{i=1}^{n-k} (X_i - \bar{X})(X_{i+k} - \bar{X})}{\mathbb{V}\mathrm{ar}(X)} \tag{6.11}$$

The main use of the autocorrelation function is to find periodicity. If the sequence is periodic, with high values occurring, say, every ℓ samples, then $r(\ell)$ will be relatively large whereas for other values it will be small.

End Box

An example is shown in Figure 6.35. The top part graphs page views of Wikipedia at an hourly resolution over a period of 17 days. The daily cycle is clearly visible, even though activity at night is still very high. The autocorrelation of this series based on a whole year of data is shown below (a graph like this showing the autocorrelation function is sometimes called a *correlogram*). The autocorrelation at a lag of 0 is 1, as may be expected, because the sequence is being compared to itself and therefore produces an exact match. The autocorrelation quickly decays when the lag is increased. At a lag of 12 hours the autocorrelation is negative, because we are matching daytime activity with nighttime activity. But then it peaks again at 24 hours; that is, at a lag of one day.

[3] An alternative form uses a normalization of $\frac{1}{n}$ rather than $\frac{1}{n-k}$, because it has better properties that are relevant when using spectral analysis; in any case, the difference is small for k small relative to n.

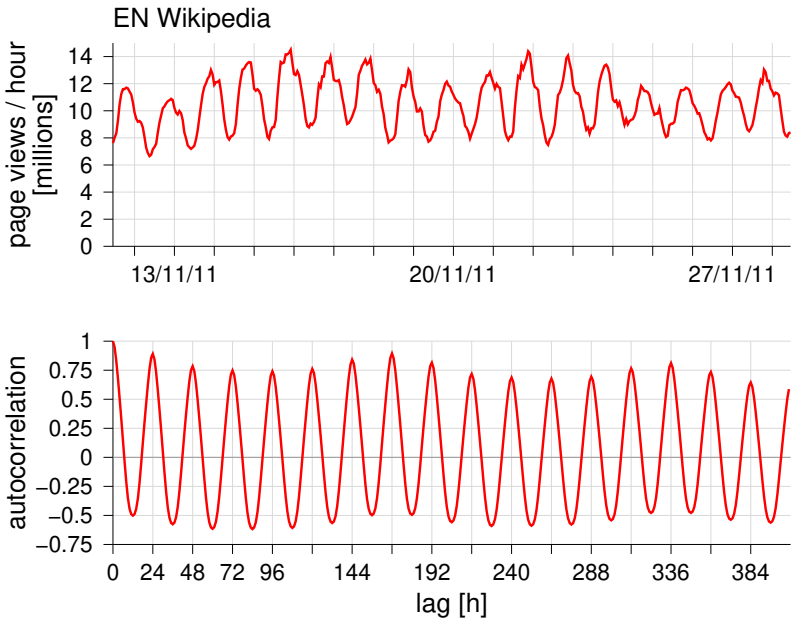


Figure 6.35: *Excerpt from hourly data on Wikipedia page views over time starting at 11 AM on 11/11/11 (top, dates are for Sundays), and the beginning of the autocorrelation function of an entire year of data (bottom).*

It also has peaks at multiples of this value, because the sequence tends to match itself whenever the lag is some integral number of days. One can also pick out the weekly cycle, in which the seventh peak is slightly higher than the others (and again, this repeats at multiples of seven).

The Wikipedia data is very clean in the sense that the daily cycle is clearly visible. As a result the oscillations in the autocorrelation function are also very pronounced. But the autocorrelation function can also extract periodicity from much noisier data. An example is shown in Figure 6.36, which displays the arrivals at the SDSC Paragon machine at an hourly resolution over a period of eight days. The daily cycle is barely discernible (note that at the grid lines denoting midnights the load is always very low). But the autocorrelation of the arrivals in anentire year of the log (shown in the bottom graph) clearly shows peaks at 24-hour intervals. For comparison the average runtimes of the arriving jobs in each hour are also shown. In this case there is a very weak autocorrelation if any, at all lags, and no peaks.

The peaks of the autocorrelation function in Figure 6.36 are rather flat and wide. The reason is that the data itself is like that, exhibiting a wide surge in activity during the daytime and a dip during the night. Thus if we match up 12 noon with 2 PM the next day, rather than matching it with 12 noon the next day (a lag of 26 hours instead of 24 hours), we don't lose much in terms of the correlation.

However, there are other domains in which autocorrelation functions do exhibit sharp

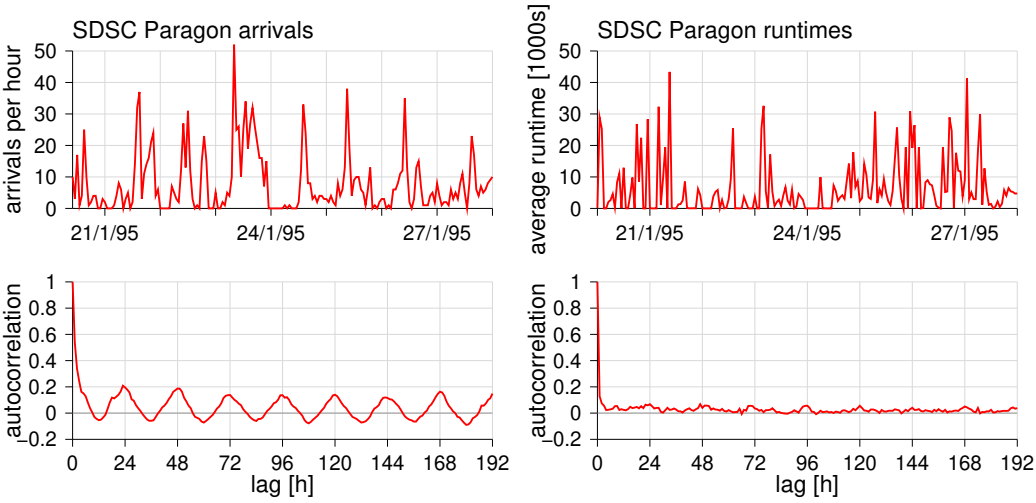


Figure 6.36: *Excerpt from data on arrivals and runtimes of jobs on a parallel supercomputer over time (top), and the autocorrelation functions of the entire series (bottom).*

peaks. This happens if the data has an irregular fine pattern that nevertheless repeats itself. Under such circumstances, only an exact match (that is, a lag of precisely some integral number of cycles) will produce a high correlation.

Practice Box: Calculating the Autocorrelation

The autocorrelation, like the variance (see page 97), can be calculated in one pass on the data. The required running estimate of the mean is calculated as

$$M_n = \frac{n-1}{n} M_{n-1} + \frac{1}{n} X_n$$

and the sum of the products of deviations from the mean at lag k as

$$S_n = S_{n-1} + \frac{n-1}{n} (X_n - M_{n-1})(X_{n-k} - M_{n-1})$$

To obtain the autocovariance divide this by $n-k$, and to obtain the autocorrelation further divide by $\mathbb{V}\mathrm{ar}(X)$ (which should be estimated similarly). Note that this formulation requires the last k samples to be stored, and that only a single lag is given.

To calculate the autocorrelation function efficiently for *all* lags, albeit not online as the data is being read, the FFT algorithm is used [70, 150, chap. 30].

End Box

Categorical Data

The previous material described of how the autocorrelation function works for numerical data, such as the arrival rate of new jobs. However, periodicity may also be apparent with categorical data.

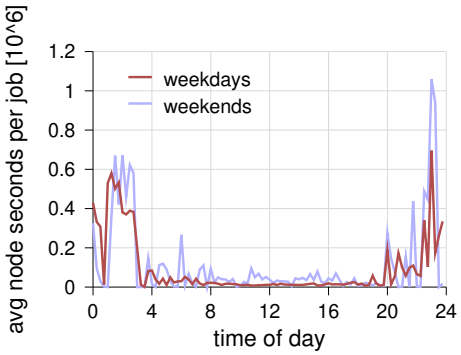


Figure 6.37: *The average job size correlates with time: larger jobs ran at night. Data from the NASA Ames iPSC/860.*

Examples of categorical data abound in workloads. The most prevalent is what type of work is being done. For example, many people start their day with reading and responding to email or reading the news, but do this less often later in the day [382]. An interesting study of web search behavior showed that the popularity of different search topics changes at different times of day, with music searches peaking at 3–4 AM, porn at 5–6 AM, and personal finance at 8–10 AM [63]. The largest shifts in popularity in video-on-demand also occur during the early morning hours [747].

In fact, such correlation with an arrival pattern (and thus, with time) may also occur for numerical data. An example is given in Figure 6.37, which shows the average demand (defined as the node-seconds consumed) of jobs that ran on the NASA Ames iPSC/860. Obviously, large jobs ran only at night, whereas daytime hours were dominated by relatively small jobs.

Modeling the Daily/Weekly Cycle

There are three main approaches for dealing with daily and weekly cycles. One is to divide the day into a number of ranges and model each one separately, assuming that it is stationary. Another is to find a function that models how the workload changes with the time of day. The third is to use a generative model that includes a daily cycle.

Partitioning the day into a number of stationary periods has the advantage of also being easily applicable to categorical data. For example, the following partitioning into two or three relatively homogeneous intervals has been proposed [131]:

- Low rate from midnight to 8 AM on workdays and midnight to 9 AM on weekends.
- Intermediate rate from 5 PM to midnight on workdays and from 9 AM to midnight on weekends.
- High load from 8 AM to 5 PM on workdays.

If this is insufficient, a finer resolution of one hour can be used [540, 42].

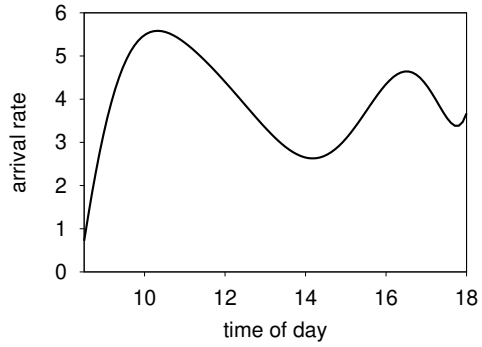


Figure 6.38: *Daily arrival pattern according to the model of Calzarossa and Serazzi.*

The other approach is to use a parameterized distribution, and to model the daily cycle by showing how the parameters change with the time of day. In particular, interarrival times can be modified such that more arrivals occur during prime time, and less at night [321, 454]. This first requires a model of the level of activity as a function of the time of day.

Calzarossa and Serazzi have proposed a polynomial of degree 8 for this, which captures the variations between the morning and the afternoon [102]. The proposed polynomial for "normal" days is

$$\lambda(t) = 3.1 - 8.5t + 24.7t^2 + 130.8t^3 + 107.7t^4 - 804.2t^5 - 2038.5t^6 + 1856.8t^7 + 4618.6t^8$$

where $\lambda(t)$ is the arrival rate at time t, and t is in the range $[-0.5..0.5]$ and should be scaled to the range from 8:30 AM to 6:00 PM; this is shown in Figure 6.38. This expression represents the centroid for a set of polynomials that were obtained by fitting measured results for different days. Slightly different polynomials were discovered for abnormal days, in which the administrative office closed early, or were the first day after a weekend or a holiday. The model is naturally smoother than the data shown above. It has a pronounced peak in the morning hours and a large dip at lunchtime. Lublin has proposed a simpler model, based on a gamma distribution, shifted so that the minimum occurs at 5 AM [454].

Given a functional description of the arrival rate for different times of day, one still needs to generate the arrivals themselves. Normally, with a homogeneous model, one would generate interarrival times that are inversely proportional to the arrival rate. With a uniform arrival rate, these interarrival times are exponentially distributed (as shown in Section 3.2.1). So if the previous arrival occurred at time t, and the arrival rate is λ, we can generate a uniform random variate u and set the next arrival to be at $t - \frac{1}{\lambda}\ln(u)$. (Because $u < 1$ we have $\ln(u) < 0$, so the sign is correct.)

The problem is that, when arrival rates change, a low arrival rate does *not* imply that additional arrivals should not occur until a long time into the future, because the arrival rate my increase shortly thereafter. Rather, a low arrival rate means that there is a low probability for an arrival *now*. In principle we can achieve this with a time-based

model, where for each time unit an arrival is generated with a probability that is inversely proportional to the momentary arrival rate. However, this has high overhead because we need to consider each time instant, at the finest resolution, and most of them will be empty (for example, if the arrival rate is one every 3threeseconds on average, and we are working at a resolution of milliseconds, there will be one arrival every 3000 time units on average).

A better solution is to generate arrivals at the maximal possible rate and reject the extra ones. Denote the arrival rate at time t by $\lambda(t)$, and the maximal arrival rate ever as $\lambda^* = \max_t \lambda(t)$. We then generate arrivals uniformly at a rate of λ^* as described earlier. But we don't use all of these arrivals. Instead, for an arrival that occurs at time t, we use it with probability $\lambda(t)/\lambda^*$. If λ^* is not too high relative to the average arrival rate, the overhead for wasted arrivals is acceptable.

A potentially better alternative is to use a user-based generative model, in which the daily and weekly cycles are created by user sessions that are synchronized with these cycles. For example, Zilber at al. propose a model with four user classes, corresponding to combinations of users who are active at day or at night, and on weekdays or on weekends [763]. Each user class has its characteristic combination of sessions types, and there are more daytime users than nighttime users, leading to a daily cycle of fluctuations in the load. Shmueli and Feitelson elaborate a model for daytime users. In their model, users have a propensity to leave the system around 6 PM and return between 8 and 10 AM the next day [616, 248]. This is done by keeping track of wallclock time when generating new arrivals. When the arrivals model generates a new arrival that should occur too late, the daily cycle model modifies it to shift it to the next day.

6.5.2 Trends

Nonperiodic correlation of a workload with time typically reflects the evolution of workload attributes. In some cases, this evolution can be a sequence of discrete changes, as may occur when users move from one project to another. There is not much to say about such changes from a modeling perspective. We therefore limit this discussion to cases where the evolution of the workload follows a distinct trend.

An example of workload data with a trend is shown in Figure 6.39. This is the activity experienced by the 1998 soccer World Cup website in the weeks preceding the start of the tournament. As the beginning of the games drew closer, more and more users visited the website in anticipation. As is often the case, the trend is superimposed by a daily and weekly cycle. To analyze the cycles, we need to characterize and remove the trend.

In order to observe the trend itself, without being distracted by the details of the daily cycle, it is best to work at the granularity of complete cycles. To do so, we simply sum over each 24-hour cycle (Figure 6.40). We can then take the differences between these daily values. Although the differences fluctuate around zero, one should take care not to jump to conclusions regarding their behavior. First, it is obvious that the average difference must be positive, because the original data contains an unmistakable upward trend. We therefore perform a linear regression on the differences, to find whether they

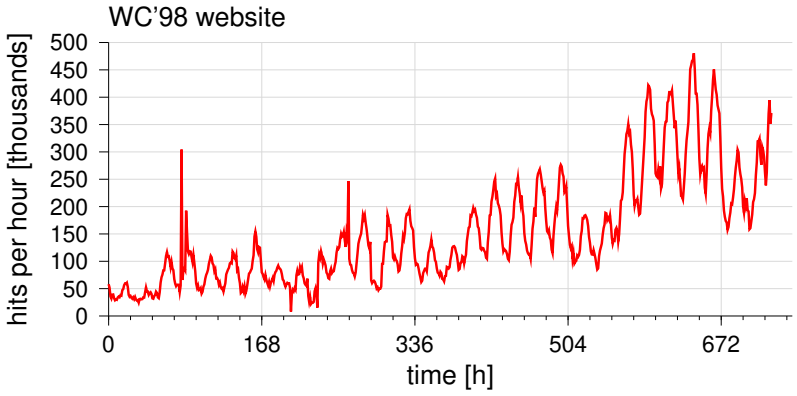


Figure 6.39: *Hits on the WC'98 website show a distinct increasing trend as the start of the tournament approaches.*

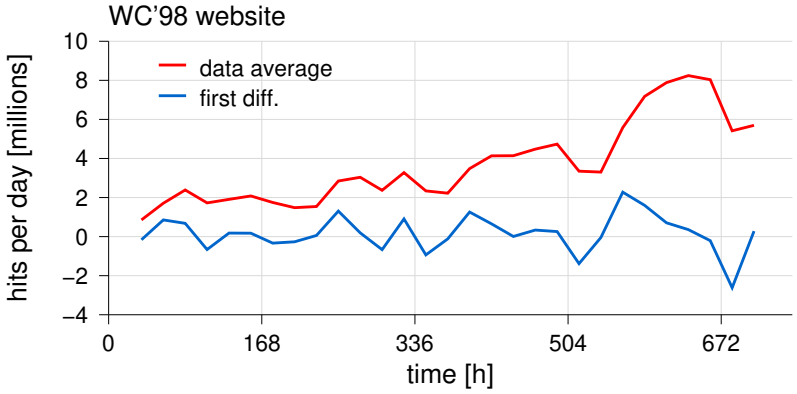


Figure 6.40: *The data from Figure 6.39 after summing over 24-hour intervals, and taking first differences.*

too display some trend. If they are well modeled by a horizontal line, then first differences suffice, and the trend in the original data is a linear trend. But if the differences also display an upward trend, we should in principle check the second differences and consider a quadratic model for the original data.

In our case, the slope of a regression line representing the first differences is -154, which is minuscule considering the values involved (which are in the millions). Likewise, the correlation coefficient of the first differences with the time is only 0.034 (the correlation coefficient of the original data with the time is 0.868). Thus it indeed seems that we can use a linear model for the trend.

After removing the linear trend, we are left with the daily cycle, the weekly cycle, and random fluctuations (Figure 6.41). The analysis can continue based on the observa-

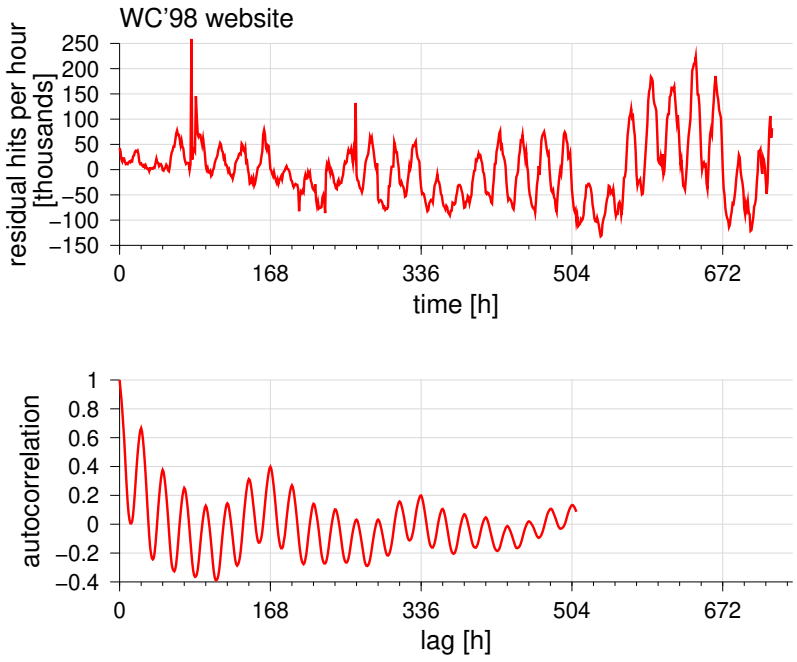


Figure 6.41: *The data after removing the trend, and its autocorrelation function. 168 hours are one week.*

tion that the amplitude of the daily and weekly cycles grows with time. This implies a multiplicative effect, which can be turned into an additive effect by using a logarithmic transformation. Such procedures are covered in texts on time series analysis; this is not pursued further here.

However, it may be instructional to compute the autocorrelation function again, as we did in the previous section. Here the data is much cleaner, so the peaks of the autocorrelation are much higher. In addition, there is a bigger difference between weekdays and weekends, so the seventh peak is noticeably higher than the others. The sixth and eighth are also higher than the third, fourth, and fifth, because the weekend is two days long, and matching one of them is better than not matching either.

7

Self-Similarity and Long-Range Dependence

Self-similarity and long-range dependence are, formally speaking, distinct phenomena. However, in practice they typically come together. Self-similarity is about scaling: the workload includes bursts of increased activity, and similar-looking bursts appear at many different time scales. Thus the workload appears similar to itself when viewed at a different scale (e.g., at a resolution of minutes rather than at a resolution of seconds). Long-range dependence is about correlations: what happens now is correlated to what happened a moment ago, and actually also with what happened in the more distant past. Of course, the correlation with the past does in fact decay with time. However, it decays slowly, and thus effects accumulate over a long period. As a result long-range correlations may create the observed bursts of activity.

Even more so than heavy tails, these are advanced topics in statistical modeling that are typically not encountered at the introductory level. In fact, the mathematical sophistication of these topics is significantly higher than any other material in this book. But these topics reflect real-life situations and cannot be ignored. In the interest of promoting understanding, we emphasize gaining an intuition of what the different definitions mean. This material is usually followed by some mathematical derivations, at least in outline.

The domain in which self-similarity and long-range dependence are encountered is the arrival process: how work arrives at the system. This chapter begins by reviewing Markovian arrival processes, such as the Poisson process. It then contrasts them with the phenomena of self-similarity and long-range dependence.

To read more: There is now quite a bit of literature regarding self-similarity and long-range dependence in workloads. A good place to start is the book edited by Park and Willinger [537], in particular the introductory chapter by the editors that reviews the field and the underlying mathematics [538]. Surveys include Cappé et al. [106], Abry et al. [3], and Samorodnitsky [589]. Barford has collected an annotated bibliography of the most important papers on the subject from the late 1990s, which can be found at URL http://www.cs.bu.edu/pub/barford/ss_lrd.html.

The study and modeling of self-similarity were actually started in hydrology, and later applied to other geophysical fields [469]. Another field in which self-similarity is in wide use is

economics — based on the premise that if you can predict market fluctuations you can make money. Books on econometrics are therefore good sources on the subject. In particular, Peters provides a good introduction [545].

For the more mathematically inclined, detailed treatments have been written by Samorodnitsky and Taqqu [590] and by Beran [67].

7.1 Poisson Arrivals

A Markovian arrival process is based on a Markov chain: it can be in any of a given set of states, and can move from one state to another with certain probabilities (for background on Markov chains, see page 242). Certain transitions of the Markov chain are associated with "arrival events", meaning that, when such a transition happens, a new arrival is generated by the model. As a result, interarrival times have a phase-type distribution. The complexity of the model, and the variability of the generated workload, depend on the state space of the underlying Markov chain and which transitions correspond to arrivals.

7.1.1 The Poisson Process

The simplest Markovian arrival process is the Poisson process. It has only one state, so arrivals are generated at a constant rate. It is defined more formally as follows (repeating the definition in Section 3.2.1).

Consider a period of time T during which events occur at an average rate of λ events per time unit. Partition the period T into very many small intervals. To qualify as a Poisson process, the following three conditions must hold:

1. There is no more than a single event in each interval (of course, many intervals will have no events in them).
2. The probability of having an event is the same for all intervals.
3. The existence of an event in a specific interval is independent of whatever happens in other intervals.

Given a dataset (e.g., arrivals of jobs at a parallel supercomputer, or arrivals of packets at a network router), an important question is whether these arrivals can be modeled as a Poisson process, or perhaps another more complicated model is required. This chapter focuses on those other models, but first let us consider tests that can be applied to the data to determine whether a Poisson model is appropriate.

The obvious approach is to test for the salient features of Poisson processes. In Section 3.2.1 we showed that the interarrival times of a Poisson process are exponentially distributed, with a mean of $1/\lambda$ time units. It is therefore possible to check whether arrivals in a log conform to a Poisson process by verifying that their interarrival times are exponentially distributed and independent. These properties can each be checked individually [540].

To check that interarrival times are exponentially distributed, the best parameter value θ is estimated as the mean interarrival time. The data is then compared with an exponential distribution with parameter θ using the Kolmogorov-Smirnov (or some other) goodness-of-fit test. To check for independence we can use the autocorrelation function. The test is very simple: if the interarrivals are independent, there should be no correlation between them. So it is enough to check the autocorrelation at lag 1 (the "serial" correlation) and see that it is close to zero [320]. Additional tests for a Poisson process are listed below in Section 7.4.1.

Another option is to use a graphical method. Start by counting the number of arrivals in successive time units (e.g. seconds). Such counting defines a series of random variables X_i, where X_1 is the number of arrivals in the first second, X_2 the number of arrivals in the second second, and, in general, X_i is the number of arrivals in the ith second. Such a series is called a time series, because the index denotes time. Now aggregate this series. This means that we compute the sums of non-overlapping subseries. For example, we can sum up the first 10 values, the next 10 values, and so on. Doing so gives us the same data at a coarser granularity: instead of arrivals per second, we now have arrivals per 10 seconds. And we can aggregate again and again, to get arrivals per 100 seconds and arrivals per 1000 seconds.

Figure 7.1 shows what happens when we perform such aggregation on a Poisson process with a rate of $\lambda = 5$. The top-left graph shows that at a fine granularity arrivals are bursty. Although there are five arrivals per second on average, some seconds have more arrivals and others have fewer. But as we aggregate the data these deviations tend to cancel out, because they are independent. Therefore the aggregated data is much smoother. As we see later, this smoothing does not happen (or rather, happens much more slowly) for self-similar data, where the arrivals in successive seconds are correlated.

7.1.2 Nonhomogeneous Poisson Process

Before we start with self-similarity, it is appropriate to briefly describe a couple of more advanced Poisson-based processes. The first is the nonhomogeneous Poisson process. In this model, the arrival rate λ changes with time. This process is intended to capture situations in which the assumption of a constant arrival rate is inappropriate.

Nonhomogeneous Poisson processes have been proposed for modeling the daily cycle of activity (see Section 6.5.1). One way to do so is to find a function that models the arrival rate as a function of the time of day [102, 454]. This changes the requirement that the probability of an arrival event be the same for all intervals, and replaces it with different probabilities that reflect the different arrival rates. Another option is to assume that the change is not continuous, but instead that different λs apply for different periods, such as morning, evening, and night [540, 131]. Note that this is different from having a Markovian process with several states with different λs, because the value of λ is determined by the time, and not randomly by the underlying Markov chain.

Another class of nonhomogeneous Poisson processes includes those that are not periodic, such as the Markov-modulated Poisson process (MMPP). This model includes an underlying Markov chain, and when this chain is in state i, arrivals occur at a rate λ_i

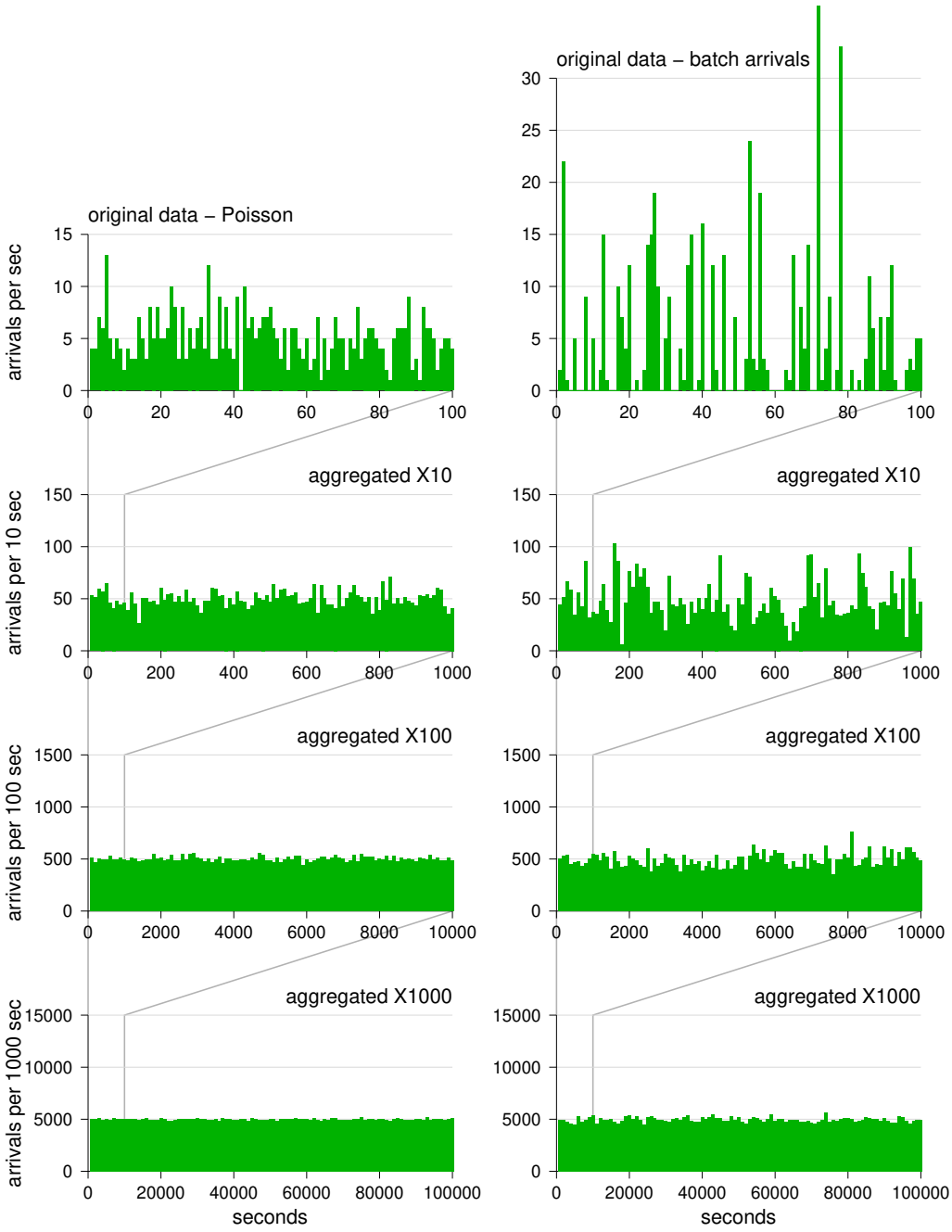


Figure 7.1: *A Poisson process shown at different levels of aggregation. Left: a simple Poisson process with mean arrival rate of 5 arrivals per second. Right: a process with batch arrivals. Batch sizes are exponentially distributed with a mean of 5, and the arrival rate is reduced by a factor of 5 to compensate.*

[262]. Nonhomogeneous Poisson processes can also be used to model trends or workload evolution in general.

7.1.3 Batch Arrivals

Another extension of Poisson processes is to include batch arrivals. This increases the burstiness of the arrivals by postulating that upon each arrival, not one but several items will arrive together. The number of arrivals in each batch is selected from a distribution that is also part of the model.

For example, batch arrivals may be used to model a situation in which customers arrive by bus. The arrivals of buses is a Poisson process, but each bus brings with it an entire batch of customers. In the context of computer workloads, such a model may be used for requests arriving at an HTTP server. When a web page is downloaded, it typically generates a set of additional requests to download embedded objects (e.g. graphics) from the same server. These separate requests can be modeled as a batch of requests that arrive together.

Although batch arrivals obviously increase the burstiness of arrivals, they still fail to capture the type of burstiness that is actually observed in many arrival processes. As shown in Figure 7.1, batch arrivals cause increased burstiness only at high resolution, but they nevertheless average out when aggregated. In self-similar data, the arrivals in successive seconds are correlated, and do not average out when aggregated. This is the subject of the rest of this chapter.

7.2 The Phenomenon of Self-Similarity

The Poisson process shown in Figure 7.1 looks different at different scales. It looks unchanging on a global scale, but bursty when we zoom in to investigate the short-term fluctuations. A self-similar process, in contradistinction, looks nearly the same at all scales.

Note that when we plot an arrival process, as in Figure 7.1 and Figure 7.3, the X axis represents time and the Y axis represents intensity (arrivals per time unit). A central question is what should be the relative scaling of these two axes. So far we have assumed that they are both scaled together by the same factor: when we aggregate by a factor of 10, we also increase the intensity scale by a factor of 10. Such scaling is crucial in making a Poisson process appear to be smooth upon aggregation. As we see later, the definition of self-similarity (or more precisely, the degree of self-similarity) actually depends on the relative scaling of the two axes.

7.2.1 Examples of Self-Similarity

Self-similarity refers to situations in which a phenomenon has the same general characteristics at different scales [466, 602]; that is, if we zoom in, we will see the same structure as we did before. This implies that parts of the whole are actually scaled-

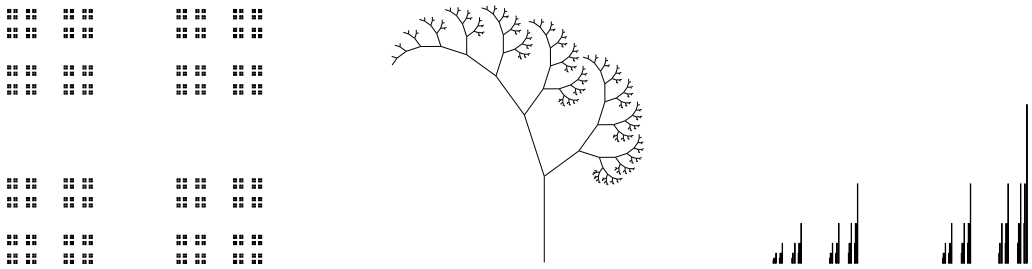


Figure 7.2: *Examples of fractals: a 2D Cantor set, a tree in which each branch and sub-branch are similar to the whole, and a 1D Cantor set with unequal weights that begins to look like an arrival process.*

down copies of the whole. This feature is the source of the name "self-similar": the phenomenon is similar to itself at a different scale.

Probably the best-known self-similar objects are fractals, geometrical constructions such as those shown in Figure 7.2. Note that these fractals are purely mathematical objects, and that their self similarity is precise and complete: the object contains exact copies of itself at a smaller scale.

Background Box: Fractals

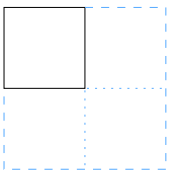
Fractals are geometric objects that have the following two (related) properties: they are self-similar, and they do not have a characteristic scale.

Being self-similar means that parts of the object are similar (or, for pure mathematical objects, identical) to the whole. If we enlarge the whole object we end up with several copies of the original. This is also why there is no characteristic scale — it looks the same at every scale.

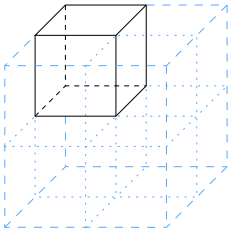
The reason they are called fractals is that they can be defined to have a fractional dimension. This means that these objects fill space in a way that is different from what we are used to. To explain this, we first need to define what we mean by "dimension".

Consider a straight-line segment. If we double it, we get two copies of the original.

If we take a square, which is composed of four lines, and double the length of each of them, we get a larger square that is equivalent to four copies of the original.



For a cube (which is composed of 12 line segments), doubling each one creates a larger cube that is equivalent to 8 copies of the original.



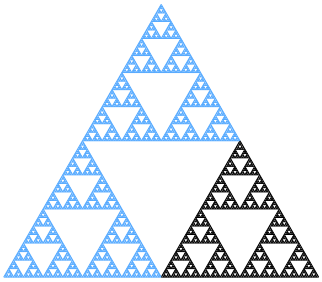
Let us denote the factor by which we increase the length of the line segments by f, and

the number of copies of the original that we get by n. These three examples motivate us to define dimensionality as

$$d = \log_f n$$

With this definition, the line segment is one-dimensional ($\log_2 2 = 1$), the square is 2D ($\log_2 4 = 2$), and the cube is 3D ($\log_2 8 = 3$). The intuition is that we multiply by a factor of f in each dimension, and therefore the number of copies we get is f^d.

Now apply the same definition to the endlessly recursive Sierpinski triangle. Doubling each line segment by two creates a larger triangle which contains three copies of the original. Using our new definition, its dimension is therefore non-integral: $\log_2 3 = 1.585$. This motivates naming it a fractal. A nice feature is that the dimension comes out between 1 and 2: it is more than a line, but does not quite fill out the plane.



While the above is a purely mathematical construction, examples of fractals from nature abound [466]. They span the range of sizes from minute crystal formations to planetary landscapes, including sea shells, composite leaves, and lungs along the way. Fractals also occur in human-made artifacts, from Jackson Pollock's drip paintings [680] to music [68], stock market stock prices [545], and the scientific literature [207].

To read more: There are other definitions of fractal dimensions, with various mathematical justifications. For example, if you enlarge the tree in Figure 7.2 you do not get an integral number of replicas of the same size, but rather a set of replicas at different sizes plus some extra stems. For a detailed treatment of fractals, complete with anecdotes and nice graphics, see the books by Mandelbrot [466] and Schroeder [602].

End Box

Self-similarity also occurs in nature. Of course, in natural phenomena we cannot expect perfect copies of the whole, but we can expect the same statistical properties. Maybe the most famous natural fractal is the coastline of Britain, which is reputed to have inspired Mandelbrot's investigation of the subject [465, 466]. The self-similarity appears when we try to measure its length using yardsticks of decreasing length. The shorter the yardstick, the more details we can observe, and the longer the coastline becomes. Thus we cannot really characterize it with a length; rather, we can characterize it with a fractal dimension, which describes how the length grows as a function of the yardstick.

Workloads often also display such behavior. The first demonstrations of self-similarity in computer workloads were for LAN traffic, and used a striking visual demonstration (reproduced in Figure 9.20) [436]. A time series representing the number of packets transmitted during successive time units was recorded. At a fine granularity (i.e., when using small time units), this series was seen to be bursty. But the same bursty behavior persisted also when the time series was aggregated over several orders of magnitude, by using larger and larger time units. This contradicted the common Poisson model of packet arrivals, which predicted that the traffic should average out when aggregated.

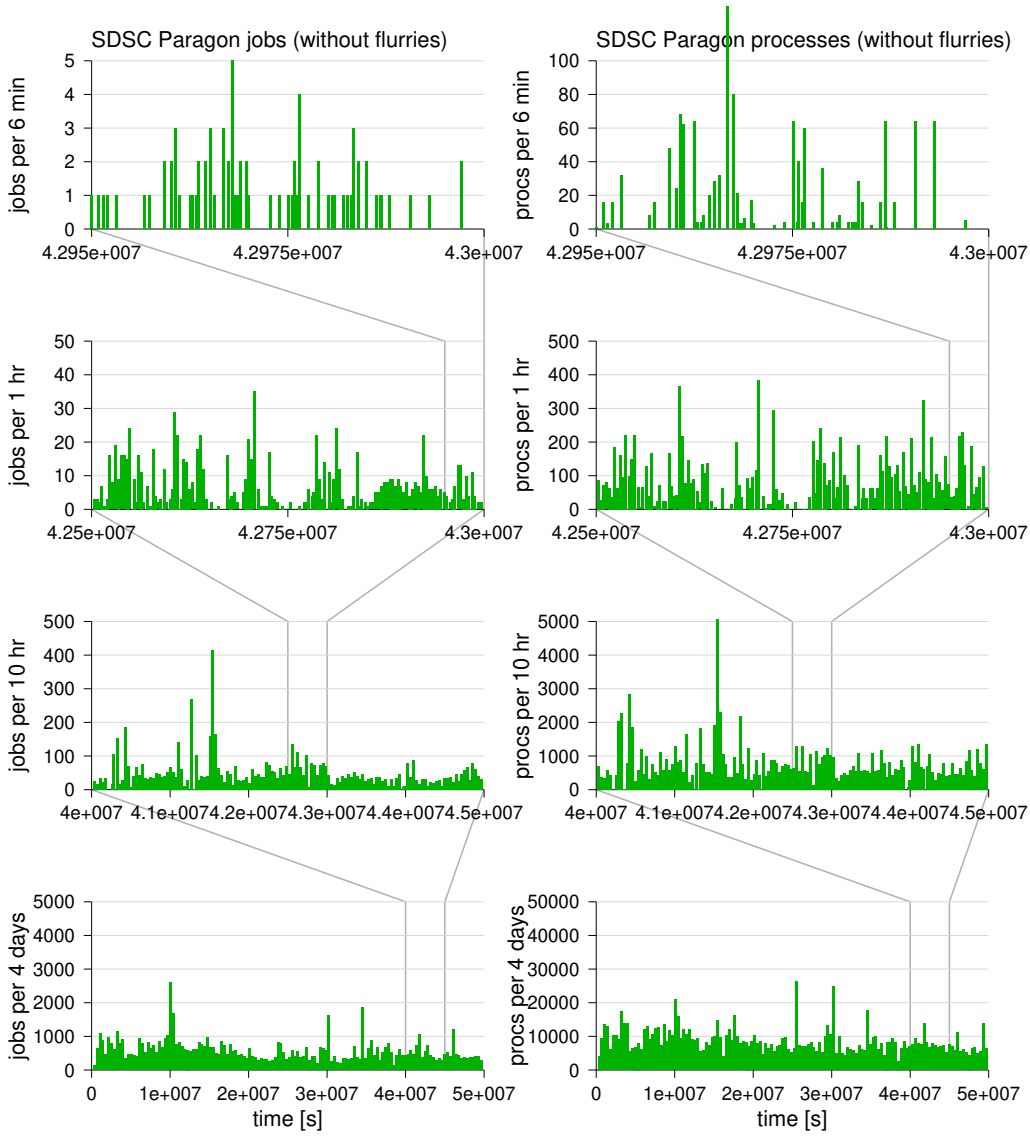


Figure 7.3: *Job and process arrivals at the SDSC Paragon parallel supercomputer shown at different levels of aggregation. Each parallel job is composed of many processes that execute concurrently. In all the graphs time is in seconds; the duration of the full log is two years, which is about 63 million seconds. Compare with the Poisson process shown in Figure 7.1.*

Similar demonstrations have since been done for other types of workloads, including the traffic in other networks, e.g. WANs [540]. Figure 7.3 gives an example of jobs arriving at a parallel supercomputer (and see also [653, 670]). Self-similarity has also been shown in file systems [304], web usage [155], and video traffic [69], to name a few.

7.2.2 Self-Similarity and Long-Range Dependence

As shown in Figure 7.3, self-similarity in workloads manifests itself in the form of burstiness. Burstiness implies a highly variable load, which fluctuates between low loads and high loads. The self-similarity means that not only do such fluctuations occur, but that they also occur at all time scales. This implies that at the coarser time scales the fluctuations also have to be bigger.

But burstiness also occurs in the Poisson process, at least when we zoom in and observe it at a fine resolution. So what is the difference? How does a self-similar process retain its burstiness when we aggregate it over several orders of magnitude?

The answer is that the samples in the self-similar process are not independent. Quite the contrary — they are correlated with each other, and the correlation spans multiple time scales. Not only are successive samples correlated to each other, but also samples that are relatively far away from each other tend to be correlated.

As a result of this correlation, samples tend to come in long sequences of samples that are similar to each other. Thus if a sample deviates from the mean, chances are that it will be surrounded by other samples that deviate from the mean *in the same direction*. Intuitively, this means that long sequences of similar samples tend to occur, and that these sequences are further grouped into super-sequences. This intensifies the deviation from the mean and leads to the bursty nature of the workload.

In mathematical terms the correlation among far-away samples is called "long-range dependence". Of course, sequences of similar samples may also occur by chance in processes where each sample is independent. For example, if you toss a coin many times, you may see sequences of throws that come out heads one after the other. When using a fair coin, the lengths of such sequences are exponentially distributed. In long-range dependent processes, the observed similar sequences tend to be much longer than would be expected if they came from an exponential distribution. In fact, they come from a heavy-tailed distribution.

So are self-similarity and long-range dependence just two faces of the same phenomenon? The answer is no. In fact, self-similarity can arise from either of two sources, which Mandelbrot picturesquely called the Noah effect and the Joseph effect [466, p. 248]. The Noah effect refers to unique large-scale events, as happens when the original (high resolution) samples are heavy-tailed. As we saw in Chapter 5, summing heavy-tailed random variables does not average out, but rather leads to a heavy-tailed sum. Thus when a process composed of heavy-tailed samples is aggregated, we will get a process with similar statistics. In other words, it will be self-similar.

The other possible source of self-similarity, the Joseph effect, is long-range dependence. As described above, this means that the events at a given time are correlated with events that happened a long time before. This correlation prevents the averaging out of

the process when it is aggregated, because the aggregation sums many samples that are similar to each other: they are either mostly larger or mostly smaller than the mean.

The (mathematical) definitions of self-similarity, heavy tails, and long-range dependence are independent of each other. In general we may have one without the other, and in particular, one can construct self-similar processes that are not long-range dependent. But in the practical situations of interest these two phenomena turn out to coincide, and the self-similarity is typically a result of long-range dependence, and not a result of heavy-tailed distributions [538].

7.2.3 The Importance of Self-Similarity

The effect of the high variability that is associated with self-similarity has been studied mostly in the context of communication networks. Communication networks have often been analyzed using Poisson-related models of traffic, which indicate that the variance in load should smooth out over time and when multiple data sources are combined. This property allows for precise evaluations of how much buffer space is needed to support communications at a given average rate, as well as for the design of algorithms for provisioning different levels of service to different users.

But in 1994 Leland and co-workers showed, based on extensive observations and measurements from Ethernet LANs, that the load does not smooth out in practice [436]. Instead, they found that when traffic streams are aggregated or merged, they retain a high level of burstiness. This behavior was consistent with self-similar models, and such models were soon after shown to apply to other types of communications as well [540, 155, 69].

Using self-similar workload models that capture the burstiness of network traffic affects the results of performance evaluations. For example, such evaluations lead to larger and more realistic estimates of required buffer space and other parameters [214]. They also cast doubt on the feasibility of various schemes for providing guaranteed levels of service, because the high variability of the load prevents the system from being able to estimate how much extra capacity is available [538].

Connection Box: Hydrology and Computer Networking

Two names are often associated with the development of the field of self-similarity: Harold Edwin Hurst, who provided empirical data for many natural phenomena, and Benoit Mandelbrot, who interpreted this data as reflecting long-range dependence.

The work of Hurst is especially interesting because of its close analogy with current models of computer communications. Hurst was a hydrologist, who spent much of his professional life in Egypt, studying the Nile River. His aim was to design a reservoir that would guarantee a steady flow of water, thereby overcoming yearly fluctuations in the rainfall over the river's basin. To do so, the reservoir had to be big enough to hold excess water from years with lots of rain, and make up for the deficit in drought years. Using data spanning 1305 years (from 641 to 1946) Hurst showed that the fluctuations from the average were much larger than what would be expected if the amount of water available each year was an independent random variable [354]. Rather, rainy years and drought years come in long sequences, as in the biblical story of Joseph (which is why Mandelbrot later called this the

Joseph effect). This meant that a larger reservoir would have to be built — a reasoning that eventually led to the construction of the high dam at Aswan in the 1960s.

Exactly the same considerations apply in computer communication networks. A router connects several links with given maximal bandwidths. To optimally use this bandwidth, the router should have buffers that are large enough to store excess data that arrives when the output link is overly congested, thus allowing the data to be transmitted later when the load abates. And indeed, one finds that the required buffers are larger than would be expected if packet arrivals were independent random events. Although Hurst probably did not foresee this specific analogy, the abstract of his 1951 paper ends with these words [354]:

> It is thought that the general theory [of scaling as presented in the paper] may have other applications than the design of reservoirs for the storage of water.

End Box

At a more abstract level the phenomenon of self-similarity implies that the notion of a "steady-state" workload is questionable. Assuming a steady state is the foundation of queueing network models and many other types of analysis. But if significant fluctuations may occur at many different time scales, the assumption of stability is violated. This casts a shadow not only on analyses that assume stability, but also on simulations that claim to use a single representative workload configuration. A better approach may be to try and capture the heterogeneity of the workload, and the resulting diversity of system behaviors and performance results [264].

7.2.4 Focus on Scaling

The figures commonly used to portray self-similarity, like Figure 7.3, are somewhat misleading. It is not true that self-similar processes are bursty and a Poisson process is not. The different processes are actually all quite similar. The difference lies only in their scaling behavior.

In both Figure 7.1 (the Poisson process) and Figure 7.3 (the "self-similar" process), the scale used in the different graphs grew in direct proportion to the level of aggregation. Thus in the Poisson process the scale at the most detailed view was 0 to 10, after aggregating by a factor of 10 it was 0 to 100, and after aggregating by a further factor of 10 it was 0 to 1000. Using such scales, it was possible to see that the average values grow in direct proportion to the level of aggregation, as may be expected.

However, it is hard to quantify how the variance (or rather, the standard deviation) changes with aggregation. To get a better view, we show the same data again in Figure 7.4, but this time the scale is set to fit the data. What we find is that in all cases the variability actually *grows* with aggregation, and moreover, the Poisson process looks self-similar when viewed at this vertical scale! However, the variability grows much more slowly than the average. As a result the *relative size of the variability*, when compared to the average as in the original figures, seems to decrease to the point where the process looks completely stable.

By setting the scale to match the observed variability, we can get a rough estimate of how the variability scales. The results are as follows:

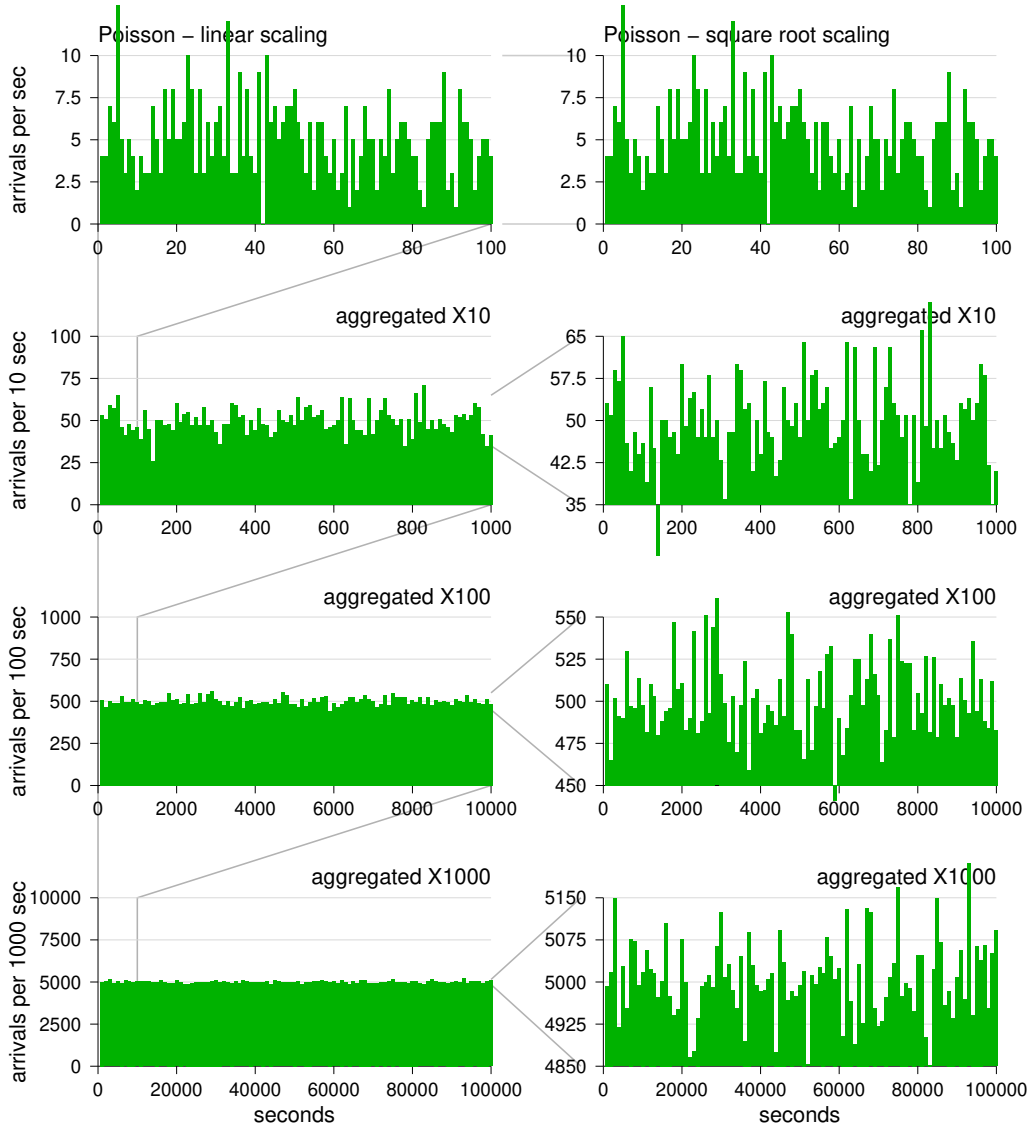


Figure 7.4: *The Poisson date from Figure 7.1 on the left, and the same data again on the right but shown at a different scale — namely a scale that corresponds to the variability of the data.*

Aggregation	*Range*		*Relationship*
1	5 ± 5	$\approx$	$5 \times (1 \pm \sqrt{1})$
10	50 ± 15	$\approx$	$5 \times (10 \pm \sqrt{10})$
100	500 ± 50	$\approx$	$5 \times (100 \pm \sqrt{100})$
1000	5000 ± 150	$\approx$	$5 \times (1000 \pm \sqrt{1000})$

Thus we see that when we aggregate by a factor of g, the average grows by a factor

of g as well, but the variability grows only by a factor of $\sqrt{g}$. Hence if we draw the process on a linear scale the variability seems to decrease as $\frac{1}{\sqrt{g}}$. This result is in fact to be expected. The Poisson distribution (which is the distribution of the actual number of arrivals in each time unit of a Poisson process) has the property that its variance is equal to its mean. Therefore its standard deviation, which is reflected by the variability we see, is equal to the root of the mean.

In the self-similar process of Figure 7.3 the standard deviation also grows, and again, it grows more slowly than the average. But in this case the difference in the rate of growth is not as large as in the Poisson process. Therefore we do not see the variability disappear as we increase the aggregation — at least not for the levels of aggregation shown in the figure.

The commonly used model for self-similarity indicates that when the process is aggregated by a factor of g the variability grows by a factor of g^H. The Poisson process is characterized by the exponent $H = 0.5$. More bursty processes are characterized by higher exponents, e.g. $H \approx 0.7$ or $H \approx 0.8$. Thus H is a parameter that characterizes the burstiness. This is the Hurst parameter, and it figures prominently in the rest of this chapter.

As a matter of terminology, a process with $H = 0.5$ (such as the Poisson process) satisfies the mathematical definition of self-similarity as given below. And indeed it looks self-similar when drawn at the appropriate scale, as in Figure 7.4. However, common usage departs from mathematical precision in this case, and only processes with $H > 0.5$ are typically called self-similar.

7.3 Mathematical Definitions

As noted earlier, the mathematical definitions of self-similarity, heavy tails, and long-range dependence are independent of each other. But in the practical situations of interest they turn out to coincide, and the self-similarity is typically a result of long-range dependence and not of heavy-tailed distributions [538]. This starts with Hurst's own work, which demonstrated that self-similarity is observed even if the individual samples are normally distributed [354]. Mandelbrot later showed how this self-similarity is due to long-range dependence (as surveyed in [467]). Therefore procedures to measure either self-similarity or long-range dependence can be used for the analysis of data, and procedures to generate either one of them can be used in modeling.

The material in this section involves a higher level of mathematics than other parts of the book. As we prefer to emphasize intuition and understanding rather than the exact mathematical definitions, each subsection starts with a non-technical description of the concepts being discussed.

7.3.1 Data Manipulations

The definitions of self-similarity and long-range dependence are based on some preprocessing of the data. We therefore start by describing how data may be manipulated.

Aggregation

In discrete processes the main tool used is aggregation. We start with a time series $X_1, X_2, \ldots, X_n$ that is given at some specific resolution. For example, if the resolution is milliseconds, X_1 could be the number of packets that arrived in the first millisecond, X_2 the number of packets arriving in the second millisecond, and so on.

We then use aggregation to view the same data at a different resolution. For example, we can sum up successive groups of 10 elements in the series, to derive a 10-fold aggregation. The first element in the new, aggregated series will be $X_1^{(10)} = X_1 + \cdots + X_{10}$, and it represents the number of arrivals in the first 10 milliseconds. The second element will be $X_2^{(10)} = X_{11} + \cdots + X_{20}$, and so on. In general, the elements of a series with m-fold aggregation are

$$X_i^{(m)} = \sum_{j=(i-1)m+1}^{i\,m} X_j \tag{7.1}$$

Increments and Accumulation

A basic distinction can be made between increments and accumulations. They are related to each other as a function and its integral, or as a derivative and its function: the accumulations are the sums of the increments. In mathematical notation, we start with a time series X_j and define

$$Y_i = \sum_{j=1}^{i} X_j$$

that is, Y_i is the sum of the first i elements of the series $(X_j)_{j=1..n}$. The Y_is are then called the *cumulative process* associated with the X_js, and the X_js are called the *increment process* associated with the Y_is. The name "increment process" derives from the obvious relationship

$$Y_{i-1} + X_i = Y_i$$

i.e., the Xs are the increments between successive Ys. Another name is the differences (or first differences) of the Ys; yet another name is innovations. This relationship is illustrated in the top two graphs of Figure 7.5.

Centering

The study of burstiness is concerned with deviations from the mean. We may know the mean arrival rate, but what we are interested in are the bursts that deviate from this mean. To focus on the deviations it is best to use *centered* data. To obtain centered data we calculate the average of the whole series, and subtract it from each sample (or in a theoretical framework, if we know the distribution, we would subtract the expected value). We now have a new sequence:

$$Z_i = X_i - \bar{X}$$

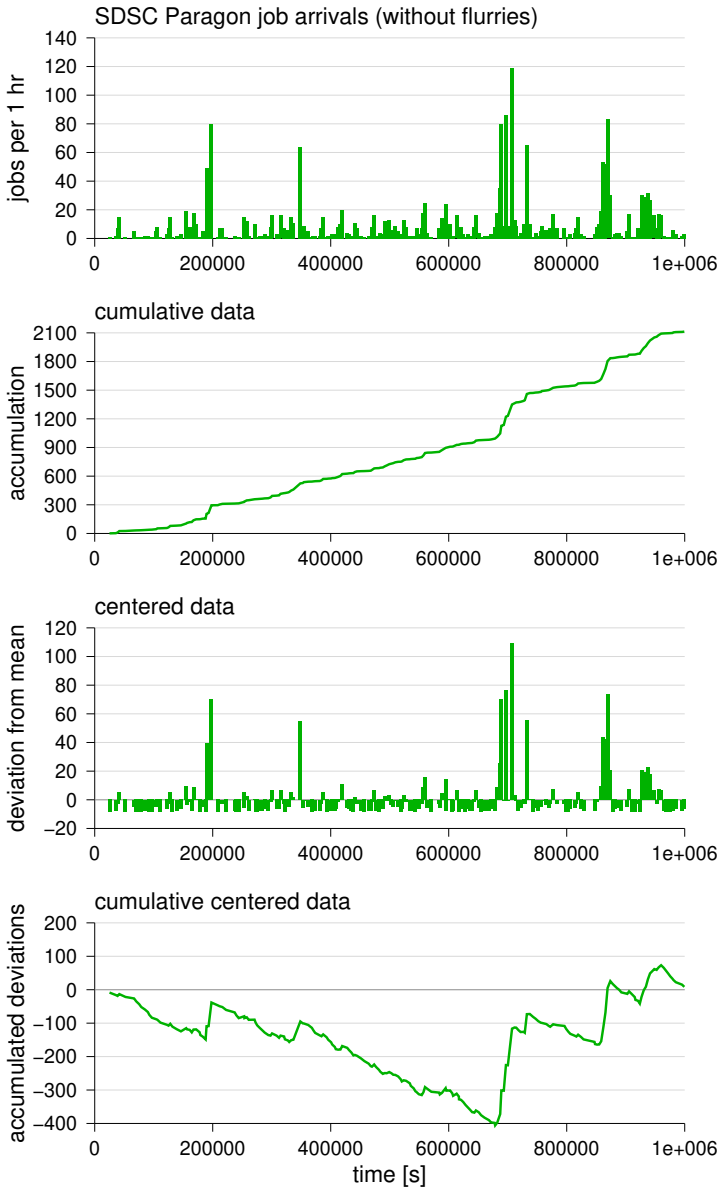


Figure 7.5: *Centering and accumulation of data.*

This transformation is illustrated in the third graph from the top of Figure 7.5. Note that while the original data is typically positive (or at least non-negative, $X_i \geq 0$), the centered data can be negative.

Using the centered data we can define a new cumulative process

$$Y_i = \sum_{j=1}^{i} Z_j = \sum_{j=1}^{i} (X_j - \bar{X}) \tag{7.2}$$

This has the unique property of starting and ending at 0, because the mean of the centered data is 0:

$$Y_n = \sum_{i=1}^{n} Z_i = \sum_{i=1}^{n} (X_i - \bar{X}) = \sum_{i=1}^{n} X_i - n\bar{X} = 0$$

This process is illustrated in the bottom graph of Figure 7.5.

The following discussions try to consistently use X or Z for increment processes, and Y for cumulative processes, but sometimes X, Y, and Z are just arbitrary random variables...

7.3.2 Exact Self-Similarity

Fractals such as those shown in Figure 7.2 are *exactly self-similar* because even when we change their scale, the original and scaled versions are identical (strictly speaking, this property only applies to infinite structures). To define the self-similarity of stochastic processes, we must first be able to change their scale. This can be interpreted as being able to view the phenomena of interest at different scales.

In a continuous stochastic process, the index is time. Viewing it at different scales means viewing it at different resolutions, or with different time units. Formally, this is achieved by scaling time: we replace $Y(t)$ with $Y(at)$. If $a > 1$ we move faster along the time axis, and the process seems to be condensed; conversely, if $a < 1$ we move more slowly and the process is diluted.

To qualify as self-similar, the scaled process should exhibit the same behavior as the original one. Therefore only the magnitude is allowed to change. Such reasoning leads to the following definition of exact self-similarity:

$$Y(t) \overset{\text{d}}{\approx} \frac{1}{a^H} Y(at) \tag{7.3}$$

Given that we are dealing with stochastic processes, we cannot expect both sides to be really identical; they can only be identical in a statistical sense. Therefore we use $\overset{\text{d}}{\approx}$, which means that the two sides have the same finite dimensional distributions.

Background Box: Finite Dimensional Distributions

Saying that two processes have identical finite dimensional distributions is a very strong statement.

Consider the processes X and Y and observe them at the same time instant t. This gives the random variables $X(t)$ from process X, and $Y(t)$ from Y. If they have the same distribution, then for every value z we have

$$\Pr(X(t) \leq z) = \Pr(Y(t) \leq z)$$

Note that the distribution for a different time t' may be different from that at time t; the important thing is that it is the same for $X(t')$ and $Y(t')$. The distributions of single

samples are known as *first-order* distributions. So if they are all the same we may say that X and Y have identical first-order distributions. (If, in addition, the processes are stationary, then the distribution will in fact be the same for all values of t.)

Now select a pair of samples from each process: $(X(t_1), X(t_2))$ from X and $(Y(t_1), Y(t_2))$ from Y. Pairs of samples have a joint distribution. If the samples from the two processes have the same joint distribution, than for every pair of values z_1 and z_2 we have

$$\Pr(X(t_1) \leq z_1, X(t_2) \leq z_2) = \Pr(Y(t_1) \leq z_1, Y(t_2) \leq z_2)$$

If this is true for every pair of samples, we say that X and Y have identical *second-order* distributions. This in turn implies that they have the same covariances. This property follows immediately from the definition of the covariance as $\gamma(X(t_1), X(t_2)) = \mathbb{E}[(X(t_1) - \mathbb{E}[X(t_1)])(X(t_2) - \mathbb{E}[X(t_2)])]$, which is calculated as the double integral of such products weighted by their joint (second-order) probability. (Covariance is discussed at length later.)

In the general case, we sample the two processes at n different times, obtaining $(X(t_1), X(t_2), \ldots, X(t_n))$ from X and $(Y(t_1), Y(t_2), \ldots, Y(t_n))$ from Y. Again, in each subset the samples have some joint distribution. The requirement is that all these joint distributions be the same, for all subsets of size n — and, moreover, for all sizes n. Hence the name "finite dimensional distributions".

Importantly, the behavior of an infinite series is fully defined by its finite dimensional distributions. Thus considering them entails no loss of generality.

End Box

The use of Y is no accident — this definition applies to cumulative processes. In effect, by scaling time we are looking at different levels of aggregation. $Y(t)$ is the sum of X up to time t. $Y(at)$ is the sum up to time at, which can be regarded as the aggregation of the sum up to t, the sum from t to $2t$, the sum from $2t$ to $3t$, and so on.

Cumulative processes are nonstationary by definition: the distribution of what we see depends on t, the amount of time that we have been accumulating data. Thus in such processes the magnitude changes monotonically with time, and scaling time can lead to a similar process that simply grows at a different rate. The definition of self-similarity cannot be applied to a stationary process, because stationarity implies that $X(t)$ and $X(at)$ should have precisely the same distributions. There can be no correction of magnitude to compensate for the change in scale.

Conversely, in a nonstationary cumulative process, we can have a change of scale. The factor a^H compensates for the change in rate resulting from the scaling. If $a > 1$ we move faster along the time axis and get to high values faster. We therefore divide by $a^H > 1$ to make up for this effect. The opposite happens when $a < 1$.

Note that we multiply time by a, but divide the effect on Y by a^H. The reason is that the effect is not linear. In fact, the way in which scaling time affects the process is the whole point, as noted earlier in Section 7.2.4. The model is that the effect scales as a raised to the power H. H is the *Hurst parameter*, which characterizes the degree of self-similarity.

Importantly, the same exponent H applies to all scaling factors and to distributions of all orders. Self-similar processes are fully characterized by this exponent, and are therefore denoted H-ss. In the more general case, different distributions may scale according

to different exponents, but then the process is not called self-similar. Such complex scaling is mentioned briefly at the end of this chapter.

Although the definition of self-similarity applies to cumulative processes, in practice, we are more often interested in increment processes than in cumulative processes. One reason is that the increment process may be stationary, and thus makes for a better model. As a result, we are often interested in self-similar processes that have stationary increments, and specifically in the increments themselves. Such processes are denoted H-sssi and form a subset of all self-similar processes.

7.3.3 Focus on the Covariance

Considering all the finite-dimensional distributions of the process gives a full characterization. If the distributions are identical in the original process and the scaled process, then the process indeed possesses the property of exact self-similarity. But one can also opt for a partial characterization. For example, one may require that only second-order distributions be the same. Doing so greatly simplifies the description, and may also lead to simpler models.

Although using only second-order distributions does not provide a full characterization, it does provide a good characterization of the basic behavior of the process. To explain what this means we use the following analogy. Imagine some quantity has a unimodal distribution without significant tails. This quantity is then well described by its mean and its standard deviation. This does not provide all the information that the distribution itself provides, but it succinctly describes the two most important features, namely the expected magnitude and dispersion. Likewise, when we have a series of values, the most important feature to characterize is the covariance.

Characterizing the covariance of a series does not mean summing it up to a single number. Rather, we need to use the *covariance matrix*. Assume we are interested in characterizing the series $X_1, X_2, \ldots, X_n$. The covariance of any two elements, X_i and X_j, is

$$\gamma(X_i, X_j) = \mathbb{E}[\,(X_i - \mathbb{E}[X_i])\,(X_j - \mathbb{E}[X_j])\,]$$

(see the box on page 292 for an explanation of this expression). The covariance matrix is then

$$\Sigma = \begin{pmatrix} \gamma(X_1,X_1) & \gamma(X_1,X_2) & \ldots & \gamma(X_1,X_n) \\ \gamma(X_2,X_1) & \gamma(X_2,X_2) & \ldots & \gamma(X_2,X_n) \\ \vdots & & & \vdots \\ \gamma(X_n,X_1) & \gamma(X_n,X_2) & \ldots & \gamma(X_n,X_n) \end{pmatrix}$$

The diagonal elements of this matrix are the variances of the different elements in the series. The off-diagonal elements give a characterization of their pairwise joint distribution. In particular, they quantify the degree to which pairs of elements tend to deviate from their mean together in the same direction.

In many cases we can further assume that X_i is stationary and centered, so $\mathbb{E}[X_1] = \mathbb{E}[X_2] = \ldots = \mathbb{E}[X_n] = 0$. In this case, the matrix is actually largely redundant. First, the variances (the diagonal elements) are obviously all equal. But stationarity implies

that other statistics are also not dependent on the index. In particular, the covariance of X_1 and X_2 is the same as the covariance of X_2 and X_3, the covariance of X_3 and X_4, and so on. Thus we find that for *every* diagonal, the elements on that diagonal are all equal to each other.

When this is the case, it is possible to characterize the full covariance matrix by its first row alone. This row gives the covariance of each element in the series with the first element. This is then simply the autocovariance function. Thus for stationary processes the autocovariance gives a good characterization of the entire joint distribution.

(There is also one case where the autocovariance, and second-order statistics in general, actually provide the full characterization, and not just a good characterization of the main features. This is when the process is Gaussian, and $X_1, \ldots, X_n$ have a multinormal (or multivariate normal) distribution. Because this distribution is completely specified by its vector of expectations and its covariance matrix, ignoring higher-order statistics does not entail any loss of information.)

In the following two sections we focus on characterizations based on the autocovariance rather than the full distributions. This material is highly technical. To continue with the intuition of what it all means, skip to Section 7.3.6.

7.3.4 Long-Range Dependence

The property that typically leads to self-similarity is long-range dependence. This is defined directly in terms of the increments X, as opposed to self-similarity, which was defined in terms of the cumulative process Y. We assume that these increments are stationary.

The "dependence" part means that successive elements are *not* independent of each other, as is often assumed. Rather, each event is dependent on previous ones. If we see some low-load elements, we therefore expect to see more low-load elements. If we are observing high load levels, we expect to see more of the same.

The "long-range" part means that this dependence is far-reaching, and elements that are very far apart may nevertheless be correlated with each other. However, correlation does actually taper off with distance. The point is that it decays very slowly. When the correlation decays exponentially with distance, there is short-range dependence. But if it decays according to a power law, we have long-range dependence. This is similar to the distinction between short tails and long tails considered in Chapter 5.

More formally, long-range dependence is defined as having an autocorrelation function that falls off polynomially, with an exponent smaller than 1 [69]. To define this, we start with the autocovariance:

$$\gamma(k) = \mathbb{E}[\,(X_{t+k} - \mathbb{E}[X])\,(X_t - \mathbb{E}[X])\,] \tag{7.4}$$

This is independent of t and depends only on the lag k, because X_t is stationary. The autocorrelation is simply the autocovariance normalized by dividing by the variance. This can be expressed as $r(k) = \gamma(k)/\gamma(0)$. The property of long-range dependence then states that this function (and also the covariance) decays according to a power law:

$$r(k) \approx k^{-\beta} \qquad 0 < \beta < 1 \tag{7.5}$$

Such an autocorrelation decays as the lag k grows: there is a stronger correlation across short distances than across long distances. But the correlation across long distances is still strong enough to have a significant influence. In particular, it causes the autocorrelation function to be nonsummable, meaning that the sum of the autocorrelation function diverges:

$$\sum_{k=1}^{\infty} r(k) = \infty \tag{7.6}$$

Alternatively, if the autocorrelation falls off more rapidly, and, as a result, it is summable (that is, the sum is finite), we call the process short-range dependent.

Note that the summability of the autocorrelation function is only an issue for infinite series. With a finite series, as all real data is, correlations beyond the length of the series are undefined (and by implication zero). However, the property that the autocorrelation decays polynomially over the range where it *is* defined is still applicable.

Details Box: Slowly Varying Functions

All the definitions just presented are simple versions that can be generalized. The generalizations allow multiplication by a "slowly varying function" $L(k)$. Thus the expression for a polynomially decaying autocorrelation function is actually $r(k) \approx L(k)\, k^{-\beta}$, and so on. Including this function makes the expressions more cluttered, but does not seem to have any practical significance. We therefore ignore these mathematical subtleties.

However, in case you are interested, slowly varying functions are functions that are asymptotically constant as their argument grows. The definition is

$$\lim_{k\to\infty} \frac{L(a\,k)}{L(k)} = 1 \qquad \forall a > 0$$

Examples include the logarithmic function $L(k) = \log k$, the ratio of two polynomials with the same degree, and, of course, the constant function $L(k) = C$.

A discussion of this topic is given by Feller [254, chap. XVII, sect. 5].

End Box

What is the relation between long-range dependence and self-similarity? Long-range dependence is defined in terms of the autocorrelation, which is the autocovariance normalized by the variance. We therefore need to look into the structure of the autocovariance function of the increments of a self-similar process. To do so, we start with the covariance of the accumulations.

Assume that $Y(t)$ is an exactly self-similar process as defined earlier, and that it has stationary increments. Further assume that the data is centered, so its expected value is 0. Its covariance at times t_1 and t_2, where $t_1 > t_2$, is therefore

$$\gamma(t_1, t_2) = \mathbb{E}[Y(t_1)Y(t_2)]$$

To handle such expressions consider the expression $(Y(t_1) - Y(t_2))^2$. Opening the parentheses gives

$$(Y(t_1) - Y(t_2))^2 = Y(t_1)^2 - 2Y(t_1)Y(t_2) + Y(t_2)^2$$

which, by changing sides, yields

$$Y(t_1)Y(t_2) = \tfrac{1}{2}\left(Y(t_1)^2 + Y(t_2)^2 - (Y(t_1) - Y(t_2))^2\right) \tag{7.7}$$

According to our assumption, Y has stationary increments. Therefore the distribution of $Y(t_1) - Y(t_2)$ (the increment from time t_2 to time t_1) is the same as the distribution of $Y(t_1 - t_2) - Y(0)$ (the increment from time 0 to time $t_1 - t_2$, which covers the same duration). But Y is a cumulative process, so $Y(0) = 0$. Therefore $Y(t_1) - Y(t_2) \overset{\text{d}}{\sim} Y(t_1 - t_2)$.

Returning to the covariance, we find the expectation of both sides of Equation (7.7). As expectation is a linear operator, it can be applied to each term of the right-hand side. Moreover, because the expectation depends only on the distributions, we can substitute $Y(t_1 - t_2)$ for $Y(t_1) - Y(t_2)$. Therefore

$$\mathbb{E}[Y(t_1)Y(t_2)] = \tfrac{1}{2}\left(\mathbb{E}[Y(t_1)^2] + \mathbb{E}[Y(t_2)^2] - \mathbb{E}[Y(t_1 - t_2)^2]\right) \tag{7.8}$$

Before we continue, let us derive one useful expression. Using Equation (7.3) to see what happens at time $t = 1$, we get $a^H Y(1) \overset{\text{d}}{\sim} Y(a)$. Note that we use $\overset{\text{d}}{\sim}$ rather than $\overset{\text{d}}{\approx}$; the reason is that here we are not looking at the entire process, but rather at the random variable $Y(t)$ for a specific value t. By renaming a to t, this expression turns into

$$Y(t) \overset{\text{d}}{\sim} t^H Y(1) \tag{7.9}$$

By considering integral values of t we can discretize time, and we now know how the distribution of $Y(t)$ evolves from one time-step to the next.

Applying this to the three terms in expression (7.8) then yields

$$\mathbb{E}[Y(t_1)Y(t_2)] = \tfrac{1}{2}\left(|t_1|^{2H} + |t_2|^{2H} - |t_1 - t_2|^{2H}\right)\mathbb{E}[Y(1)^2] \tag{7.10}$$

(where we use absolute values because the squares in Equation (7.8) are, of course, necessarily positive). Moreover, $\mathbb{E}[Y(1)^2]$ is simply the variance of $Y(1)$, because $Y(1)$ is equal to X_1, and X is centered.

Now let us revert to the increments, which are defined as $X_j = Y(j+1) - Y(j)$. Because they are stationary, their autocovariance depends only on the lag and may be defined as

$$\gamma(k) = \mathbb{E}[X_k X_0]$$

Plugging the definition into this expression, we get

$$\gamma(k) = \mathbb{E}[(Y(k+1) - Y(k))(Y(1) - Y(0))]$$

But $Y(0) = 0$ as shown earlier, so the second factor is just $Y(1)$. Opening the parentheses and applying the expectation to each term individually, we get

$$\gamma(k) = \mathbb{E}[Y(k+1)Y(1)] - \mathbb{E}[Y(k)Y(1)] \tag{7.11}$$

We can now apply the equation for the covariance of the cumulative process given in Equation (7.10) to each term. The result is

$$\gamma(k) = \frac{\sigma^2}{2}\left(|k+1|^{2H} - 2|k|^{2H} + |k-1|^{2H}\right) \tag{7.12}$$

where $\sigma^2 = \mathbb{V}\mathrm{ar}(X) = \mathbb{V}\mathrm{ar}(Y(1)) = \mathbb{E}[Y(1)^2]$.

Finally, we can take a factor k^{2H} out of the parentheses and express this as

$$\gamma(k) = \frac{\sigma^2}{2}k^{2H-2}k^2\left(\left(1+\frac{1}{k}\right)^{2H} - 2 + \left(1-\frac{1}{k}\right)^{2H}\right)$$

To gain some understanding of how this expression behaves, let us consider the Taylor expansion of $f(x) = (1+x)^{2H}$ about 0. The first three terms in the Taylor expansion about a are $(1+a)^{2H} + 2H(1+a)^{2H-1}(x-a) + \frac{1}{2}2H(2H-1)(1+a)^{2H-2}(x-a)^2$. At $a = 0$, and setting $x = \frac{1}{k}$, this becomes $1 + 2H\frac{1}{k} + H(2H-1)\frac{1}{k^2}$. Repeating this for $x = -\frac{1}{k}$, and plugging into the above expression for $\gamma(k)$, shows that the parentheses with the factor k^2 tend to $2H(2H-1)$ as $k \to \infty$. Dividing by σ^2 we obtain the end result that

$$r(k) \longrightarrow H(2H-1)k^{2H-2} \qquad\qquad k \to \infty$$

Defining $\beta = 2(1-H)$, we find that for $\frac{1}{2} < H < 1$ the exponent β satisfies $0 < \beta < 1$, and therefore we have an autocorrelation function that satisfies the definition of long-range dependence.

Thus we have shown that the increments of a self-similar process exhibit long-range dependence. The converse is not, in general, true. There are many different processes that may have such long-range dependence, and not all of them are self-similar. However, processes with long-range dependence are in fact asymptotically second-order self-similar. (And in the special case of Gaussian distributions, they are actually asymptotically self-similar, and not only second-order).

7.3.5 Asymptotic Second-Order Self-Similarity

Exact self-similarity is not very useful in practice because the requirement of identical finite dimensional distributions is too strong. In the context of modeling workloads we are therefore more interested in phenomena that are *second-order self-similar*: after scaling, only the second order distributions are required to remain the same. In particular, we define second-order self-similarity in terms of the autocovariance function, which is a second-order measure. Second-order self-similarity is defined as retaining the same autocovariance function after aggregation and rescaling. This definition is especially important, because it turns out that it results from long-range dependence.

Recall that the autocovariance at a lag k of a time series $X_1, X_2, X_3, \ldots$ is defined to be

$$\gamma(k) = \mathbb{E}[\,(X_{t+k} - \mathbb{E}[X])\,(X_t - \mathbb{E}[X])\,]$$

(for an explanation see the box on page 292). To be meaningful, we require that this be stationary, and, specifically, second-order stationary. This means that it depends only

on the lag k, and not on the particular indices t and $t+k$. In other words, $\mathbb{E}[(X_{i+k} - \mathbb{E}[X])(X_i - \mathbb{E}[X])] = \mathbb{E}[(X_{j+k} - \mathbb{E}[X])(X_j - \mathbb{E}[X])]$ for all i, j, and k.

Similarly, for each level of aggregation m, we can define the autocovariance $\gamma^{(m)}(k)$ of the aggregated process $X_i^{(m)}$. Using the aggregated process as defined in Equation (7.1) is problematic because the autocovariance will naturally grow as a function of aggregation, and therefore will not be equal to the autocovariance of the original process. We therefore define a rescaled version of the aggregated process in which each element is divided by the standard deviation of the aggregated process. Given that the original process is stationary, this is the same as dividing by the standard deviation of the first element, so the definition is

$$\mathcal{X}_i^{(m)} = \frac{X_i^{(m)}}{\sqrt{\mathbb{V}\mathrm{ar}(X_1^{(m)})}} \tag{7.13}$$

(where Equation (7.1) defined $X_i^{(m)} = \sum_{j=(i-1)m+1}^{i\,m} X_j$.)

With this version of aggregation, $\gamma^{(m)}(k)$ is simply the autocovariance of the rescaled aggregated process:

$$\gamma^{(m)}(k) = \mathbb{E}[\mathcal{X}_{t+k}^{(m)}\, \mathcal{X}_t^{(m)}]$$

where we do not need to subtract the mean because the process is centered, and it does not depend on t due to stationarity. Then X_i is exactly second-order self-similar if, for every m and k

$$\gamma^{(m)}(k) = \gamma(k)$$

and it is asymptotically second-order self-similar if, for every k,

$$\gamma^{(m)}(k) \to \gamma(k) \qquad\qquad m \to \infty$$

Note that $\gamma(k)$ itself is as in Equation (7.12), with the special case of $\sigma = 1$ due to the construction of the normalized aggregated process. This is not the covariance of the original process in this case, because this process is *not* self-similar — only asymptotically so.

We want to show that long-range dependent processes are asymptotically second-order self-similar. The statement that a process is long-range dependent simply means that its autocovariance function (or, equivalently, its autocorrelation) decays like a power law:

$$\gamma(k) = \mathbb{E}[X_{k+1}X_1] \propto k^{-\beta}$$

for large values of k. Now consider the cumulative process $Y_n = \sum_{i=1}^{n} X_i$. Because it is based on a centered process, its mean is 0. Its variance is therefore

$$\mathbb{V}\mathrm{ar}(Y_n) = \mathbb{E}\left[Y_n^2\right] = \mathbb{E}\left[\left(\sum\nolimits_{i=1}^{n} X_i\right)^2\right]$$

By opening the parentheses we find that this is the expectation of the sum of terms of the form X_iX_j. By changing the expectation of the sum into a sum of expectations, we find

that this is actually the sum of the covariances. But X is stationary, so the covariance depends only on the differences in indices, not on the indices themselves. Therefore the elements on each diagonal of the covariance matrix are equal to each other, and the sum of the whole matrix becomes

$$\mathbb{V}\mathrm{ar}(Y_n) = n\gamma(0) + 2\sum_{k=1}^{n-1}(n-k)\gamma(k)$$

(the first term is the main diagonal, and the sum is on the other diagonals, where diagonal k has $n-k$ elements that are all $\gamma(k)$, and there are actually two such diagonals — above and below the main diagonal).

By multiplying and dividing each term by $n^{2+\beta}$ the sum can be expressed as follows:

$$\sum_{k=1}^{n-1}(n-k)\gamma(k) = n^{2-\beta}\frac{1}{n}\sum_{k=1}^{n-1}\left(1-\frac{k}{n}\right)n^{\beta}\gamma(k)$$

Recall that $\gamma(k) \propto k^{-\beta}$. Hence $n^{\beta}\gamma(k) \propto (\frac{n}{k})^{\beta}$. For large n the sum multiplied by $\frac{1}{n}$ then approximates the integral $\int_0^1(1-x)x^{-\beta}\mathrm{d}x$, which is finite when $\beta < 1$. We are therefore left with the relationship

$$\mathbb{V}\mathrm{ar}(Y_n) = \mathbb{E}[Y_n^2] \propto n^{2-\beta} = n^{2H}$$

for $H = 1 - \frac{\beta}{2}$.

Now let us look at the accumulations of the aggregated process. Note that the first element in the aggregated process is the sum of the first m elements in the original process. So is the mth element of the original cumulative process. Therefore $X_1^{(m)} = Y_m$. Using this in the definition of the cumulation of the rescaled process from Equation (7.13) we get

$$Y_n^{(m)} = \sum_{i=1}^{n}\mathcal{X}_i^{(m)} = \sum_{i=1}^{n}\frac{\sum_{j=(i-1)m+1}^{i\,m}X_j}{\sqrt{\mathbb{V}\mathrm{ar}(Y_m)}}$$

But in the rightmost sum we actually have a sum over all elements of X from the first element up to the nmth element. Therefore the expression can be simplified to

$$Y_n^{(m)} = \frac{Y_{nm}}{\sqrt{\mathbb{V}\mathrm{ar}(Y_m)}}$$

Using this we now characterize the variance of the cumulative aggregated process:

$$\mathbb{V}\mathrm{ar}(Y_n^{(m)}) = \frac{\mathbb{V}\mathrm{ar}(Y_{nm})}{\mathbb{V}\mathrm{ar}(Y_m)} \approx \frac{(nm)^{2-\beta}}{m^{2-\beta}} = n^{2-\beta} = n^{2H}$$

As before, this applies for every n, as $m \to \infty$. We used the fact that the denominator $\sqrt{\mathbb{V}\mathrm{ar}(Y_m)}$ is deterministic, so it can be taken out of the expression for the variance and squared.

Now we are finally ready to calculate the autocovariance of the rescaled aggregated process. According to Equation (7.11) this may be expressed as

$$\gamma^{(m)}(k) = \mathbb{E}[Y_{k+1}^{(m)}\, Y_1^{(m)}] - \mathbb{E}[Y_k^{(m)}\, Y_1^{(m)}]$$

Using Equation (7.8) and relying on the stationarity, we get

$$\begin{aligned}\gamma^{(m)}(k) &= \tfrac{1}{2}[\mathbb{V}\mathrm{ar}(Y_{k+1}^{(m)}) + \mathbb{V}\mathrm{ar}(Y_1^{(m)}) - \mathbb{V}\mathrm{ar}(Y_k^{(m)})] - \\ &\quad \tfrac{1}{2}[\mathbb{V}\mathrm{ar}(Y_k^{(m)}) + \mathbb{V}\mathrm{ar}(Y_1^{(m)}) - \mathbb{V}\mathrm{ar}(Y_{k-1}^{(m)})] \\ &= \tfrac{1}{2}[\mathbb{V}\mathrm{ar}(Y_{k+1}^{(m)}) - 2\mathbb{V}\mathrm{ar}(Y_k^{(m)}) + \mathbb{V}\mathrm{ar}(Y_{k-1}^{(m)})]\end{aligned}$$

And, using the above derivation for $\mathbb{V}\mathrm{ar}(Y_n^{(m)})$, we arrive at the conclusion that

$$\gamma^{(m)}(k) \to \tfrac{1}{2}\left[(k+1)^{2H} - 2k^{2H} + (k-1)^{2H}\right] \qquad m \to \infty \tag{7.14}$$

which is what we want based on the expression for $\gamma(k)$ given in Equation (7.12).

7.3.6 The Hurst Parameter and Random Walks

All the definitions in the preceding sections use the same parameter H, called the Hurst parameter, to express the degree of self-similarity (equivalently, $\beta = 2(1-H)$ can be used). But what does it mean?

A useful model for understanding the correlations leading to self similarity, and the meaning of H, is provided by random walks. A one-dimensional random walk is exemplified by a drunk on a sidewalk. Starting from a certain lamppost, the drunk takes successive steps either to the left or to the right with equal probabilities. Where is he after n steps? Note that this is just a picturesque version of the processes studied earlier, with the individual steps representing the increment process, and the location after n steps as the cumulative process.

Let us assume the steps are of unit size and independent of each other, and denote the drunk's location after i steps by Y_i. The relationship between Y_i and Y_{i+1} is

$$Y_{i+1} = Y_i + X_{i+1} = \begin{cases} Y_i + 1 & \text{with probability 0.5} \\ Y_i - 1 & \text{with probability 0.5} \end{cases}$$

where $X_{i+1} = \pm 1$ is the $i+1$st step. The expected location after this step is

$$\mathbb{E}[Y_{i+1}] = \mathbb{E}[Y_i + X_{i+1}] = \mathbb{E}[Y_i] + \mathbb{E}[X_{i+1}]$$

But $\mathbb{E}[X_{i+1}] = 0$ because the two options (a step to the left or a step to the right) cancel out. Using induction we then find that in general $\mathbb{E}[Y_i] = 0$.

But the chance that the drunk will return exactly to the origin is actually quite low. What really happens is that his location is binomially distributed around the origin, with equal probabilities of being a certain distance to the left or to the right. To find how far from the lamppost he will get, we therefore need to prevent such symmetries from

canceling out. A simple option is to look at Y_i^2 instead of at Y_i. For this, the expectation is

$$\begin{aligned}\mathbb{E}[Y_{i+1}^2] &= \mathbb{E}[(Y_i + X_{i+1})^2] \\ &= \mathbb{E}[Y_i^2 + 2Y_iX_{i+1} + X_{i+1}^2] \\ &= \mathbb{E}[Y_i^2] + \mathbb{E}[2Y_iX_{i+1}] + \mathbb{E}[X_{i+1}^2]\end{aligned}$$

Given that steps are independent, the middle term is 0 (because $\mathbb{E}[X_{i+1}] = 0$). The last term is simply 1. This then leads to the relation $\mathbb{E}[Y_{i+1}^2] = \mathbb{E}[Y_i^2] + 1$, and, by induction, to $\mathbb{E}[Y_{i+1}^2] = i + 1$. But we are interested in the average distance from the lamppost, not in its square. We cannot derive $\mathbb{E}[\,|Y_n|\,]$ directly from these results, because the relationship of $|Y_n|$ and Y_n^2 is nonlinear. But we can say that the root-mean-square of the distance is $\sqrt{n}$, that is

$$\sqrt{\mathbb{E}[Y_n^2]} = n^{0.5}$$

The real average distance turns out to be slightly smaller: it is $\mathbb{E}[\,|Y_n|\,] \to (\frac{2}{\pi}\, n)^{0.5}$. Such random walks are shown in the top-left graph of Figure 7.6.

Now consider a drunk with inertia. Such a drunk tends to lurch several steps in the same direction before switching to the other direction. The steps are no longer independent — rather, each step is typically correlated with the following steps, which tend to be in the same direction. Overall, the probabilities of taking steps in the two directions are still the same, but these steps come in longer sequences.

To make this more precise, we can try to characterize the lengths of sequences of steps in the same direction. If the steps are independent, these sequences are geometrically distributed, because the probability of taking k steps in the same direction is $(\frac{1}{2})^k$. Hence the probability of seeing long sequences decays exponentially.

But if the steps are correlated, long sequences of steps in the same direction will be more common. Let us take the simplest possible model of such correlations. In this model the drunk makes sequences of steps in the same direction, where the lengths of the sequences are Pareto distributed. As the Pareto distribution has a heavy tail, he will occasionally take very many steps in the same direction. Thus the drunk makes much more progress in either direction, and gets farther away from the lamppost. This is enough to change the exponent in the expected distance from the origin, which is found to behave like n^H, with $0.5 < H < 1$. Specifically, we get $H = \frac{1}{2}(3 - a)$, where a is the tail index of the Pareto distribution used to generate the sequences of steps. This is illustrated in Figure 7.6.

This, then, is the meaning of the Hurst parameter H. It is the exponent that describes the cumulative expected deviation from the mean after n steps. Higher values of H are the result of stronger long-range dependence of the process. And this observation — that long-range dependence is reflected in the range covered after n steps — is the basis of the rescaled range method developed by Hurst to measure long-range dependence (as described below in Section 7.4.2).

In common usage, the parameter H is also said to *quantify* the self-similarity of the process. But H can in principle have any value from 0 to 1. The interpretations of the different values are as follows.

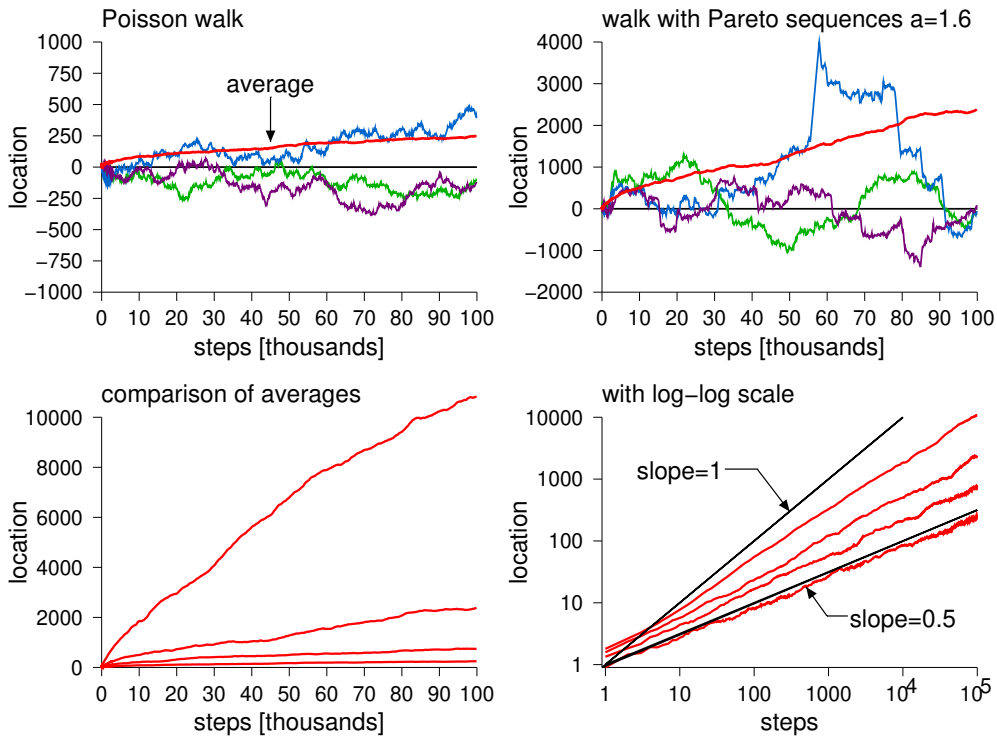


Figure 7.6: *A random walk with independent steps and a walk where steps are correlated because they come in sequences whose lengths are sampled from a Pareto distribution. In each case three example walks are shown, together with the distance from the origin averaged over 100 such walks. The bottom graphs show the averages for the Poisson walk and correlated walks where* $a = 2$, $a = 1.6$, *and* $a = 1.2$ *(and* $H = 0.5$, $H = 0.7$, $H = 0.9$*) from bottom to top.*

$H = 1$: In this case the process has a consistent trend and does not fluctuate around the origin at all. The expected distance covered is linearly related to the number of steps.

$\frac{1}{2} < H < 1$: This is a process in which consecutive steps are positively correlated. It is this type of process that is usually meant when someone uses the terms "self-similar" or "long-range dependent". The occurrence of longer sequences of similar deviations than would be expected had the steps been independent is called the *Joseph effect* (after the story of the seven good years and seven bad years from the book of Genesis) [466, p. 248]. It is also called a *persistent process*.

$H = \frac{1}{2}$: This is a boundary condition in which steps are uncorrelated, or at most have short-range correlation. An example is the Poisson process.

$0 < H < \frac{1}{2}$: This is a process in which consecutive steps tend to be inversely correlated, leading to its being called an *anti-persistent process*. The effect is that the

process covers less distance than would be expected if the steps were independent. The smaller H is, the more it tends to remain close to the origin. However, this situation seems to be much less common in practice.

$H = 0$: In this case each step deterministically cancels the previous one. The process stays exactly at the origin.

Of course, the transition from one range to the next is not sharp. When $H = \frac{1}{2}$, the distribution of sequence lengths is exponential (where we refer to sequences of steps in the same direction). As H goes down from $\frac{1}{2}$ to 0, they become successively shorter. Conversely, as H goes up from $\frac{1}{2}$ to 1, they become successively longer, and it takes more and more time for the process to return to the origin. At the limit of $H = 1$ it does not return any more.

7.4 Measuring Self-Similarity

Self-similar models are an alternative to random Poisson processes. This section therefore start with testing whether an arrival process is a Poisson process. It then discusses tests that directly test for self-similarity. These tests typically work by trying to estimate the Hurst parameter: if it is in the range $0.5 < H < 1$, the process is self-similar.

The tests operate in either of two ways. Some, such as the rescaled range method and the variance-time method, are time-based. These methods consider the data as a time series, and analyze its fluctuations. Others are frequency-based, and relate to spectral methods of analyzing time series, in which the data is first transformed to the frequency domain using a Fourier transform.

7.4.1 Testing for a Poisson Process

The Poisson process is actually self-similar with a Hurst parameter of $H = \frac{1}{2}$. However, according to common usage if a process is Poisson, it is not called self-similar.

The recognition that a process is Poisson can be based on any of its known characteristics. For example, the following tests have been proposed: [427, 540, 369].

- In a Poisson process the interarrival times are exponentially distributed. Conversely, if the interarrival times are exponential and independent, it is a Poisson process.

  A simple test is that the coefficient of variation of interarrival times is 1 (or close to 1), as it should be for an exponential (memoryless) distribution. If it is much larger, this indicates that arrivals are bursty and not exponentially distributed. As a simpler visual check, the log histogram of the interarrival times should be linear: if $f(t) = \frac{1}{\theta} e^{-t/\theta}$, then $\log f(t) = -\log \theta - \frac{1}{\theta} t$.

  The exponential distribution that provides a maximum likelihood fit to given interarrivals data is an exponential with a parameter θ that corresponds to the mean interarrival time. It can be compared with the empirical distribution using the

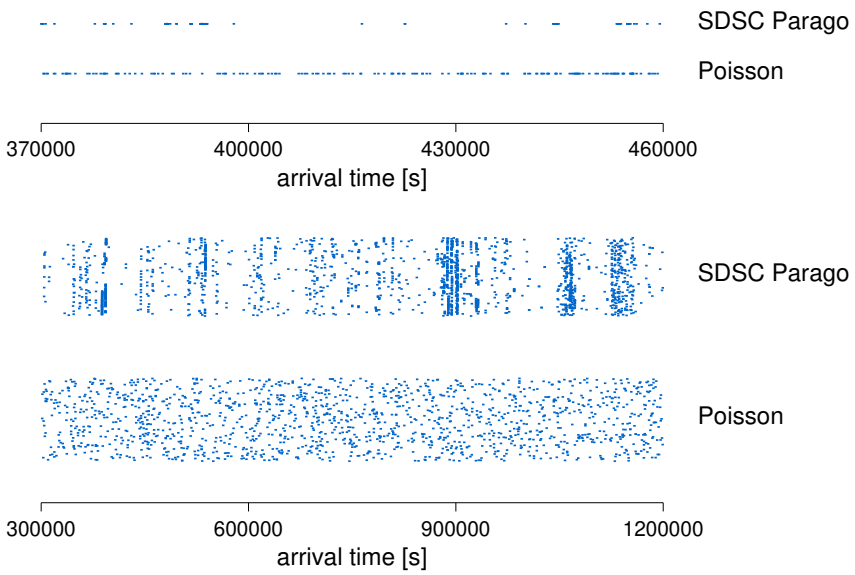


Figure 7.7: *Top: plotting arrivals from the SDSC Paragon log and a Poisson process with the same average arrival rate. Bottom: texture plots for the same two arrival processes.*

goodness-of-fit techniques described in Section 4.5 (e.g., the Kolmogorov-Smirnov test).

A potential problem with these estimates occurs in networking. In this context packets are sometimes transmitted back to back. In particular, the more congested the link, the less idle time there is between packets. When many packets are transmitted back to back, the interarrival times are determined by the packet sizes [395].

- Alternatively, arrivals of a Poisson process are distributed uniformly over the duration of the process. This is tested by verifying that the arrival times conform to a uniform distribution.

  A graphical method to test this is to simply plot them. The simplest approach is to plot arrivals as dots along a timeline [540]. An example is shown in the top graph in Figure 7.7, which shows that arrivals from a Poisson process have smaller gaps than real arrivals from a self-similar process. However, gaps do in fact occur, because the exponential distribution also has a tail. The way they look may just result from the scale.

  A better approach is therefore to create a texture plot. Partition the timeline into segments of equal lengths; this length is the time unit used. Then plot these segments *vertically* next to each other (bottom of Figure 7.7). In other words, if we have an arrival at time t and the time unit is u, the arrival would be plotted at coordinates $(\lfloor t/u \rfloor, t \bmod u)$. It is important to select a suitable time unit, one that

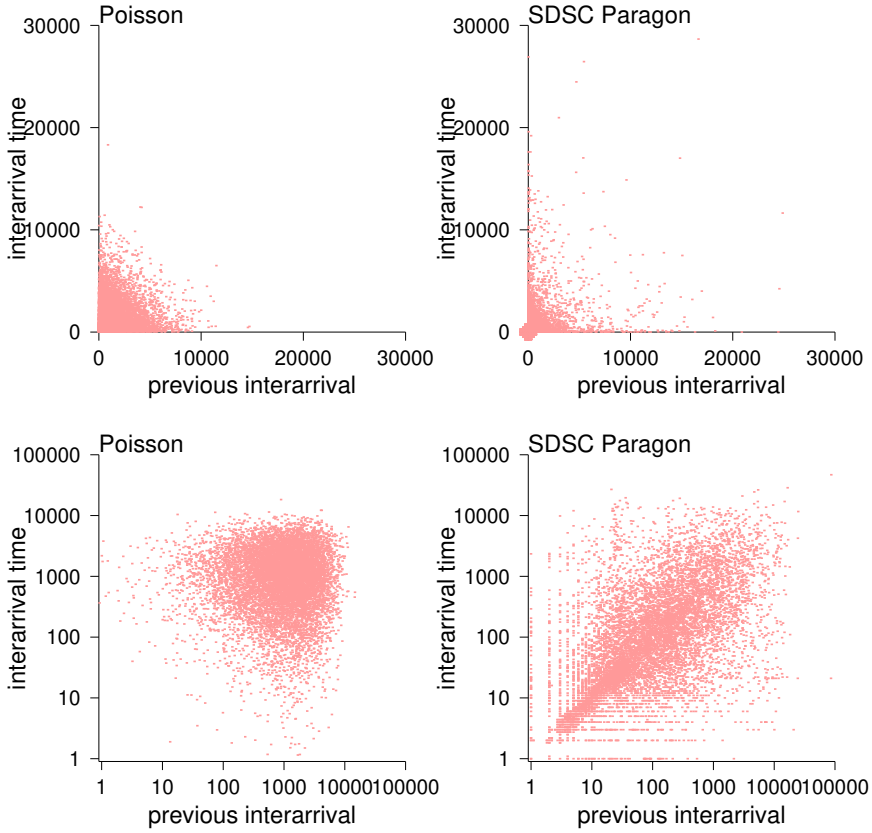


Figure 7.8: *A scatterplot showing each interarrival as a function of the previous one can be used to find correlations. A linear trend such as that in the SDSC Paragon data with log scaling may be a result of the daily cycle (in which case the interarrivals are indeed not independent).*

is not too small nor too big. For example, a time unit that is several times larger than the average interarrival time may be good. The result for uniform arrivals will then be a strip with constant shading. But if the arrivals are correlated with large gaps we will see a varying texture.

- Apart from the distribution, the interarrivals must also be independent. This can be tested by using the autocorrelation function. One simple test is to verify that the autocorrelation is low; in particular, the serial correlation (autocorrelation at lag 1) should already be close to zero [320]. Another test is to verify that the autocorrelation is positive approximately the same number of times that it is negative.

  A graphical method to test independence is to plot each interarrival time as a function of the previous one (Figure 7.8). If a pattern emerges, they are not independent.

To test for a nonhomogeneous Poisson process, it is possible to divide the arrivals into non-overlapping intervals that are thought to be homogeneous, and test each one separately.

7.4.2 The Rescaled Range Method

The following tests can be used to assess whether a process is self-similar, and at the same time to measure the Hurst parameter H. A value of $H = \frac{1}{2}$ indicates that the process is not self-similar, but conforms with a Poisson process. A value of H in the range $\frac{1}{2} < H < 1$ is taken to mean that the process is indeed self-similar.

One way of checking whether a process is self-similar is directly based on the definition of the Hurst parameter and its relation to random walks as described in Section 7.3.6: measure the range covered after n steps, and check the exponent that relates it to n. In fact, this is the method originally used by Hurst [354].

Assume you are given a time series $X_1, X_2, \ldots X_n$. The procedure is as follows [467, 545]:

1. Center it by subtracting the mean $\bar{X}$ from each sample, giving $Z_i = X_i - \bar{X}$. The mean of the new series is obviously 0.
2. Calculate the distance covered after j steps for all j:
$$Y_j = \sum_{i=1}^{j} Z_i$$
3. The range covered after n steps is the difference between the maximum distance that has occurred in the positive direction and the maximum distance that has occurred in the negative direction:
$$R_n = \max_{j=1\ldots n} Y_j - \min_{j=1\ldots n} Y_j \tag{7.15}$$
Note that by the definitions of Z_i and Y_j, we get that $Y_n = 0$. Therefore $\max_j Y_j \geq 0$ and $\min_j Y_j \leq 0$.
4. The magnitude of R_n is related to two factors: how many consecutive steps are typically taken in the same direction, and the size of each such step. The sizes of the steps are related to the variability of the original time series. This variability can be canceled out by measuring R_n in units of the standard deviation. Therefore rescale it by dividing by S, the standard deviation of the original n data points $X_1 \ldots X_n$. The result then characterizes the correlation in the time series.
5. The model is that the rescaled range, $(R/S)_n$, should grow like cn^H (recall the random walks of Section 7.3.6). To check this take the log, leading to
$$\log \left(\frac{R}{S} \right)_n = \log c + H \log n \tag{7.16}$$
If the process is indeed self-similar, plotting this for multiple values of n will lead to a straight line, and the slope of the line gives H.

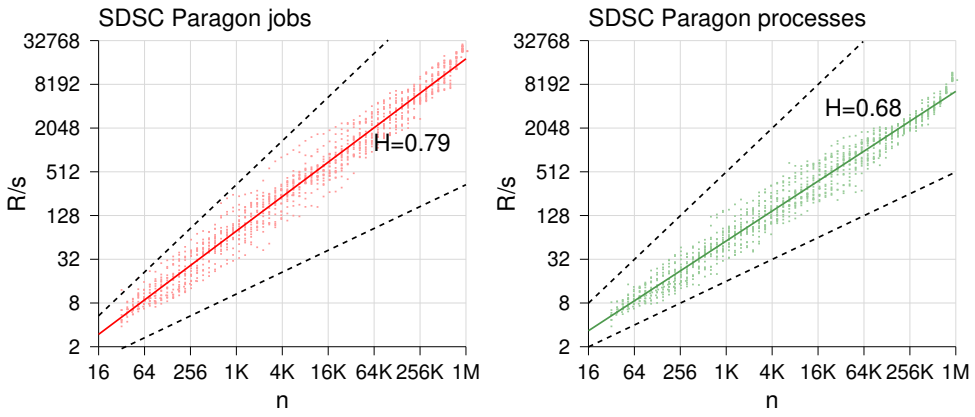


Figure 7.9: *Finding H using a pox plot of $(R/S)_n$ values calculated for multiple (possibly overlapping) subsets of the data. The data is the same as that used in Figure 7.3, at 1-minute resolution. Dashed lines showing the slopes corresponding to $H = \frac{1}{2}$ and $H = 1$ are given for reference.*

To apply this procedure, we need to generate data for different values of n. Assume that our original dataset has N elements (e.g., the number of packets that arrived during N consecutive milliseconds). We then need to answer two questions:

1. Which values of n should we use?
2. Which data elements should we use for each value of n?

Given that the analysis is done in a logarithmic scale, the answer to the first question is to use logarithmically spaced values. Obviously, it is possible to use powers of two. But these are rather distant from each other, leading to situations where the largest possible spans are not represented if N is not a power of two. It is therefore common to use powers of a smaller number, such as 1.2. The number of distinct sizes is then $m = \lfloor log_{1.2} N \rfloor$. For each $i \leq m$ the size of each subset is $n_i = \lfloor 1.2^i \rfloor$. Note that, for small values of n_i, we may see many subsequences that are all zero, and therefore uninteresting. It is thus common to skip the smallest values, and to start with the minimal number of samples needed to "see some action".

As for which data elements to use, it is better to use multiple subsets of size n, rather than using only the first n elements. This ensures that all the available data is used for each size, and not only a subset of the data that may not be representative of the whole. Furthermore, for large n it is common to use multiple overlapping subsets.

Given a large number of subsets of different sizes, we calculate the $(R/S)_n$ metric for each one. We then draw a scatterplot of all these data points (i.e., $(R/S)_n$ vs. n) on log-log axes. The result is called a *pox plot*, and looks like the plots shown in Figure 7.9 (which include up to 20 subsets of each size). This plot is used for two purposes. First, it enables a qualitative assessment of whether the points fit a straight line. If they do, then

the model stating that the rescaled range grows as a power of n is satisfied. Second, we then use linear regression to fit a line through all the data points. The slope of this line is our estimate of the Hurst parameter H.

Practice Box: Considerations for Pox Plots

There are several ways to decide exactly which values of n and which subsets of each size to use.

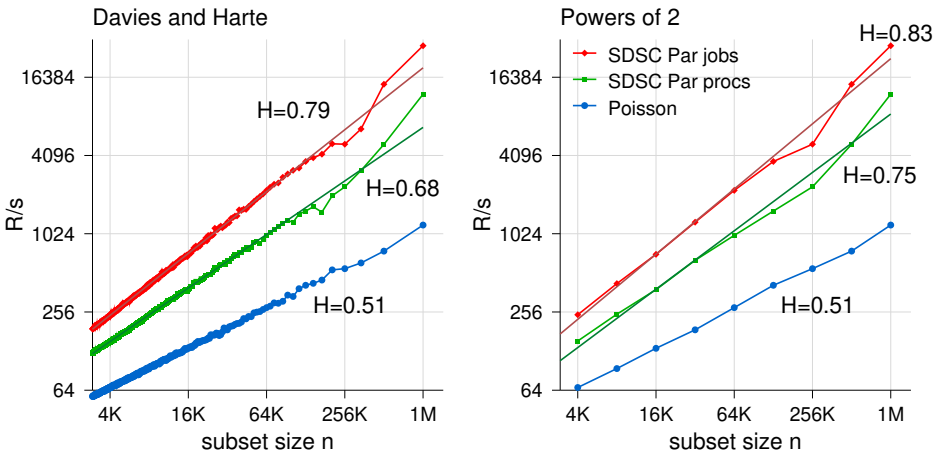
Davies and Harte suggest using non-overlapping subsets, and averaging the subsets of each size before performing the linear regression [165]. Starting from the top, they find sizes that divide N: $n_1 = N$, $n_2 = \lfloor N/2 \rfloor$, $n_3 = \lfloor N/3 \rfloor$, $n_4 = \lfloor N/4 \rfloor$, $n_5 = \lfloor N/5 \rfloor$, and $n_6 = \lfloor N/6 \rfloor$. Thereafter they use logarithmically spaced sizes, with $n_i = n_{i-1}/1.15$. For each size n_i they then partition the data into $k_i = \lfloor N/n_i \rfloor$ non-overlapping subsets (if $n_i k_i < N$, an equal number of extra elements is left at each end). These subsets are used to calculate k_i values of $(R/S)_{n_i}$, and these values are averaged:

$$\bar{R}_{n_i} = \frac{1}{k_i} \sum_{j=1}^{k_i} \left(\frac{R}{S} \right)_{n_i,j}$$

These can be plotted on log-log axes (that is, plot $\log \bar{R}_{n_i}$ as a function of $\log n$). If they seem to conform to a straight line, a linear regression is performed to find the slope, which is an estimate of H.

A more extreme version of this approach is to use only powers of two. Assume that N is a power of two, that is $N = 2^m$ for some m. We will use subsets that are smaller powers of two. For any given $i \leq m$, the size of each subset is $n_i = 2^i$, and we have enough data for $k_i = 2^{m-i}$ such subsets. We therefore partition the data into k non-overlapping parts, and continue as above. Although in practice N is not always a power of two, we may be able to make it close by judicious selection of the basic time unit. The example shown below is based on the SDSC Paragon log, which is 63,370,152 seconds long. The values of i used range from 12 to 20, corresponding to n values from 4096 to 1,048,576. By choosing the basic time unit to be 60 seconds, the highest value covers 62,914,560 seconds and nearly subsumes the entire log.

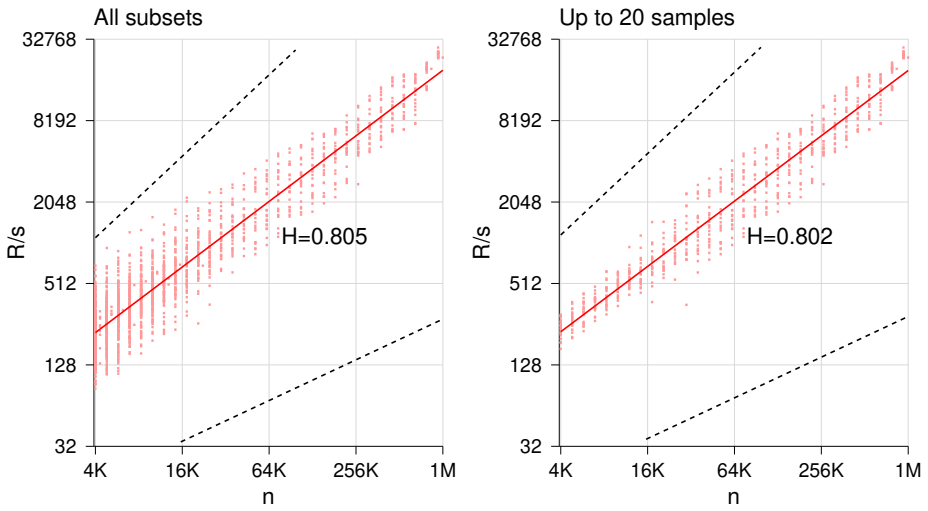
Because we calculate the average for each size before performing the linear regression, the result is not a scatterplot but rather a line plot. The following examples uses the same data as in Figures 7.3 and 7.9. A Poisson process with no self-similarity is included as reference. The Davies and Harte scheme includes many more points and seems to be more accurate; at least, it conforms with the results of the pox plots in Figure 7.9.



The alternative approach is to use overlapping subsets — as was done in Hurst's original work [354]. This method makes more efficient use of the available data if the total size is not a power of two and is also not so big. The idea is to use many more subsets for each value of n, especially for the large ones, where by definition we do not have enough data for multiple non-overlapping subsets.

The values of n are typically logarithmically spaced; there is no need for special treatment of the larger values, because we don't insist on partitioning the data into non-overlapping subsets. Therefore we can simply use powers of a number slightly larger than 1, say 1.2. The number of distinct sizes is $m = \lfloor log_{1.2} N \rfloor$, and the sizes themselves are $n_i = \lceil 1.2^i \rceil$ for $i \leq m$. But instead of using only $\lfloor N/n_i \rfloor$ non-overlapping subsets, we use many more overlapping ones. The extreme case is to use all subsets of size n_i that start from the first data element through the $(N - n_i)$th data element. This leads to a very dense scatterplot. But a worse consequence is that it places significant weight on small values of n_i, which are represented by multiple points, at the expense of large values of n_i, which are represented by fewer points. As we are actually interested in the asymptotic behavior as n grows, this seems ill-advised.

An alternative is to use a constant number k of representatives for each size n_i. For large n_i, each representative is a subsequence, and they overlap each other. Specifically, if $k > N/n_i$, we cannot partition the dataset into k non-overlapping parts, so we use overlapping ones; subset number j will then start with element number $(j-1)\frac{N-n_i}{k}+1$. If $k < N/n_i$, in contrast, we have the opposite problem: using only k subsets of size n_i leaves many data elements unused. To use all the data we therefore calculate the R/S metric for *all* non-overlapping subsets of size n_i. We then group these results into k groups and find the average of each group. This limits us to the desired k data points, but at the same time uses all the original data. In the following figure (using the SDSC Paragon jobs data), $k = 20$ is used.



Finally, an important question is how to handle situations in which short subsequences (small n) are identically zero. This happens, for example, in the parallel supercomputer arrival data. There are periods of more than an hour in which no new work had arrived, leading to several thousand zero entries in a time series recorded with 60-second resolution. One option is to only include those subserieses in which some activity had occurred, and to disregard the rest. A better alternative is to only use sizes for which all subserieses are nonzero. This is possible if the total dataset size N is large enough relative to the longest quiet period.

End Box

Note that the regression line calculated from the pox plot provides not only a slope, but also an intercept. This gives some indication of the short-range dependence among the samples. If a strong short-range dependence is present, the deviation from the mean will be somewhat greater, but will not continue to grow with n.

Indeed, a major drawback of R/S analysis is its susceptibility to short-range dependence. In fact, short-range dependence alone can also lead to high $(R/S)_n$ values. To counter such effects, it has been suggested to normalize the range differently, using [447]

$$\sqrt{S^2 + \frac{2}{n}\sum_{k=1}^{q}\left(1 - \frac{k}{q+1}\right)\left(\sum_{i=1}^{n-k}(X_i - \bar{X})(X_{i+k} - \bar{X})\right)}$$

rather than S in the denominator. The added term is a weighted sum of the autocovariance up to a lag of q, where the weight drops linearly with the lag. Thus an increased range that is a result of correlation up to this lag will cancel out, and only truly long-range dependence will lead to high $(R/S)_n$ values. However, the bound q should not be too high, so as to avoid false negatives where true self-similarity is mistaken for short-range dependence and therefore not identified [682].

Alternatively, one can try to assess the effect of short-range dependence by chopping up the data into short sequences, reshuffling them, and applying the test again. This is described in Section 7.4.8.

To read more: The R/S method has been the focus of much study, which showed it to be robust for processes originating from many different underlying distributions. A detailed review is provided by Mandelbrot and Taqqu [467]. The paper by Lo provides a good review of its statistics and shortcomings [447].

7.4.3 The Variance Time Method

Another way of checking for self-similarity is based on the rate in which the variance decays as observations are aggregated [754].

Let us start with a self-similar cumulative process, which has stationary increments. We are interested in the variance of these increments, and even more so, in the variance of the aggregated increment process. Since the X_is' distribution is stationary, we can represent their distribution by the distribution of the first random variable X_1, and likewise for their aggregations.

Aggregation as defined in Equation (7.1) is closely related to computing a sample average: we sum m samples and divide by m. But this can be expressed in terms of the cumulative process:

$$\bar{X}_1^{(m)} = \frac{1}{m}\sum_{j=1}^{m} X_j = \frac{1}{m}\, Y(m)$$

So using an earlier derivation, we can now say something about the distribution of the aggregated (or rather, averaged) increment process:

$$\bar{X}_1^{(m)} \overset{\mathrm{d}}{\sim} \frac{1}{m}\, m^H Y(1)$$

Using the fact that by definition $Y(1) = X_1$, we then get

$$\bar{X}_1^{(m)} \overset{\mathrm{d}}{\sim} m^{H-1} X_1$$

Recall that this is not equality of expressions, but rather equality of distributions. As a result, we can use the known equality $\mathbb{V}\mathrm{ar}(aX) = a^2\mathbb{V}\mathrm{ar}(X)$ to derive a relationship between the variance of the increment process and the variance of its aggregation. Substituting the factor m^{H-1} for a, the result is

$$\mathbb{V}\mathrm{ar}(\bar{X}_1^{(m)}) = m^{2(H-1)}\mathbb{V}\mathrm{ar}(X_1)$$

In effect, the variance is reduced as the level of aggregation grows. It is therefore more common to rewrite the last equation as

$$\mathbb{V}\mathrm{ar}(\bar{X}_1^{(m)}) = \frac{1}{m^\beta}\mathbb{V}\mathrm{ar}(X_1) \qquad X_1, X_2, \ldots \text{ long-range dependent} \tag{7.17}$$

where $\beta = 2(1-H)$.

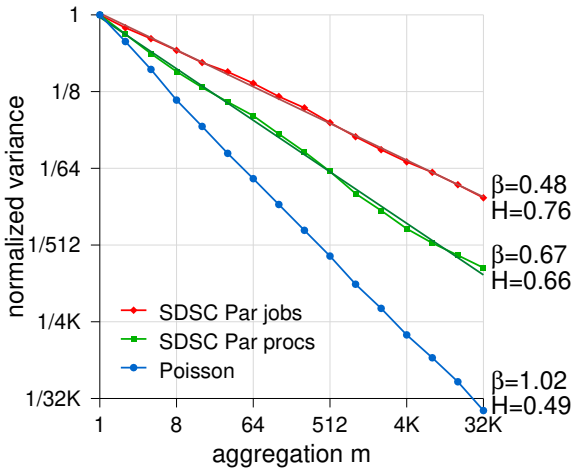


Figure 7.10: *The variance-time method for measuring self-similarity, applied to the data in Figure 7.3. A Poisson process with no self-similarity is included as a reference, as well as linear regression lines.*

Just for comparison, let's consider what we would expect to see if the X_is are independent of each other (but still all come from the same distribution). In that case, the variance of the sum is the same as the sum of the variances: $\mathbb{V}\mathrm{ar}(\sum^m X_i) = m\,\mathbb{V}\mathrm{ar}(X)$. Because averaging includes an additional factor of $\frac{1}{m}$, that is $\bar{X}_1^{(m)} = \frac{1}{m}\sum X_i$, the end result is

$$\mathbb{V}\mathrm{ar}(\bar{X}_1^{(m)}) = \frac{1}{m}\mathbb{V}\mathrm{ar}(X_1) \qquad\qquad X_1, X_2, \ldots \text{ independent}$$

In other words, the self-similarity affects the rate at which the variance is reduced with aggregation. If $H = \frac{1}{2}$ (and $\beta = 1$) the behavior is the same. But in the range $\frac{1}{2} < H < 1$ we have $0 < \beta < 1$, and then the variance is reduced more slowly than for a non-self-similar process. This is what we will use as a test for self-similarity. In addition, estimating β immediately translates into an estimate of H.

According to Equation (7.17), the variance decays polynomially with aggregation. By taking the log from both sides we get

$$\log(\mathbb{V}\mathrm{ar}(\bar{X}_1^{(m)})) = -\beta \log m + \log(\mathbb{V}\mathrm{ar}(X_1))$$

or alternatively

$$\log\left(\frac{\mathbb{V}\mathrm{ar}(\bar{X}_1^{(m)})}{\mathbb{V}\mathrm{ar}(X_1)}\right) = -\beta \log m \tag{7.18}$$

This second form may be slightly more convenient because the variance is normalized and always starts from 1. In either case, plotting the logarithm of the variance of the aggregated data as a function of the logarithm of the degree of aggregation should lead to a straight line, with a slope of $-\beta$. The Hurst parameter is then given by

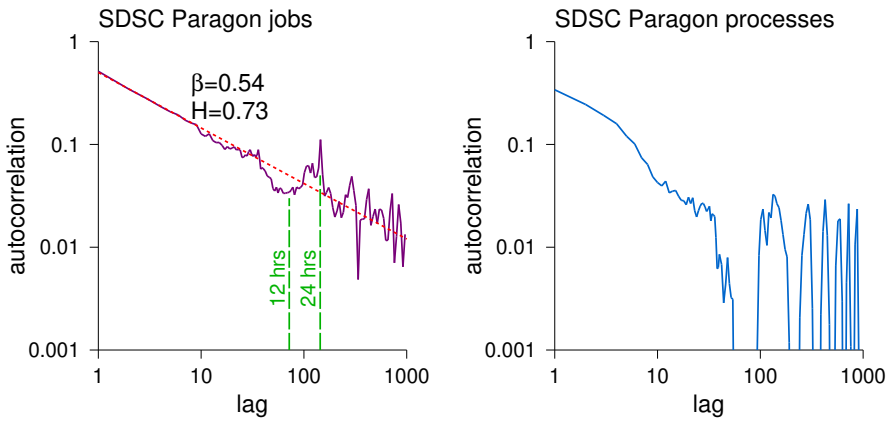


Figure 7.11: *Measuring the long-range dependence of the SDSC Paragon data from Figure 7.3.*

$$H = 1 - (\beta/2) \tag{7.19}$$

The resulting plot is called a *variance-time plot*. An example, using the same datasets used previously, is shown in Figure 7.10. It starts with an aggregation of 1 (that is, no aggregation) and continues up to an aggregation level of 32,768. This figure was chosen as the upper limit because the full dataset contains only a few more than a million samples (at 1-minute resolution). After aggregating, we are then left with 32 aggregated samples, and can calculate their variance. Further aggregation will leave us with too few samples.

7.4.4 Measuring Long-Range Dependence Directly

As noted earlier, self-similarity goes hand in hand with long-range dependence. This means that the autocorrelation function of the series of samples decays polynomially [69]:

$$r(k) \approx k^{-\beta} \qquad 0 < \beta < 1$$

Thus if we calculate the autocorrelation function, and display it as a function of the lag k in log-log axes, we will get a straight line with slope $-\beta$:

$$\log r(k) \propto -\beta \log k$$

This simple procedure is illustrated in Figure 7.11 The original data is at a 10-minute resolution. Consider first the jobs data. The autocorrelation functions becomes noisy at lags above one day. In addition, one can observe a dip in the autocorrelation corresponding to a lag of 12 hours, and a peak corresponding to 24 hours. Nevertheless, the linear relationship is clear. The slope of the line indicates that $\beta = -0.54$, leading to $H = 1 - \beta/2 = 0.73$ — which is in pretty good agreement with the previous results.

However, this procedure does not always work so well, as illustrated by the processes data. The data is much noisier, and even becomes negative at certain lags (notably 12 hours), which cannot be accommodated by a logarithmic plot. Thus it is hard to argue that this represents a linear relationship.

An additional problem with a direct characterization of the autocorrelation function is that it is susceptible to error caused by nonstationarity. Consider a process that is actually piecewise stationary, meaning that it is composed of several intervals that are each stationary but different from the others. Furthermore, assume that each such stationary process is not long-range dependent. Calculating the autocorrelation function of each individual process will therefore show that the autocorrelation decays quickly to zero. But computing the autocorrelation function of the complete composite process will show a much slower decay. This slower decay reflects the fact that the samples in each individual constituent process are indeed more correlated to each other than to samples coming from the other processes. Thus there exists a relatively high correlation up to lags that reflect the lengths of the constituent processes.

Finally, empirical estimation of the autocorrelation function is always limited to lags that are much smaller than the full length of the series, because sufficient data for longer lags is not available.

Because of these considerations, direct measurement of long-range dependence is not commonly used.

7.4.5 Using Wavelets and Logscale Diagrams

A completely different approach to analyzing and measuring scaling effects is to use multiresolution analysis with wavelets [6, 3, 5]. The idea is to start with a basic oscillating pattern of limited duration — the wavelet — and check its correlation to the data at different scales and shifts. Thus the structure of the analysis corresponds directly to the scale-invariant self-similar structure we are looking for. The result of the analysis is a space-time decomposition of the data, that is, a set of coefficients characterizing the relative importance of wavelets at different scales and shifts. The statistical properties of these coefficients, notably how they decay with scale, are then used to identify and quantify self similarity and long-range dependence.

Discrete Wavelet Transform with Haar Wavelets

More formally, wavelets are functions $\psi(t)$ with the following properties:

- They have a zero integral, i.e.,

$$\int_{-\infty}^{\infty} \psi(t)\mathrm{d}t = 0$$

  This implies that they must oscillate between negative and positive values; hence the wave metaphor.

- They are localized in time, meaning that in effect they have a limited support. Beyond this range they either decline very rapidly or are identically zero; hence the diminutive form, "wavelet".
- They also have a limited bandwidth around some basic frequency. The amplitude of other frequencies, if they exist, declines rapidly the farther they are from the basic frequency.

When using wavelets to analyze a signal (in our case a time series representing some workload feature, such as the number of arrivals in successive units of time), one uses not one but an entire family of wavelets. The basic one is called the *mother wavelet*. The others are translations and dilations of the mother wavelet. A translated wavelet is shifted in time, and hence matched with different parts of the signal. (This is meaningful because the wavelet has a bounded support, as noted above; conventional spectral analysis cannot provide this information, because its basis functions, which are sine waves of different frequencies and phases, extend to infinity.) A dilated wavelet is scaled to a different size. Both translations and dilations are typically done using powers of two. Thus the wavelet at scale j and shift k will be[1]

$$\psi_{j,k}(t) = \frac{1}{\sqrt{2^j}}\,\psi\left(\frac{1}{2^j}(t - 2^j k)\right) \tag{7.20}$$

In the argument of ψ, the term $-2^j k$ shifts the center of attention from 0 to $2^j k$. Multiplying by the factor $\frac{1}{2^j}$ increases the time unit (for positive j), so that the shift of $-2^j k$ is a shift of k such time units. For example, if $j = 3$ and $k = 6$, a wavelet focused on the interval $(0, 1)$ is stretched and shifted to focus on the interval $(48, 56)$: the time unit becomes $2^3 = 8$ times larger, and we shift by six such time units. The normalization factor $\frac{1}{\sqrt{2^j}}$ ensures that the integral of the squared function remains the same, so it has the same energy (in signal-processing applications). This is important so that the transform is invertible.

There are many different functions that may serve as the mother wavelet, each with its desirable properties. For our exposition we use the simplest and most intuitive one, which is the Haar wavelet. This is one cycle of a square wave, defined as

$$\psi(t) = \begin{cases} 1 & 0 \le t < 0.5 \\ -1 & 0.5 \le t < 1 \\ 0 & \text{otherwise} \end{cases}$$

It is shown for scales $j = 0$ to $j = 2$ in Figure 7.12. Note, however, that the Haar wavelet is rarely used in practice, because other wavelets have better properties for most applications. In particular, the Haar wavelet is discontinuous and therefore has a relatively large bandwidth, meaning that it is not well localized in the frequency domain.

Multiresolution analysis as done by wavelets is a recursive process. Assume we start with N samples (e.g., the number of arrivals in N consecutive time units), where $N = 2^n$. The simple case of the Haar wavelet proceeds as follows (Figure 7.13).

[1] More precise terminology is to call 2^j the scale, and to call j itself the octave.

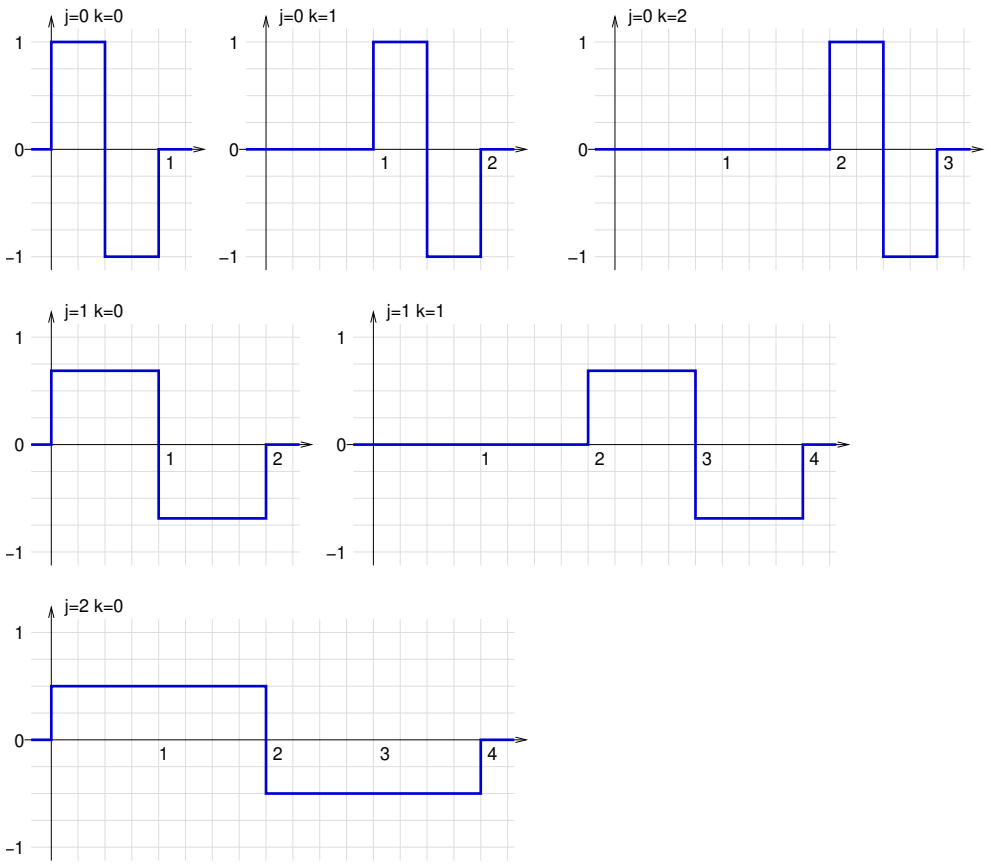


Figure 7.12: *The Haar wavelet for scales $j = 0$ (the mother wavelet), $j = 1$, and $j = 2$ and some shifts ($k = 0$ to 2).*

1. Start with the original samples as $X_1, X_2, \ldots, X_N$.

2. Create an *approximation* of the input. This is a smoothed version with less details. It is obtained by summing consecutive pairs of inputs and normalizing: $a_{1,k} = \frac{1}{\sqrt{2}}(X_{2k-1} + X_{2k})$.

3. Find the *details* that separate the original input from the approximation. These are the normalized differences between consecutive pairs of inputs: $d_{1,k} = \frac{1}{\sqrt{2}}(X_{2k-1} - X_{2k})$.

4. Separate the details from the approximation. Each of these has half the previous number of elements. This step is called *decimation*, because it is implemented by removing the even or odd elements. In Figure 7.13, the details are moved to the left and the approximations to the right.

5. Return recursively to step 2 using the approximation results as the input to the next iteration. In each iteration, the first index of the new approximations and details

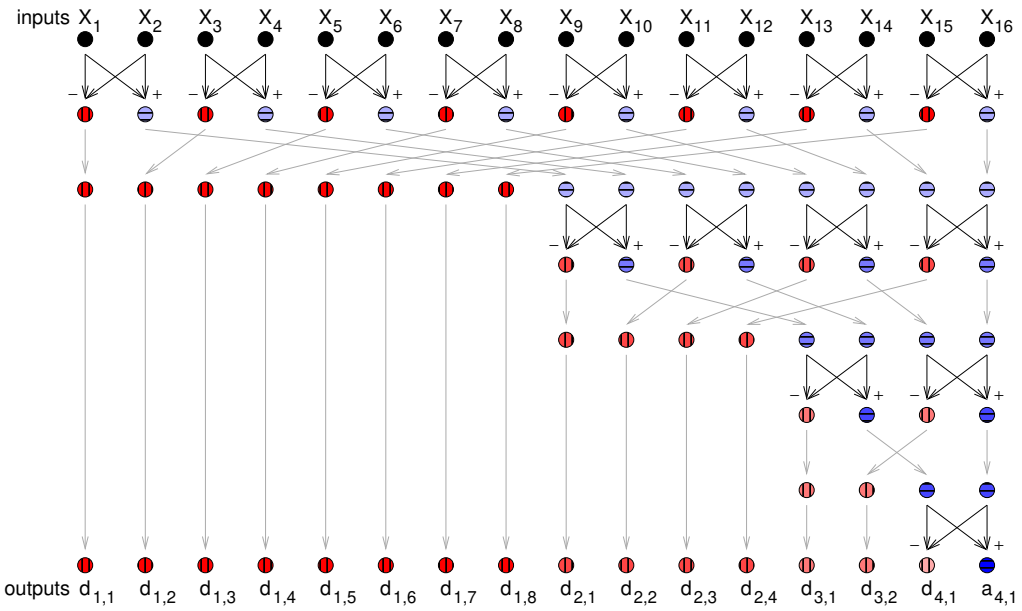


Figure 7.13: *The Haar wavelet transform.*

will denote the iteration number ($a_{2,k}$ and $d_{2,k}$ in the second iteration, $a_{3,k}$ and $d_{3,k}$ in the third, etc.).

6. At the end we are left with a single approximation that represents the normalized sum of the whole original input. The output is the set of all details from all the stages and this global approximation.

Note that, because we halve the data we are working with at each stage, and there are a logarithmic number of stages, the complexity of the entire procedure is linear in the input size.

The relationship between this description of the Haar transform and the previously described Haar wavelet may not be immediately apparent. The difficulty stems from the fact that the wavelet was introduced as a continuous function, but the transform operates on samples. These samples are obviously discrete, and are assumed to come at fixed intervals of time. The solution is to use a discretized version of the wavelet. For the Haar wavelet, two samples with values of 1 and –1 are natural. Each of the differences in step 3 above can be viewed as a convolution of the wavelet vector $(1, -1)$ with a pair of input samples (X_{2i-1}, X_{2i}), and normalization by a factor of $\frac{1}{\sqrt{2}}$. The convolution with the pair (X_1, X_2) represents the mother wavelet. The convolution with pair i, namely (X_{2i-1}, X_{2i}), represents a translation of $i - 1$ units. In the second stage, when we use the approximations $\left(\frac{1}{\sqrt{2}}(X_{2i-1} + X_{2i}),\ \frac{1}{\sqrt{2}}(X_{2i+1} + X_{2i+2})\right)$ as input pairs, this represents wavelets with dilation by a factor of 2. In subsequent stages, where approximations are sums of more inputs, this represents dilation to ever larger scales.

Details Box: The Discrete Wavelet Transform

To perform the multiresolution analysis in the general case, two functions are actually needed: the wavelet function and a scaling function. Denote the wavelet by $\psi(t)$, and the scaling by $\phi(t)$. In the discrete wavelet transform (DWT) both these functions are represented by a certain number of samples. In the Haar transform this was two samples, but in other cases it is usually four or more.

The scaling function is a low-pass filter, and the approximations are obtained by convolving the scaling function with the input. In the case of the Haar transform, the vector used is $(\frac{1}{\sqrt{2}}, \frac{1}{\sqrt{2}})$, which is like the average (it would be the average if not for the normalization by a factor of $\sqrt{2}$).

The wavelet function is a high-pass filter, and the details are obtained by convolving the wavelet with the input. In the case of the Haar transform, the vector used is $(\frac{1}{\sqrt{2}}, -\frac{1}{\sqrt{2}})$, which is like the distance from the average (again with the normalization factor).

More formally, denote the inputs at stage i by $X_{i,1}, \ldots, X_{i,N_i}$. The samples of the filter are denoted $\psi_1, \ldots, \psi_\ell$ (or $\phi_1, \ldots, \phi_\ell$). Note that N_i, the length of the data, is typically much longer than ℓ, the length of the filter, except in the last iterations.

Filtering is performed by convolving the data with the filter. Convolution consists of matching elements of the two vectors with each other at different offsets k. For each offset, the paired elements are multiplied with each other, and then the sum of all these products is computed. This is actually similar to the correlation operation, except that in convolution one of the vectors is reversed first.

In our case the convolution of vector X of length N_i with vector ψ of length ℓ is a vector C of length $N_i + \ell - 1$, whose elements are

$$C_k = \sum_{j=1}^{\ell} X_{k-j+1}\psi_j \qquad 1 \le k < N_i + \ell - 1$$

Elements of X beyond the original inputs (including negative indices) are taken as 0. The expression for ϕ is analogous.

Given these definitions, we can now describe the discrete wavelet transform as a generalization of the Haar transform described above. The procedure is as follows.

1. Denote the input at stage i by $X_{i,1}, \ldots, X_{i,N_i}$.
2. Create an *approximation* of the input by convolving the input with the low-pass filter based on the scaling function. Denoting convolution by $*$, this is
$$App_i^{full} = X_i * \phi$$
The specification ^{full} indicates that this is the full result of the convolution.
3. Find the *details* left out by the approximation by convolving the input with the high-pass filter based on the wavelet function. This is
$$Det_i^{full} = X_i * \psi$$
4. Decimate the full details, keeping one sample of two. This is typically denoted by $\downarrow_2$. Thus the details at this stage will be
$$Det_i = Det_i^{full} \downarrow_2$$
This is appended to the output of the transform.

5. Decimate the full approximation, again keeping one sample of two. Thus the approximation at this stage will be
$$App_i = App_i^{full} \downarrow_2$$
This is the input to the next stage, namely
$$X_{i+1} = App_i$$
6. Return recursively to step 2 to continue at the next coarser scale. This can be done for a predefined number of times, or until the approximation is reduced to a single value.
7. Append the last approximation(s) to the output, thus completing the transform.

Note that at each iteration we apply the high-pass filter of the wavelet to the results of the low-pass filters applied previously. The net effect is a band-pass filter. In other words, each iteration focuses on a different range of frequencies, corresponding to a different level of dilation.

End Box

An important property of the wavelet transform is that it is invertible: given the details and approximation of the transform, you can reconstruct the original input. This testifies to the fact that no information is lost. An example of the inverse transform for the Haar wavelet is shown in Figure 7.14. In essence, we start from the very coarse global approximation, and re-create finer approximations by adding in the details — eventually reaching the full input. As we'll see below in Section 7.5.2, this procedure can also be used to generate synthetic self-similar data based on details that are sampled from appropriate distributions.

Logscale Diagrams and Self-Similarity

The discrete wavelet transform is based on computing approximations, which are essentially aggregations of the input at different levels, and details, which are related to increments. The output of such a transform is a sequence of coefficients representing different translations and dilations. In particular, the first half represents details of translations of the highest resolution wavelet. The next quarter are translations of a slightly lower resolution, the next eighth translations of a yet lower resolution, and so on. The magnitudes of the different coefficients reflect the variability in the input. Thus, by measuring how the magnitude of the coefficients changes with the scale they represent, we can learn about the scaling of the variability — and hence about long-range dependence and self-similarity.

The procedure is simple [6, 7, 5]. For each scale j, we compute the average of the squares of the details at that scale $d_{j,k}$. The model is that this should be linear (in logscale) with j, and the slope should reflect the Hurst parameter H:
$$\log\left(\frac{1}{N_j}\sum_k d_{j,k}^2\right) = (2H-1)j \tag{7.21}$$
where N_j is the number of details at scale j, which is $N/2^j$. A diagram showing this average as a function of j using a logarithmic Y axis is called a *logscale diagram*.

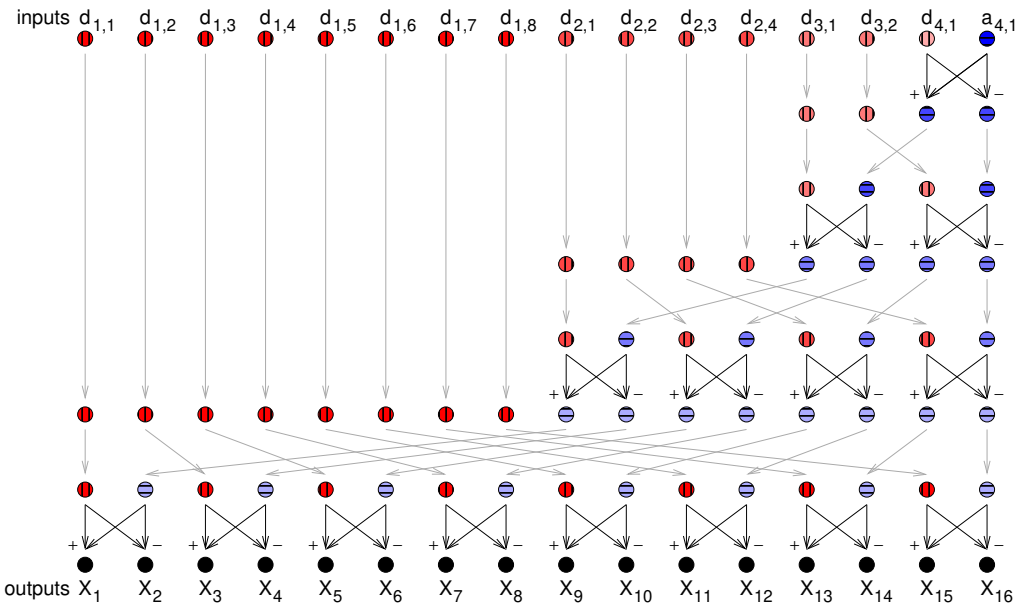


Figure 7.14: *The inverse Haar wavelet transform. Compare with Figure 7.13.*

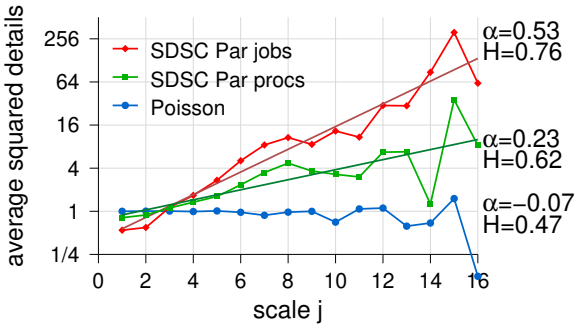


Figure 7.15: *A logscale diagram based on using the Haar wavelet transform to measure the self-similarity of the data in Figure 7.3. The details were normalized before the analysis to enable them to be shown on the same scale.*

An example, again using the same data as in the previous figures, is shown in Figure 7.15. Although the graphs appear a bit noisy, the results nevertheless correspond to those obtained by the other methods. The slope α is converted to an estimate of the Hurst parameter using the formula $H = \frac{1}{2}(\alpha + 1)$.

7.4.6 Spectral Methods: The Periodogram and Whittle Estimator

Previously we have examined our data in the time domain. This is natural because the data is typically a series of values for successive time instants: the number of packets in

the first time unit, the number in the second time unit, and so on. But it is also possible to perform the analysis in the frequency domain.

The material in this section is more mathematically advanced than in any other part of this book. In addition, it suffers from different conventions, normalizations, and notations being used by different authors. To facilitate an intuitive understanding of the underlying meaning we sometimes gloss over some details, especially in the translations from continuous or infinite-sequence definitions to estimations for finite sequences.

Long-Range Dependence in the Frequency Domain

In a nutshell, the characterization of long-range dependence is that the autocorrelation function decays according to a power law. This means that it decays slowly, or in other words, it retains its value for ever increasing lags. In the frequency domain such consistency implies a dominance of low frequencies. (The alternative, of rapid changes up and down, would imply a dominance of high frequencies.) In particular, an autocorrelation function that decays according to a power law has a spectrum that tends to infinity as the frequency tends to zero.

In mathematical notation, denote the autocovariance function by $\gamma(k)$; it has only one index due to the stationarity assumption. Given a data series of N elements with mean 0, the autocovariance function is estimated by

$$\gamma(k) = \frac{1}{N} \sum_{i=1}^{N-k} X_i \, X_{i+k} \tag{7.22}$$

where we use the normalization $\frac{1}{N}$ rather than $\frac{1}{N-k}$ because we need to use all N elements of the autocovariance. If we use $\frac{1}{N-k}$, the last elements (where k is close to N) may be large and erratic because they depend on very few samples, contradicting the common expectation that $\gamma(k)$ is small for large ks. We could also normalize the autocovariance function by the variance to obtain the autocorrelation function, but the spectrum would be the same so that step is not needed.

The spectral density is obtained by a discrete Fourier transform of the autocovariance function,

$$S(f) = \sum_{k=-(N-1)}^{N-1} \gamma(k) \, e^{-i2\pi f k} \tag{7.23}$$

where f is a frequency, $\gamma(-k) = \gamma(k)$ because the correlation of element j with element $j+k$ is the same as the correlation of element $j+k$ with element j, and $i = \sqrt{-1}$ (note that i is not an index but an "imaginary" number; see the box below for an explanation of this expression). The essence of this transformation is finding how similar the autocovariance function $\gamma(k)$ is to a sine wave of frequency f, for all relevant fs. The spectrum is a function of the frequency: for every frequency, it specifies the amplitude, or "strength", of this frequency in $\gamma(k)$. The important point is that $\gamma(k)$ can actually be decomposed into these frequencies; in other words, if we sum up waves with these frequencies and amplitudes, we will get the function $\gamma(k)$.

Given that the spectrum is just another representation of the same data, it is not surprising that long-range dependence may be defined based on either of these two representations. Using the autocovariance, we have seen that the definition is based on the autocovariance decaying with k according to a power law: $\gamma(k) \approx k^{-\beta}$ as $k \to \infty$, with $0 < \beta < 1$. The equivalent definition using the spectral density is that it grows asymptotically to infinity as the frequency goes down to zero [67, 538, 308]:

$$S(f) \approx C \frac{1}{f^{\alpha}} \qquad f \to 0,\ 0 < \alpha < 1 \tag{7.24}$$

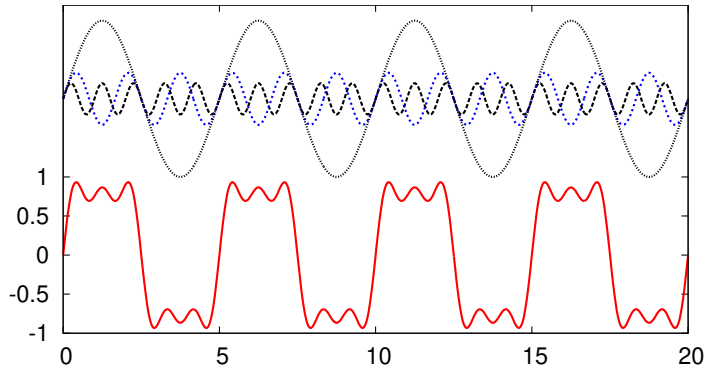
where C is a constant. This makes sense, because the essence of long-range dependence is that things change slowly, namely with a very low frequency.

Note that the growth of the spectral density near 0 is shaped like a hyperbola (i.e., also follows a power law). This shape of the spectrum has earned such functions the name "$1/f$ noise". Importantly, the exponents in the two power laws are both related to the Hurst parameter H: for the autocovariance we have $\beta = 2-2H$, and for the spectral density we have $\alpha = 2H-1$. Consequently, $\alpha = 1-\beta$, and both exponents are between 0 and 1.

The periodogram method for estimating H is based directly on Equation (7.24). The periodogram is an estimator for the spectral density, and has the same behavior at low frequencies. By calculating it and plotting it on log-log axes, one can use linear regression to estimate the slope α, and, from it, H.

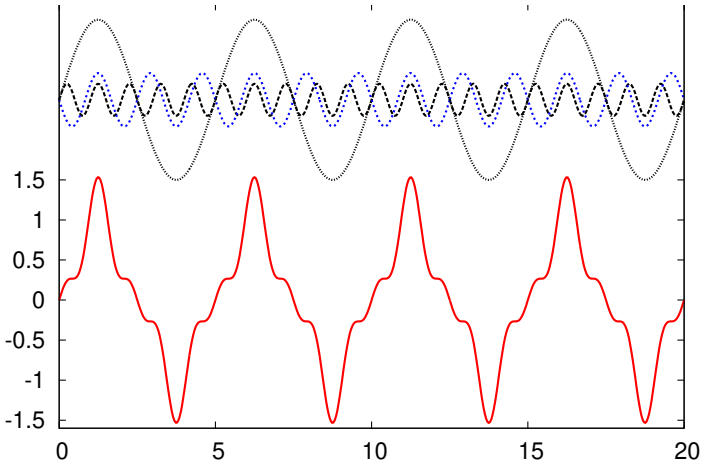
Background Box: The Discrete Fourier Transform

Before delving into spectral analysis, let us first understand what a spectral decomposition is all about. The following illustration depicts three sine waves in the time domain. The biggest sine wave has a frequency of $\frac{1}{5}$ — it completes one cycle in 5 time units. The others have frequencies that are three and five times higher. When we add them together we get the squarish wave pattern shown below them. So we can now describe this squarish wave in two ways: we can either show its graph, or we can say it is the sum of these three sine waves. The first one is a description in the time domain, whereas the second is a description in the frequency domain — in fact, its spectrum.



Importantly, the transformation from the time domain to the frequency domain is a transformation from one dimension to two dimensions. In the time domain, for each instant of

time we have only the height of the waveform at that instant. But in the frequency domain, we need two numbers to describe each frequency: its amplitude and its phase. The amplitude specifies the height of the sine wave with this frequency. The phase specifies how its starts at time 0: is it going up, or down, or maybe it is near the top? To show how important the phase is, the following illustration depicts another waveform that is composed of exactly the same three sine waves as the previous one. The only difference is in the phase of the middle sine wave: at time 0, it goes down instead of up.



The Fourier transform, which is how we compute the spectrum, works by correlating the input function with the sine waves that may be used to create it. The previous examples show continuous functions that in principle extend from $-\infty$ to ∞. But in time series analysis (and in digital signal processing) we are interested in discrete finite sequences, which we denote by X_k for $k = 0 \ldots N-1$. Therefore we need the discrete Fourier transform. The correlation is then computed by summing the products of the series elements and a sine at discrete points. If the series and the sine both go up together and go down together, the sum will be large, and this sine will be recognized as a major component of the series — and will have a high amplitude in the spectrum. If they do not, the sum will be small and this sine will have only a low amplitude in the spectrum. In mathematical notation, the sum is

$$F_i(f) = \sum_{k=0}^{N-1} X_k \sin(2\pi f k)$$

where f is the frequency, and we do this calculation for many different frequencies (to be defined later). The factor of 2π is needed because we define our frequencies relative to the data points, for example saying that the lowest frequency completes a single cycle across all N points, so it completes $\frac{1}{N}$ of the cycle per point. But the period of a sine is defined to be 2π when measured in radians, so we need to add $\frac{2\pi}{N}$ to the argument of the sine per point.

The preceding equation cannot be the whole story because it accounts only for the basic sine, which goes up at 0, and not for the phase. To capture the phase, we must also find the correlation with a cosine, because the sum of appropriately weighted sine and cosine is equivalent to a sine with a different phase (as you may remember from trigonometry class, $\alpha \sin(\omega t) + \beta \cos(\omega t) = r \sin(\omega t + \varphi)$ where $r = \sqrt{\alpha^2 + \beta^2}$ is the amplitude and $\varphi = \tan^{-1}(-\beta/\alpha)$ is the phase). We therefore also compute

$$F_r(f) = \sum_{k=0}^{N-1} X_k \cos(2\pi f k)$$

The last issue to settle is the set of frequencies for which we compute this. Given that the input is discrete, there is an upper bound on the relevant frequencies: any frequency that completes more than a full cycle over two points of the input is indistinguishable from a lower frequency that reaches the same height at those points. (This is called aliasing.) Using the interval between adjacent points as our unit, this means that the highest frequency of interest is $\frac{1}{2}$.

But note too that we start with only N data points. Therefore we don't need the infinite different frequencies that are smaller than $\frac{1}{2}$. Instead, we can make do with $N/2$ frequencies of the form $f_j = j/N$ for $j = 1 \ldots \frac{N}{2}$, called the Fourier frequencies. The lowest frequency, $f_1 = 1/N$, completes a single cycle across all the N data points. The others are multiples of this frequency, called harmonics. For example, in the figures shown above, the biggest sine wave is the basic frequency (assuming our unit is 5) and the other two are harmonics: they have frequencies that are three and five times higher.

We can now put all of this together and understand the meaning of the amplitudes we calculated. As noted earlier, the core idea is that the series we started with may be expressed as a sum of sine waves with appropriate frequencies, amplitudes, and phases. And this is equivalent to a sum of sines and cosines with appropriate frequencies and amplitudes. The frequencies are $f_j = j/N$ for $j = 1 \ldots \frac{N}{2}$. The required amplitudes are the $F_i(f_j)$ and $F_r(f_j)$ we computed. So we can write

$$X_k = \frac{1}{N} \sum_{j=1}^{N/2} [F_i(f_j) \sin(2\pi f_j k) + F_r(f_j) \cos(2\pi f_j k)] \tag{7.25}$$

(the normalization factor $\frac{1}{N}$ is explained later). The amplitudes are the unique part that embodies the information about a specific input series. Hence we started out with N data points in the time series, and transformed them into $\frac{N}{2}$ pairs $(F_i(f_j), F_r(f_j))$. Equivalently, we could transform them into $\frac{N}{2}$ pairs of amplitude and phase.

The common expression of the discrete Fourier transform calculates both elements of each pair at once by using complex numbers. Complex numbers have two parts: a real one and an imaginary one. The imaginary part is denoted as a multiple of $i = \sqrt{-1}$ (-1 does not have a square root, which justifies calling i imaginary). Moreover, the exponential function with an imaginary exponent is equivalent to the sum of the cosine and an imaginary sine:

$$e^{ix} = \cos(x) + i \sin(x)$$

(this is Euler's formula). The two preceding equations for F_i and F_r can therefore be written in shorthand as

$$F(f_j) = \sum_{k=0}^{N-1} X_k \, e^{-i2\pi f_j k} \tag{7.26}$$

where the two dimensions of the result are represented as a complex number: $F_r(f_j)$ is the real part, and $F_i(f_j)$ is the imaginary part.

As shown in Equation (7.25), the coefficients calculated by the Fourier transform can be multiplied by their respective sines and cosines to regain the original input series. Thus the inverse Fourier transform is, in shorthand,

$$X_k = \frac{1}{N} \sum_{j=1}^{N/2} F(f_j)\, e^{i2\pi f_j k} \tag{7.27}$$

This expression also shows that the set of sines with the given frequencies and phases are a basis, and can therefore be used to represent any function. The normalization factor $\frac{1}{N}$ is needed because when we substitute the spectral components of Equation (7.26) into the sum of Equation (7.27) — which is how we prove that this is indeed the inverse transform — we get NX_k.

The way to actually calculate the discrete Fourier transform is by using the fast Fourier transform algorithm. Using a naive calculation based on the above formulas requires $O(N^2)$ operations to transform N points. With the fast Fourier transform, only $O(N \log N)$ operations are required [70, 150, chap. 30]. When N is large, this makes a significant difference.

An often confusing aspect of the Fourier transform is the use of k that range from $-N$ to N. This is not really a problem provided the function being transformed is even, meaning that $X_k = X_{-k}$, which is the case for the autocovariance function [308, 121]. The symmetric sum is then related to the above definition (Equation (7.26)) by a multiplication by 2 and adding the variance.

To read more: The Fourier transform is a huge subject. This box provided merely a brief exposition of one variant; the more commonly studied variant is the continuous transform. There are many books about Fourier analysis in general and spectral methods for time series analysis in particular. The classic basis for many of them is the two-volume set by Zygmund [767]. A good book devoted to spectral analysis is Stoica and Moses [664]. More general texts on time series (including chapters on spectral analysis) have been written by Chatfield [121] and by Shumway and Stoffer [619].

End Box

The Periodogram

Spectral methods to estimate the Hurst parameter H are naturally based on the spectral representation. Specifically, the periodogram is defined to be the square of the Fourier transform of the original time series:

$$I(f_n) = \frac{1}{N} \left| \sum_{k=1}^{N} X_k\, e^{-i2\pi f_n k} \right|^2$$

At the Fourier frequencies, $f_n = n/N$, this is in fact equivalent to the spectral density of Equation (7.23) (as explained and justified below). Therefore the behavior has the same relationship to the Hurst parameter when the frequency tends to 0:

$$I(f) \approx S(f) \approx \frac{C}{f^{2H-1}} \qquad f \to 0$$

Taking the log, we find that

$$\log I(f) \approx (1 - 2H) \log f + C'$$

at least for the lowest frequencies. So if we calculate the periodogram and graph it as a function of frequency on log-log axes, fitting a straight line to the low frequencies will enable us to estimate H: if the slope of the line is α, then $H = \frac{1-\alpha}{2}$ [283].

And now for the details. The periodogram is another representation of the spectrum, or rather the power spectrum. We start out with the discrete Fourier transform of the original time series (*not* its autocovariance). For each frequency $f_n = n/N$, this is

$$F(f_n) = \sum_{k=1}^{N} X_k \, e^{-i2\pi f_n k}$$

As explained earlier, this expression identifies the contribution of each frequency to the time series, and is typically used to identify periodic behavior. But here we use it to characterize the asymptotic behavior at low frequencies near 0.

The periodogram is the square of the Fourier transform normalized by $\frac{1}{N}$:

$$I(f_n) = \frac{1}{N} \left| F(f_n) \right|^2 = \frac{1}{N} \left| \sum_{k=1}^{N} X_k \, e^{-i2\pi f_n k} \right|^2 \tag{7.28}$$

The reason for specifying that the absolute value is being squared is that we are dealing with complex numbers. The square of the absolute value is then the sum of the squares of the real and imaginary parts.

The connection with the spectral density defined earlier is that the periodogram turns out to behave like the spectral density, that is, the Fourier transform of the autocovariance function:

$$I(f_n) \approx S(f_n) = \sum_{k=-(N-1)}^{N-1} \gamma(k) \, e^{-i2\pi f_n k} \tag{7.29}$$

To see this, let's start from the original definition. Algebraically, to find the squared absolute value of a complex number, you don't multiply it by itself; instead, you multiply it by its complex conjugate, in which the imaginary part has the opposite sign. To see this, consider a complex number $c = a + ib$ and its conjugate $c^* = a - ib$. Their product is $cc^* = (a + ib)(a - ib) = a^2 - (ib)^2 = a^2 + b^2$, because $i^2 = -1$. Applying this to Equation (7.28), we get

$$I(f_n) = \frac{1}{N} \left(\sum_{k=1}^{N} X_k \, e^{-i2\pi f_n k} \right) \left(\sum_{\ell=1}^{N} X_\ell \, e^{i2\pi f_n \ell} \right)$$

(the second set of parentheses is the complex conjugate of the first, because the sign of the exponent is +). Opening the parentheses leads to

$$I(f_n) = \frac{1}{N} \sum_{k=1}^{N} \sum_{\ell=1}^{N} X_k \, X_\ell \, e^{-i2\pi f_n (k-\ell)}$$

What we have here is a sum with N^2 terms, arranged in a square: first we sum on k (call this the rows), and for each k we sum on ℓ (call this the columns). But we can

Figure 7.16: *A periodogram used to measure the self-similarity of the data in Figure 7.3.*

also rearrange the order of summation, and sum along diagonals where $h = k - \ell$ is a constant. The expression then turns into

$$I(f_n) = \sum_{h=-(N-1)}^{N-1} \left(\frac{1}{N} \sum_{k=1}^{N-|h|} X_k \, X_{k+|h|} \right) e^{-i2\pi f_n h}$$

The factor in parentheses in the middle is the autocovariance $\gamma(h)$ of Equation (7.22). Note that this is indeed defined for negative lags, and in fact $\gamma(h) = \gamma(-h)$: for a stationary series, looking at the correlation of each element with the one that comes h places before it is the same as looking at the correlation with the one that comes h places after it. The entire expression is therefore essentially Equation (7.29).

An example of the results of performing such calculations is shown in Figure 7.16. Note that most of the data is not used, because the estimate is based on only the lowest frequencies. Figure we uses frequencies smaller than 0.01, which is only 1% of the data. This is less than the 10% that is sometimes mentioned in the literature, because our dataset is much longer. Even when using only such low frequencies the spread is rather large, and the correlation between the frequencies and the periodogram values is weak. Nevertheless, using a linear regression produces a line with a slope that corresponds to the results of other techniques.

An estimation of the Hurst parameter H based on the periodogram uses only the lowest frequencies, because it is based on the shape of the periodogram as the frequency

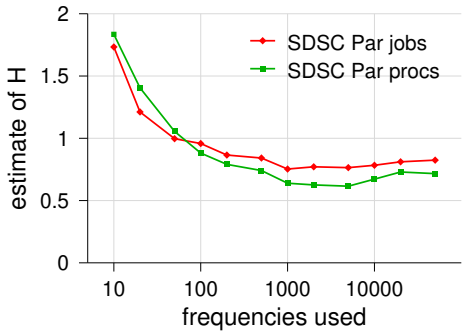


Figure 7.17: *Dependence of the estimate of the Hurst parameter H on the number of frequencies in the periodogram used in the regression.*

tends to 0. But the question of exactly how many frequencies to use has no good answer. An often cited rule of thumb is to use $\sqrt{N}$ frequencies for series with N points. In Figure 7.16, which has a bit more than a million points, that would imply using the 1000 lowest frequencies. An alternative approach is to do the analysis with several different cutoffs, and then eyeball the results to see which looks the most convincing. An example is shown in Figure 7.17. The graph shows that for very few frequencies, e.g. less than a hundred, the resulting estimate is unreliable: it comes out larger than 1. Using 1000 frequencies or more leads to more stable results, but still there are fluctuations of some 10%. The results quoted in Figure 7.16, using frequencies up to 0.01, actually used 5281 frequencies.

To read more: For a discussion of several methods to estimate the periodogram, including various smoothing techniques that are designed to reduce variance, see Stoica and Moses [664, chap. 2].

The Whittle Estimator

The Whittle estimator takes a somewhat roundabout route to estimating the Hurst parameter H, although the final procedure is reasonably simple [268, 67]. Initially one must specify a mathematical model of a time series that exhibits long-range dependence. The models that are typically used are fractional Gaussian noise or fractional ARIMA, which are outlined later in Section 7.5.1. Then an iterative procedure is used to find the parameter H that leads to a maximum likelihood fit of this model to the data. The maximum likelihood is obtained by minimizing the sum given in Equation (7.35) below. Assuming that the model indeed represents the data faithfully, this is expected to be a good estimate of the Hurst parameter of the data. However, if the original model was wrong, the estimate might be badly biased.

The technicalities of how the expression in Equation (7.35) is derived are rather involved, so we only give a superficial sketch here. We start with a series of observations $x_1, x_2, \ldots x_N$. The underlying conceptual model is to fit them to a multinormal distri-

bution with a certain correlation structure. The multinormal distribution (or multivariate normal distribution) is the joint distribution of multiple normal random variables. Let $\vec{\mu}$ denote the vector of their expectations — that is, $\mu_1 = \mathbb{E}[X_1]$, $\mu_2 = \mathbb{E}[X_2]$, and so on up to $\mu_N = \mathbb{E}[X_N]$. Let $\sum$ denote the covariance matrix, i.e.,

$$\Sigma = \Big\{\gamma(X_i, X_j)\Big\}_{1\le i,j\le N} = \begin{pmatrix} \gamma(X_1,X_1) & \gamma(X_1,X_2) & \dots & \gamma(X_1,X_N) \\ \gamma(X_2,X_1) & \gamma(X_2,X_2) & \dots & \gamma(X_2,X_N) \\ \vdots & & \ddots & \vdots \\ \gamma(X_N,X_1) & \gamma(X_N,X_2) & \dots & \gamma(X_N,X_N) \end{pmatrix}$$

where

$$\gamma(X_i, X_j) = \mathbb{E}[(X_i - \mu_i)(X_j - \mu_j)]$$

Thus $\sum$ embodies the correlation structure, which is what interests us. $\vec{\mu}$ and $\sum$ are the parameters of the distribution. Once they are given, the probability density function $f(\vec{x})$ is defined as

$$f(\vec{x}) = \left(\frac{1}{2\pi}\right)^{\frac{N}{2}} \left|\Sigma\right|^{-\frac{1}{2}} e^{-\frac{1}{2}(\vec{x}-\vec{\mu})^T \Sigma^{-1} (\vec{x}-\vec{\mu})}$$

where $\left|\Sigma\right|$ is the determinant of Σ, and Σ^{-1} is its inverse. Note that when $N = 1$ the covariance matrix $\sum$ becomes just the variance, and the entire expression becomes the well-known pdf of the normal distribution.

Given a set of observations $\vec{x}$, the likelihood of making these specific observations is given by the probability density function at $\vec{x}$. Assuming that $\vec{x}$ comes from a multinormal distribution, and taking the log of the likelihood to simplify the expression, leads to

$$\begin{aligned} \log L(\vec{\mu}, \Sigma \,|\, \vec{x}) &= \log f(\vec{x} \,|\, \vec{\mu}, \Sigma) \\ &= -\tfrac{N}{2}\log(2\pi) - \tfrac{1}{2}\log\left|\Sigma\right| - \tfrac{1}{2}(\vec{x}-\vec{\mu})^T \Sigma^{-1} (\vec{x}-\vec{\mu}) \end{aligned}$$

In our case the process is assumed to be stationary, so the elements of $\vec{\mu}$ are all the same. Moreover, the mean μ can be estimated efficiently as $\hat{\mu} = \bar{X}$, that is, the average of the observations. The process can therefore be centered by subtracting the average $\bar{X}$ from all elements. The resulting centered process has 0 mean. To simplify the notation, we will assume $\vec{x}$ denotes this centered process rather than the original data. This leaves us with

$$\log L(\mu, \Sigma \,|\, \vec{x}) = -\frac{N}{2}\log(2\pi) - \frac{1}{2}\log\left|\Sigma\right| - \frac{1}{2}\,\vec{x}^T\,\Sigma^{-1}\vec{x} \tag{7.30}$$

where $\sum$, which expresses the correlation structure, is the unknown that we still need to estimate. To do so we want to find the $\sum$ that maximizes this expression, and therefore is the most likely to have given rise to the observations. $\sum$ in turn depends on the Hurst parameter H, so by estimating $\sum$ we will be able to derive an estimate for H.

Note that the first term in the expression is constant, and does not involve $\sum$. Therefore it can be dropped from consideration. Our problem is that $\sum$ itself is an $N \times N$ matrix, and N can be very large. Moreover, for values of H near 1, there are numerical

problems: Σ's determinant may be vanishingly small, and the ratio between its maximal and minimal eigenvalues may be very large. We therefore cannot handle this expression explicitly.

The approach we will use is to transform the representation of Σ from the time domain to the frequency domain. Assuming that long-range dependence is present, the dominant elements of the spectrum are those at low frequencies. We can therefore use approximate expressions for $\left|\Sigma\right|$ and Σ^{-1} that are more accurate for low frequencies. This essentially means that the approximations become better for large N.

By expressing the autocovariance function using the spectral density (i.e., inverting Equation (7.23) using Equation (7.27)), the covariance matrix may be written as

$$\Sigma \approx \left\{ \frac{1}{N} \sum_{j=1}^{N/2} S(f_j)\, e^{i2\pi(r-s)f_j} \right\}_{1 \le r,s \le N}$$

where $f_j = j/N$ are the Fourier frequencies. Its inverse can then be approximated as

$$\Sigma^{-1} \approx \left\{ \frac{1}{N} \sum_{j=1}^{N/2} \frac{1}{S(f_j)}\, e^{i2\pi(r-s)f_j} \right\}_{1 \le r,s \le N} \tag{7.31}$$

where the quality of the approximation improves as N tends to ∞. To get a notion of why such an approximation may be reasonable, note that any symmetric matrix with unique eigenvectors A can be written as $A = U\Lambda U^T$, where U is a matrix of its eigenvectors and Λ has its eigenvalues along the main diagonal and zeros elsewhere. This simply shows that the effect of A is equivalent to a transformation to the basis of its eigenvectors, a scaling according to its eigenvalues, and a transformation back to the original basis. Its inverse is then $A^{-1} = U\Lambda^{-1}U^T$. This follows because the product $U\Lambda U^T U\Lambda^{-1}U^T$ is easily seen to be the identity matrix I. Starting from the middle, $U^T U = I$ because the eigenvectors are orthonormal, $\Lambda\Lambda^{-1} = I$ by definition, and $UU^T = I$ because again the eigenvectors are orthonormal. But given that Λ is a diagonal matrix, its inverse is simply a diagonal matrix whose elements are the reciprocals of the elements of the original matrix. In other words,

$$\Lambda = \begin{pmatrix} \lambda_1 & & 0 \\ & \ddots & \\ 0 & & \lambda_N \end{pmatrix} \quad \text{implies} \quad \Lambda^{-1} = \begin{pmatrix} \frac{1}{\lambda_1} & & 0 \\ & \ddots & \\ 0 & & \frac{1}{\lambda_N} \end{pmatrix}$$

This even makes sense: it simply says that the scaling is reversed by using the reciprocals of the scaling factors.

Let us now return to Equation (7.30). We need to approximate the two terms $\log\left|\Sigma\right|$ and $\vec{x}^T \Sigma^{-1} \vec{x}$. The first is easy. The determinant of a matrix A is the product of its eigenvalues:

$$|A| = \prod_{i=1}^{N} \lambda_i$$

This implies that its log will be

$$\log|A| = \sum_{i=1}^{N} \log \lambda_i$$

But the spectral components of $\gamma(k)$ approximate the eigenvalues of $\sum$ [300]. Therefore we have

$$\log\left|\Sigma\right| \approx \sum_{j=1}^{N/2} \log S(f_j) \tag{7.32}$$

Next, we approximate $\vec{x}^T \Sigma^{-1} \vec{x}$. The meaning of this expression is simply

$$\vec{x}^T \Sigma^{-1} \vec{x} = \sum_{r=1}^{N} \sum_{s=1}^{N} x_r \, \Sigma_{r,s}^{-1} \, x_s$$

which, based on Equation (7.31), is

$$\approx \sum_{r=1}^{N} \sum_{s=1}^{N} \frac{1}{N} \sum_{j=1}^{N/2} \frac{1}{S(f_j)} \, e^{i2\pi(r-s)f_j} \, x_r \, x_s$$

By changing the order of summation we derive

$$\approx \sum_{j=1}^{N/2} \frac{\frac{1}{N} \sum_{r=1}^{N} \sum_{s=1}^{N} x_r \, x_s \, e^{i2\pi(r-s)f_j}}{S(f_j)}$$

But we can exploit the fact that the covariance is the same along the diagonals of $\sum$, just as we did with the periodogram, by rearranging the sum: instead of summing row after row, we sum diagonal after diagonal. The index used for diagonals will be k. All this leads to

$$\approx \sum_{j=1}^{N/2} \frac{\sum_{k=-(N-1)}^{N-1} \left(\frac{1}{N} \sum_{t=1}^{N-|k|} x_t \, x_{t+k} \right) e^{-i2\pi k f_j}}{S(f_j)}$$

The parentheses are again just the empirical estimate of the autocovariance at a lag of k (Equation (7.22)). Therefore we get

$$\approx \sum_{j=1}^{N/2} \frac{\sum_{k=-(N-1)}^{N-1} \gamma(k) \, e^{-i2\pi k f_j}}{S(f_j)}$$

The numerator is now seen to be the approximate periodogram of Equation (7.29), leading to the final expression

$$\vec{x}^T \sum^{-1} \vec{x} \approx \sum_{j=1}^{N/2} \frac{I(f_j)}{S(f_j)} \tag{7.33}$$

Given the results of Equation (7.32) and Equation (7.33) we can now combine them to obtain an approximation of the desired log-likelihood function of Equation (7.30). This gives

$$\log L(\mu, \sum | \vec{x}) \approx -\sum_{j=1}^{N/2} \left[\log S(f_j) + \frac{I(f_j)}{S(f_j)} \right] \tag{7.34}$$

Recall that all this is an approximation of the log-likelihood of the multinormal model. We need to find the $\sum$ that maximizes this expression. To further simplify the expression, it turns out that the first term can be made equal to 0 by an appropriate normalization [67, p. 111]. We are then left with the second term only. Because of the minus sign, maximizing this is equivalent to minimizing its complement. The maximum likelihood is therefore obtained by minimizing

$$\sum_{j=1}^{N/2} \frac{I(f_j)}{S(f_j)} \tag{7.35}$$

This can also be understood as simply comparing the empirical periodogram $I(f_j)$ estimated from the data with the theoretical spectrum $S(f_j)$ computed from the model.

To summarize, this entire discussion boils down to a rather simple procedure:

1. Select a long-range dependent model, such as fractional Gaussian noise.
2. Use an iterative procedure to minimize Equation (7.35) as a function of the Hurst parameter H. To do so,
   (a) Calculate the periodogram $I(f_j)$ from the data using the fast Fourier transform.
   (b) Derive the autocovariance function for the selected model with the current value of H, and use this to calculate the spectral density $S(f_j)$ using the fast Fourier transform.
   (c) Calculate the sum of Equation (7.35) for this value of H, and continue with the iterations to find the value that minimizes the sum.
3. The value of H that leads to the minimal value (highest likelihood) is the estimate of the Hurst parameter of the data.

Importantly, given the theoretical background for this estimator, it is possible to show that the distribution of the estimate $\hat{H}$ is asymptotically normal, and that its variance is reduced with $\sqrt{N}$. As a result it is also possible to calculate confidence intervals for $\hat{H}$.

As noted, the Whittle estimator is based on the assumption that we have a good model of the underlying process. In particular, it is assumed that the samples come from

Method	*jobs*	*processes*
$(R/S)_n$	0.79	0.68
variance-time	0.76	0.66
autocorrelation	0.73	–
wavelets	0.76	0.62
periodogram	0.76	0.62
Whittle	0.72	0.61

Table 7.1: *Results of different methods to estimate the Hurst parameter H of job and process arrivals at the SDSC Paragon parallel supercomputer.*

a multinormal distribution. If this is not the case, the method may generate misleading results. A possible improvement, applicable if enough data is at hand, is to use the aggregated Whittle estimator [676]. The idea is to divide the data series into segments and aggregate them. Assuming finite variance, the distribution of such aggregates will be closer to a normal distribution.

Another variant is the local Whittle estimator [676]. The difference is that instead of specifying a complete model (e.g., fractional Gaussian noise), one specifies only the behavior of the spectral density near 0. After all, this is all we really need for Equation (7.35). Given that we are looking for long-range dependence, this is

$$S(f) \sim G\, f^{1-2H}$$

where G is a constant that also depends on H. To improve accuracy, Equation (7.34) is used with this specification, leading to the minimization of

$$\sum_{j=1}^{m} \left[\log G\, f_j^{1-2H} + \frac{I(f_j)}{G\, f_j^{1-2H}} \right]$$

where $m < \frac{N}{2}$ is a parameter that limits the sum to using only the lower frequencies.

7.4.7 Comparison of Results

Table 7.1 summarizes the results of estimating the Hurst parameter H by the different methods surveyed in the preceding sections. These results are remarkably consistent. This lends them credence, but still we don't really know whether they are correct.

A detailed study comparing the performance of several estimation methods (including most of those described above, as well as a host of others) was conducted by Taqqu and Teverovsky [677]. They performed the tests using synthetic data for which the correct answer is actually known in advance. The winner was the Whittle estimator, but only when the parametric form of the time series used was known. If this was not the case, and the local Whittle estimator was used, the results were not as good.

Another comparison, using a similar methodology, was performed by Karagiannis et al. [395]. They found that the periodogram method produced the best results, with the

Whittle estimator and R/S analysis being relatively close. In general, results tended to underestimate the Hurst parameter, and the difference grew as H tended to 1.

A third study of this type was conducted by Clegg [143]. It found that the R/S method has a slight tendency to underestimate results, and that a wavelet-based estimator tended to provide the best results. All the methods checked were susceptible to strong short range correlations (introduced by adding an AR(1) noise term created with $X_t = 0.9X_{t-1} + \varepsilon_t$; see page 361), and overestimated H considerably when such correlations were present. But spectral methods such as the local Whittle estimator and wavelets were able to cope with a superimposed periodic sine wave and a linear trend.

A number of additional studies have also compared wavelet-based approaches to Whittle and other approaches [6, 340, 379]. These generally show the wavelet approach to be better, in terms of both reduced bias and variability (and hence, tighter confidence intervals). Moreover, the wavelet approach has lower computational complexity. However, the Whittle estimator converges faster, and about 1000 samples are enough to get a good estimate. In contrast, wavelets require about 10,000 samples.

Finally, a survey of the complexity and theoretical statistical properties of the different algorithms has been written by Bardet et al. [56].

7.4.8 Validation

It should be noted that self-similarity is defined for infinite time series. Real data is always finite. And we do not know how the data would behave if we had more of it. Thus it is never possible to demonstrate conclusively that data is indeed self-similar. The best we can do is claim that it is consistent with self-similarity.

A practical way that may be useful to characterize the long-term behavior of the data is to use a "dynamic" approach, and emulate how the analysis would unfold if more and more data became available [260]. To do so, start with only a small initial fraction of the data, and use it to estimate the Hurst parameter. Now extend the data in several steps, until you finally use all of it, each step considering an order of magnitude more data than the previous one. In each step, repeat the estimation of the Hurst parameter using the data available in that step. If the estimates for H converge, and, moreover, the confidence intervals become more focused, this is a good indication that the data is consistent with self-similarity.

An important feature of this methodology is its emphasis on using all the available data. However, in some cases the analysis may show that the data is consistent with self-similarity only for some limited range of scales, and in particular, does not seem to be consistent with self-similarity on the longest time scales observed. While such self-similarity on a finite scale is a mathematical oxymoron, it can nevertheless be used to characterize the data behavior.

The problem with applying different procedures for measuring self similarity or long-range dependence is that any procedure will always produce a result, but still this result may be an artifact. For example, it has been demonstrated that short-range dependencies may also lead to estimates of the Hurst parameter that differ significantly from 0.5 [143].

To ensure that the results are indeed reliable, it has been suggested to chop the data into relatively short intervals, and then reorder them randomly [214]. Specifically, the length of intervals to use should correspond to the threshold above which any correlation will be considered to be "long-range". The random reordering then destroys any long-range dependencies that may have been present. Now apply your favorite estimation procedure to this reordered data. If it still produces a high value for the Hurst parameter, you know that this is due to an artifact. If the estimated Hurst parameter drops to 0.5, the original estimate is indeed reliable.

Other manipulations of the data can be used to assess the origin of the observed self-similarity or long-range dependence. The idea it to replace some specific candidate data field with randomized or homogenized values, and then repeat the analysis [338]. For example, if heavy-tailed flow sizes are suspected as the cause of self-similarity, replace all the flow sizes by their average, leaving all other attributes of the workload intact (for example, flow arrival times). If this manipulation causes the apparent self-similarity to disappear, this is evidence that the self-similarity indeed emerged as a result of heavy-tailed flow sizes.

The interaction and relative effect of short-range dependence and long-range dependence can be investigated by different manipulations of data that had been chopped up into intervals as described above. By mixing the intervals we break up the long-range dependence, but retain the short-range dependence. By leaving the intervals in the same order, and only mixing the jobs *within* each interval, we retain the long-range dependence but destroy any short-range dependence [214]. Thus we can create workload variants with any desired combination of long-range and short-range dependence, and check their effect on performance evaluations.

Mixing up the workload can also be done in other ways. For example, user resampling is a methodology to create multiple versions of the same basic workload by partitioning it into the subtraces representing different users, and then recombining them randomly to create new workloads. Interestingly, in the context of parallel job workloads at least, it has been shown that such manipulations do not affect the self-similarity of the workload [751]. This lends support to models such as the merged on-off processes described later.

7.4.9 Software for Analyzing Self-Similarity

Several software packages have been created to aid with the analysis of self-similar data, and in particular, with estimating the Hurst parameter.

Murad Taqqu, one of the preeminent researchers in the field, has a set of scripts and examples available on his website. The URL is http://math.bu.edu/people/murad/methods/.

The R statistics project has a package for evaluating and modeling long-range dependence time series. It includes Taqqu's software, as well as the programs that appear in Beran's book [67, chap. 12]. This package was used to create the periodogram of Figure 7.16 and calculate the Whittle estimator for Table 7.1; it is available at URL http://cran.r-project.org/src/contrib/Archive/fSeries/fSeries_240.10067.tar.gz.

Thomas Karagiannis implemented a tool called SELFIS [394], which is available

at URL http://www.cs.ucr.edu/%7Etkarag/Selfis/Selfis.html. It provides not only all the main methods to estimate the Hurst parameter, but also validation by shuffling segments of the data.

7.5 Modeling Self-Similarity

To model self-similarity we need to create an arrival process with long-range dependence, which is bursty at many different time scales. One approach to do so is to find the simplest and most elegant mathematical construction with these properties (an example of abstract descriptive modeling). Two candidates are fractional Gaussian noise and fractional ARIMA (autoregressive integrated moving average). An alternative is to find a mechanistic technique that gives the desired results. Two candidates for this second approach are using the inverse wavelet transform and bias models. Both these models use a hierarchical multiscale construction to build up a workload with features that persist across multiple scales.

A third approach is to construct a generative model that is based on how activity is really created in computer systems (sometimes called "modeling at the source" in this context) [731]. One such model is based on the M/G/∞ queue. In essence, the idea is that load (e.g. communication traffic) is generated by multiple sources, each of which is active for a time period that comes from a heavy-tailed distribution. Another related model is based on merged on-off processes, in which each on-off process represents the activity pattern of a single user or application, and their cumulative effect is the complete workload. It turns out that if the active and inactive periods (the on and off periods) are heavy-tailed, the resulting workload will be self-similar. Thus in both models the use of heavy-tailed distributions leads to a workload model with the desired long-range correlation.

We now survey these models in turn.

7.5.1 Classical Long-Range Dependent Models

Time series analysis is based on the need to understand — and even better, predict — the development of different processes, and has a long history. In particular, the modeling of dependent processes is based on the notions of autoregression and moving averages. Variants of these can also be used to model long-range dependent data, as shown later.

Background Box: From AR to ARIMA

An *autoregressive process* (abbreviated AR) is one in which each sample can be expressed as a function of previous samples. The simplest case is an AR(1) process, in which only one previous sample is used. The model is then

$$X_i = \phi X_{i-1} + \varepsilon_i$$

where ϕ is a constant and ε_i is an added noise term. Note that if ϕ is negative the process will tend to oscillate, but if it is positive it will tend to be more persistent. The most commonly used noise is Gaussian noise, in which ε_i is a normally distributed random

variable with mean 0 and variance σ^2. The more general case, denoted AR(p), allows dependence on p previous samples. The expression is then

$$X_i = \sum_{j=1}^{p} \phi_j X_{i-j} + \varepsilon_i \tag{7.36}$$

A *moving average process* (abbreviated MA) is defined in terms of another process, which is random. Assume that $\{Z_i\}$ are independent random variables, say from a normal distribution. We can then define an MA(q) process as

$$X_i = \sum_{j=0}^{q} \theta_j Z_{i-j} \tag{7.37}$$

that is, the weighted average of the $q+1$ elements leading to position i. Because successive averages have q elements in common, the resulting Xs will be dependent. Usually $\theta_0 = 1$.

Putting the two together, we get an *autoregressive moving average process* (naturally called ARMA(p, q)). This is an autoregressive process in which the noise is a moving average. The definition is

$$X_i = \sum_{j=1}^{p} \phi_j X_{i-j} + \sum_{j=0}^{q} \theta_j Z_{i-j} \tag{7.38}$$

Note that the structure of the process is defined by two parameters: p is the depth of the autoregression, and $q + 1$ is the window size of the moving average.

Finally, we can consider these processes as an increment process, and turn to the cumulative process. Doing so allows for modeling correlations in nonstationary processes, specifically processes with a polynomial (including linear) trend. Because the cumulative process is an integration of the increment process, it is called an *autoregressive integrated moving average process* (ARIMA(p, d, q)). It has three parameters. The p and q serve the same purpose as in ARMA processes. The d specifies how many times we need to difference in order to arrive at an ARMA process. If first differences are used, $d = 1$, it means that our process is the cumulant of an ARMA process, and will exhibit a linear trend. For higher d, it is the cumulant of the cumulant (repeated d times) of an ARMA process. Obviously, this is the most general process, and subsumes the previous ones as special cases.

Returning to the basic definitions of AR and MA, we may observe that a moving average process is actually just a linear combination of (Gaussian) white noise. Thus this type of process is also called a *linear process*. In MA processes the linear combinations used are finite, and involve $q + 1$ noise terms.

An autoregressive process, in contrast, can be viewed as one in which a linear combination of the observations gives noise. But this expression can be inverted, leading again to an expression of the observations as a linear combination of the noise, and hence a linear process. The difference from a moving average is that here the sum is infinite. Each observation is a linear combination of *all* previous noise samples, with exponentially decaying weights.

To derive this result, we start by defining B to be the "backward" operator. This means that $B(X_i) = X_{i-1}$. Next, define the "difference" operator as $1 - B$. This makes sense because

$$(1 - B)X_i = X_i - B(X_i) = X_i - X_{i-1}$$

In the simplest possible autoregressive process, $p = 1$. We then have $X_i = \phi X_{i-1} + \varepsilon_i$. By rearranging terms this is

$$X_i - \phi X_{i-1} = (1 - \phi B)X_i = \varepsilon_i$$

To express the observations X_i as a function of the noise ε_i, we need to apply an operator that is the inverse of $1 - \phi B$. Algebraically this is achieved by dividing by $1 - \phi B$:

$$X_i = \frac{1}{1 - \phi B}\, \varepsilon_i$$

Assuming $|\phi| < 1$, which is necessary for stationarity, we can consider this as the sum of an infinite geometrical series ($\sum_i q^i = 1/(1-q)$). This leads to a sum of multiple applications of the backward operator:

$$\begin{aligned} X_i &= \sum_{j=0}^{\infty} \phi^j B^j\, \varepsilon_i \\ &= \sum_{j=0}^{\infty} \phi^j\, \varepsilon_{i-j} \end{aligned}$$

For a general AR(p) process this can be done for each of the p terms and summed, leading to the same type of expression.

To read more: This box provides an extremely abbreviated description of a subject that justifies book-length treatment, and indeed has received many such treatments. For a relatively concise and readable introduction see Chatfield [121]. Another good one is the book by Box et al. [83].

End Box

Fractional Gaussian Noise

Fractional Gaussian noise (fGN) is a construction that leads to exact second-order self-similarity. It is based on the framework of autoregressive processes and random walks.

The definition of fractional Gaussian noise involves a number of steps. We start with Brownian motion. This is simply a random walk in which the individual steps are independent and Gaussian, that is, they come from a normal distribution. We typically assume a normal distribution with zero mean (so there is equal probability of moving to the left or to the right) and unit variance. We denote the location at time t by $B(t)$.

Note that $B(t)$ is a cumulative process: the location at time t is the sum of all the steps taken up to this time. The associated increment process is the steps themselves, which are Gaussian.

Now consider a slightly different type of process. Instead of summing all the steps, take the weighted average of all the steps, with weights that decline according to a power law. Specifically, the step at a lag of k in the past will receive the weight $k^{H-1/2}$. The range of H is $0 < H < 1$, and one can guess that it will turn out to be the Hurst parameter. This new process, which is actually a moving average of Gaussian noise, is called fractional Brownian motion with parameter H. Note that we could define it directly as an infinite moving average of independent Gaussian variables; the only reason to start with the Brownian motion is to justify the name.

Recall that the increment process of the original Brownian motion is Gaussian. Using this analogy, we will call the increment process of fractional Brownian motion by the name "fractional Gaussian noise". Importantly, these increments are correlated with each other, and, specifically, exhibit long-range dependence. A special case occurs when their autocovariance decays exactly as described by Equation (7.12). In this case the resulting process is exactly self-similar.

Fractional Brownian motion and fractional Gaussian noise are the simplest models for self-similarity and long-range dependence that allow for analytical treatment [378]. They have therefore been widely employed for various signal-processing applications. However, they have several drawbacks [567]. A major one is the symmetry of the underlying Gaussian increments, which may lead to negative values when the standard deviation is large relative to the mean. This is indeed a real problem as witnessed by the data in Figure 7.5, which is obviously asymmetrical. In addition, these models neither support large bursts nor short-range dependence. Therefore practitioners often prefer more mechanistic or constructive models, such as those described in the following sections.

To read more: The preceding description is an extreme simplification that ignores all the mathematical problems of this model. For a rigorous definition see Mandelbrot and van Ness [468] or Granger and Joyeux [299].

Fractional ARIMA

The desire to model short-range dependence has led to the definition of various autoregressive and moving average models as listed in the box on page 361. It turns out that a simple generalization, allowing one of the parameters to assume fractional values rather than being an integer, leads to models with long-range dependence. Specifically, fractional ARIMA processes (denoted fARIMA) are a generalization of ARIMA processes in which d is allowed to be fractional, and specifically $d < \frac{1}{2}$ [343, 344]. This leads to an expression where observations are again a linear combination of the noise, except that the weights decay according to a power law rather than exponentially. As a result there is a stronger correlation between distant observations, hence long-range dependence.

Let us now derive this more formally. First, we show that in an fARIMA process the coefficients of the linear combination decay polynomially. Second, we show that this leads to an autocovariance that also decays polynomially. This is the definition of long-range dependence (Section 7.3.4), which leads to asymptotic second-order self-similarity (Section 7.3.5).

Recall that d originally denotes the number of times the series is differenced. The first difference is obtained by subtracting the previous element. Identifying the previous element using the backward operator B, we can express differencing as $1 - B$ (see discussion on page 362). High-order differences are then obtained by applying this operator again and again, which algebraically means raising it to a power. For example, the second difference is $(1-B)^2$, and means that we apply the difference operator twice:

$$\begin{aligned}(1-B)^2X_i &= (1-B)[(1-B)X_i] \\ &= (1-B)[X_i - X_{i-1}] \\ &= (X_i - X_{i-1}) - (X_{i-1} - X_{i-2}) \\ &= X_i - 2X_{i-1} + X_{i-2}\end{aligned}$$

The square brackets in the first two lines denote the first difference. Applying the difference operator again to the series of first differences produces the third line, and collecting terms produces the fourth. Higher orders can be found in a similar manner; they turn out to be based on binomial expressions.

In an ARIMA$(0, d, 0)$ process, this is applied to a series X, and the result is the random noise ε:

$$(1-B)^d X_i = \varepsilon_i$$

But it can also be inverted, giving

$$X_i = (1-B)^{-d}\varepsilon_i$$

In other words, given a series of random noise samples ε_i we can apply the inverted operator and obtain a series of dependent X_is.

To understand this inverted operator, we use the Taylor expansion of the function $f(z) = (1-z)^{-d}$. The first derivative of this function is $f'(z) = d(1-z)^{-(d+1)}$. The second derivative is $f''(z) = d(d+1)(1-z)^{-(d+2)}$. In general, the nth derivative is $f^{(n)}(z) = d(d+1)\cdots(d+n-1)(1-z)^{-(d+n)}$. Evaluating this at $z = 0$ leaves only the initial factors, because the last one is found to equal 1. The Taylor expansion is therefore $(1-z)^{-d} = 1 + \sum_{j=1}^{\infty} \frac{1}{j!} d(d+1)\cdots(d+j-1)z^j$. Using this, we find that the desired inverted operator can be expressed as

$$(1-B)^{-d} = 1 + \sum_{j=1}^{\infty} \frac{d(d+1)\cdots(d+j-1)}{j!}\, B^j$$

This is in effect an infinite sum in which more distant terms (those that correspond to more applications of the backwards operator) have weights c_j that drop off according to a power law. To see this, note that

$$\begin{aligned}c_j &= \tfrac{1}{j!}\textstyle\prod_{k=1}^{j}(d+k-1) \\ &= \textstyle\prod_{k=1}^{j}\left(1 - \frac{1-d}{k}\right) \\ &= \exp\Big(\textstyle\sum_{k=1}^{j}\log\left(1 - \frac{1-d}{k}\right)\Big) \\ &\approx \exp\Big(-(1-d)\textstyle\sum_{k=1}^{j}\frac{1}{k}\Big) \\ &\approx \exp(-(1-d)\log j) \\ &= \frac{1}{j^{1-d}}\end{aligned}$$

where the first approximation is based on $\log(1-x) \approx -x$ for small x, which is the case for large k.

Note that if d is an integer, as in a normal ARIMA process, this implies that the weights in the sum *grow* polynomially (or are all unity if $d = 1$). As a result the sum is unbounded, and the process is ill defined. Therefore the expression makes sense only if $d < 1$. In other words, d is fractional. In fact, we need finite variance for the definition of self-similarity, implying the requirement $\sum_{j=0}^{\infty} c_j^2 < \infty$. This leads to the constraint that $2(1-d) > 1$, or $d < \frac{1}{2}$. And we can use the gamma function to interpolate the factorial for non-integer values (see page 126):

$$(1-B)^{-d} = \sum_{j=0}^{\infty} \frac{\Gamma(d+j)}{\Gamma(d)\Gamma(j+1)} B^j$$

Given that the coefficients of the linear combination of white noise samples decay polynomially, what does this say about the autocovariance of the resulting process? As we now show, it also decays polynomially, implying that this process exhibits long-range dependence.

Denote the elements of the fARIMA process by $X_t = \sum_{j=0}^{\infty} c_j \, \varepsilon_{t-j}$, that is, each is a linear combination of all preceding white noise terms with coefficients c_j (note that the index of X may in principle be negative, and that the indices of ε go back to $-\infty$). Its autocovariance is then

$$\gamma(t) = \mathbb{C}\mathrm{ov}(X_t, X_0) = \mathbb{C}\mathrm{ov}\left(\sum_{j=0}^{\infty} c_j \, \varepsilon_{t-j}, \sum_{j=0}^{\infty} c_j \, \varepsilon_{-j}\right) = \sigma^2 \sum_{j=0}^{\infty} c_{j+t} \, c_j$$

The last equality stems from the fact that when we multiply factors of the form $c_j \varepsilon_{t-j} \cdot c_\ell \, \varepsilon_{-\ell}$, the symmetric and random nature of the εs implies that they cancel out unless $t - j = \ell$. In that case the expected value of $\varepsilon_{t-j}\varepsilon_{-\ell}$ is the variance of the noise term, hence the factor σ^2.

We have shown that $c_j \approx \frac{1}{j^{1-d}}$. Therefore the expression for the autocovariance may be approximated by the integral

$$\gamma(t) \approx \sigma^2 \int_0^{\infty} \frac{1}{(x+t)^{1-d} x^{1-d}} \mathrm{d}x \longrightarrow \frac{1}{t^{1-2d}}$$

When $0 \le d < 0.5$ we have $1 - 2d \le 1$, so the sum of autocovariances is indeed infinite, implying long-range dependence.

The simplest type of fARIMA process is naturally fARIMA$(0, d, 0)$. It has long-range dependence due to the fractional differencing, but no autoregressive and moving average components. The more general fARIMA(p, d, q) models allow for modeling of both long-range and short-range dependence.

Implementing these models is conceptually straightforward. First, generate an infinite series of noise terms, with expectation 0. Then use them to generate the long-range dependent series by creating successive sums shifted by one with weights c_j. In practice this is not so simple. First, we need to decide on some threshold because, of course, we cannot create an infinite series. Regrettably, the threshold limits the range of the dependence, so it should not be too small a fraction of the number of samples required.

Second, the procedure as described earlier is very inefficient, as the sum has to be calculated anew for each shift. This implies quadratic time, meaning that the time needed to generate n samples will be proportional to n^2. But note that actually what we are doing is to convolve the series ε_i with the series c_j. This can be done in time $n \log n$ by performing Fourier transforms and multiplying in the frequency domain.

7.5.2 Multiscale Wavelet-Based Construction

As shown in Section 7.4.5, the discrete wavelet transform can be used to transform a time series (which, in the context of workload modeling, typically represents an arrival process) into a sequence of wavelet coefficients. This transformation does not result in the loss of any information. Consequently, the inverse transform can be used to turn the wavelet coefficients back into the original time series. But this procedure can also be used to generate synthetic arrival data. To do so, we simply generate suitable synthetic wavelet coefficients, and feed them into the inverse wavelet transform. The result will be a synthetic arrival process.

A specific construction based on this principle is the multifractal wavelet model of Riedi et al. [567]. This model is based on a pair of observations. First, we want to ensure that the resulting data is all positive. This is a requirement for workload models, where the data typically reflects the volume of arrival (such as the number of packets that arrive in a unit of time). Obviously, this cannot be a negative number. Second, when using the inverse Haar transform, at each step we generate a new (finer) approximation by adding and subtracting a detail $d_{j,k}$ from an approximation $a_{j,k}$. Therefore, to guarantee that everything remains positive, all we need to do is to ensure that $0 < d_{j,k} < a_{j,k}$ for all scales j and translations k. This is achieved by creating the details based on the approximations — specifically, multiplying the relevant approximation by a factor that is smaller than 1:

$$d_{j,k} = f_{j,k} \cdot a_{j,k} \qquad\qquad |f_{j,k}| \leq 1$$

Note that the factor $f_{j,k}$ may be negative, as long as its absolute value is not larger than 1. Having a negative factor just means that we first subtract and then add, rather than first adding and then subtracting. The distribution of $f_{j,k}$ should be symmetric around 0.

In addition, recall from Section 7.4.5 that when we perform a wavelet transform of self-similar data, the average squared details grow exponentially with the scale j. This is the basis for estimating the Hurst parameter H from the logscale diagram. Therefore the factors at each stage must come from an appropriate distribution. More specifically, the variance must change by a factor of $2^{(2H-1)}$ at each scale, because this was the slope of the line in the logscale diagram.

In summary, the procedure to generate a sequence of $N = 2^n$ positive values with an average of v is as follows. Note that it is more convenient to number the scales j in the opposite order than in Section 7.4.5; thus $j = 0$ here represents the coarsest scale, and $j = n$ represents the fine scale of the full generated dataset.

1. Set the global approximation $a_{0,0} = \sqrt{2^n}\, v$.

2. Set the initial variance to some number much smaller than 1. If the model is based on some data that was analyzed using a wavelet transform, use $\mathbb{V}\mathrm{ar}_0 = d_{0,0}^2/a_{0,0}^2$.
3. Perform n stages of doubling the length of the sequence, indexed from $j = 0$ to $j = n - 1$. Each stage consists of the following steps.
   (a) At each stage (except the first) increase the variance relative to the previous stage:
   $$\mathbb{V}\mathrm{ar}_j = 2^{(2H-1)}\,\mathbb{V}\mathrm{ar}_{j-1}$$
   (b) Select a distribution with this variance. For example, one can use the point distribution
   $$f = \begin{cases} c & \text{with probability } r \\ 0 & \text{with probability } 1-2r \\ -c & \text{with probability } r \end{cases}$$
   which has variance $\mathbb{V}\mathrm{ar}(f) = 2rc^2$. Setting $r = \frac{1}{3}$, for example, then leads to $c = \sqrt{\frac{3}{2}\mathbb{V}\mathrm{ar}_j}$. Note that this should satisfy $c \leq 1$.
   (c) Then, at stage j, use the 2^j values that were produced in the previous stage (here indexed by $k = 0$ to $2^j - 1$), and for each one do the following.
      i. Select a factor $f_{j,k}$ at random from the above distribution to guarantee the correct variance for this stage.
      ii. Model the respective detail as $d_{j,k} = f_{j,k} \cdot a_{j,k}$.
      iii. Compute the two new approximations
      $$\begin{aligned} a_{j+1,2k} &= \tfrac{1}{\sqrt{2}}(a_{j,k} + d_{j,k}) \\ a_{j+1,2k+1} &= \tfrac{1}{\sqrt{2}}(a_{j,k} - d_{j,k}) \end{aligned}$$
4. The final approximations $a_{n,0}$ to $a_{n,2^n-1}$ are the output sequence.

Note that in effect what this procedure does is to select 2^n factors $f_{j,k}$, and produce the outputs by multiplying the original value v by specific subsets of n factors with appropriate signs.

An example of the output of such a process is given in Figure 7.18. It starts with an average value of $v = 3$. The details are created based on factors taken from a point distribution with $r = 0.4$ and an initial variance of 0.0005. Such a low variance at the top level is required to allow for growth at finer levels without creating factors that are larger than 1. The variance grows by a factor of $\sqrt{2}$ at each stage, which corresponds to a Hurst parameter of $H = 0.75$.

7.5.3 Bias Models

The b-model can be viewed as a special case of the multifractal wavelet model, in which all the factors at all the levels are equal. The idea is to start with a certain amount of work, and then divide it recursively according to fixed proportions [720]. Thus, starting from the top level, we divide the total work W into two: $b\,W$ and $(1-b)\,W$, where $b < 1$

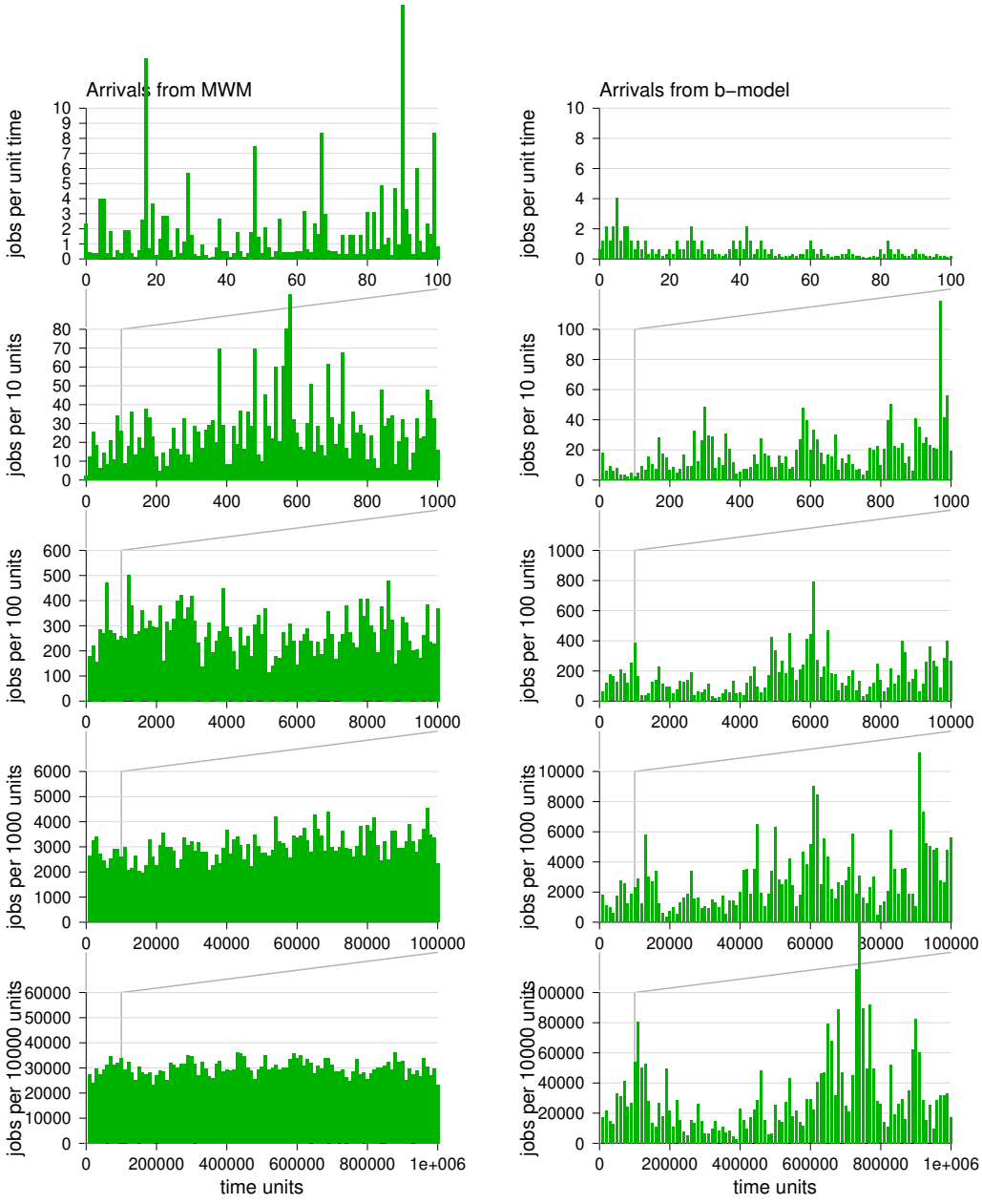


Figure 7.18: *Synthetic workloads generated by a multifractal wavelet model (left) and a b-model (right) shown at five different scales. In both cases the average workload is three jobs per time unit.*

is the bias. This continues for as many levels as we wish: given the 2^n points produced after n levels, we can partition each of them with a ratio of b to $1-b$ to produce the 2^{n+1} points of level $n+1$. Such recursive divisions inherently lead to a self-similar structure. However, at each division which part comes first can be decided randomly.

From this description we see an obvious relationship to a tree structure, in which each node splits the work among its two children with a bias of b. But maintaining such a tree is memory intensive and unnecessary. Instead we can use a stack that enables us to simulate a depth-first traversal of the tree, generating part of the output whenever we reach a leaf. The procedure is as follows.

1. Decide on the total volume of work W and the length of the workload 2^n.
2. Decide on the bias b or estimate it based on existing data that is being modeled (as explained below).
3. Place the pair $\langle W, 0\rangle$ on the stack.
4. As long as the stack is not empty, do the following.
   (a) Pop a pair $\langle work, level\rangle$ from the stack.
   (b) If $level = n$, print $work$ as the next output and return to step 4.
   (c) Otherwise, create two new pairs:

$$\langle b \cdot work,\ level+1\rangle \qquad \text{and} \qquad \langle (1-b)\cdot work,\ level+1\rangle$$

   Flip a coin to decide in which order to put them on the stack, and return to step 4.

An important question is how to set the parameter b. To do this we use entropy plots [720]. Assume that $W = 1$ and denote the ith value at the nth level by $Y_i^{(n)}$. By construction, the sum of all the values in the same level is W, namely 1. Therefore we may regard them as a probability distribution and calculate the entropy of this distribution.

In information theory, entropy measures the degree to which a distribution is uniform. The entropy of a distribution over N points ranges from 0 when the distribution is completely skewed (all the mass is concentrated on one point) to $\log N$ when it is uniform. The formula is

$$E = -\sum_{i=1}^{N} p_i \log p_i$$

where p_i is the probability of point i.

At the first level of the b-model construction there are only two points, with values of b and $1-b$, which we regard as the two probabilities. The entropy is therefore

$$E_b = -b\log b - (1-b)log(1-b)$$

We will show by induction that at level n the entropy is nE_b. The induction step is then simply that each level adds E_b to the entropy. The entropy at level $n+1$ is

$$E^{(n+1)} = -\sum_{i=0}^{2^{n+1}-1} Y_i^{(n+1)} \log Y_i^{(n+1)}$$

According to the construction, we can perform the summing in pairs:

$$E^{(n+1)} = -\sum_{i=0}^{2^n-1} \left(Y_{2i}^{(n+1)} \log Y_{2i}^{(n+1)} + Y_{2i+1}^{(n+1)} \log Y_{2i+1}^{(n+1)} \right)$$

But the elements in each pair are related, being derived from the same element in the previous level, so

$$E^{(n+1)} = -\sum_{i=0}^{2^n-1} \left(bY_i^{(n)} \log[bY_i^{(n)}] + (1-b)Y_i^{(n)} \log[(1-b)Y_i^{(n)}] \right)$$

By turning the log of a product into a sum and rearranging the terms we then get

$$\begin{aligned} E^{(n+1)} = &-\sum_{i=0}^{2^n-1} Y_i^{(n)} [b \log b + (1-b) \log(1-b)] \\ &-\sum_{i=0}^{2^n-1} [bY_i^{(n)} \log Y_i^{(n)} + (1-b)Y_i^{(n)} \log Y_i^{(n)}] \end{aligned}$$

Now in the first sum the $Y_i^{(n)}$ sum to 1, leaving E_b, and in the second sum $b+(1-b) = 1$ leaving $E^{(n)}$. This leads to the conclusion that $E^{(n+1)} = E_b + E^{(n)}$, and given that $E^{(1)} = E_b$ we find that indeed $E^{(n)} = nE_b$.

The importance of this derivation is the following. In the model, we present the tree in a top-down manner: we start with all the work at the root, and partition it at each level until we reach the leaves. But it is also possible to look at this in the other direction, namely bottom up. In this case we start with a workload sequence, and aggregate it until we reach the top. Given real data we can actually do this, and compute the entropy at each level of aggregation. Plotting the entropy as a function of the level should lead to a straight line, with a slope of E_b. From this we can extract the bias b to use as a parameter of the model.

An example of the output of using the b-model is given in the right-hand graph of Figure 7.18. The parameter used in this example is $b = 0.65$. Two observations stand out. First, even a relatively moderate parameter of 0.65 implies that half of the generated workload contains nearly double the work in the other half (a 65% to 35% split). Therefore the data seems to contain a growing trend and looks nonstationary. Second, because the same factor b is used at all levels, the variance is not reduced with aggregation. In particular, since the graph uses data from a predefined part of the generated trace for the visualization, at fine resolutions it appears to contain much less work than the average. This occurs when there happen to be many multiplications by $1-b$ leading to this part. It could also have much more work than the average, if there happen to be many multiplications by b.

An extension of the b-model is the PQRS model [720]. The goal of this model is to capture not only the burstiness and self-similarity that characterize the volume of the work, but also the locality properties that characterize the relationship between successive work items. In fact, the model is based on the observation that these two aspects of workload modeling are actually similar: just as burstiness implies that there are some times with a heavy load and others with a light load, locality implies that there are some attribute values that are very popular and others that seldom occur.

Moreover, the patterns of burstiness and locality may be correlated in various ways. This can be visualized by heat maps similar to those we used to motivate locality of sampling, where one axis represents time and the other represents some workload attribute (Figure 6.11). The PQRS model generates synthetic heat maps of this kind in a manner that is similar to the way the b-model generates a time series. The difference is that here a 2D structure is needed, so instead of using one parameter and dividing into two at each step, we need three parameters and divide into four.

In more detail, the model is based on the parameters P, Q, R, and S, such that $P + Q + R + S = 1$. A square heat map of size $2^n \times 2^n$ is created, meaning that the generated workload will span 2^n time units and the modeled workload attribute will have 2^n possible values (or bins). The construction is recursive. In the first step, the total work volume W is divided into four parts, and each is assigned to one quarter of the heat map. For example, $P \cdot W$ may be assigned to the top-left quadrant, $Q \cdot W$ to the top-right quadrant, $R \cdot W$ to the bottom left one, and the final $S \cdot W$ to bottom right. The volume in each quadrant is then partitioned among its subquadrants according to the same proportions, going on until the bottom-most 2×2 cells are reached. The ordering of the quadrants can (and probably should) be randomized at each partition.

7.5.4 The M/G/∞ Queueing Model

One generative model of load that leads to self-similarity is based on the M/G/∞ queue. In this queueing model arrivals are Poisson, that is, interarrival times are exponentially distributed (the "M" stands for Markovian). The service times, in contrast, have a general distribution (the "G"); in particular, we are interested in service times that have a heavy-tailed distribution. The number of servers is infinite, so clients do not have to wait, but rather receive service immediately upon arrival.

The connection to self-similarity is made by defining the time series $X(t)$ to be the number of active clients at time t. These are the clients that have arrived prior to t and are still active; in other words, the interval since their arrival is shorter than their service time. If service times are heavy-tailed, the active clients will include many clients that have arrived very recently, and a few that have arrived a long time ago. It can be visualized as shown in Figure 7.19. In this figure, each arrival is represented by a dot. The dot's horizontal location is the arrival time, and values along this dimension are uniformly distributed. The vertical location is the service time, and values concentrate at low values. $X(t)$ is the number of points in the shaded region.

The connection to workload modeling is done as follows. Consider each client as a generator of work. For example, each client can represent a TCP connection, in which

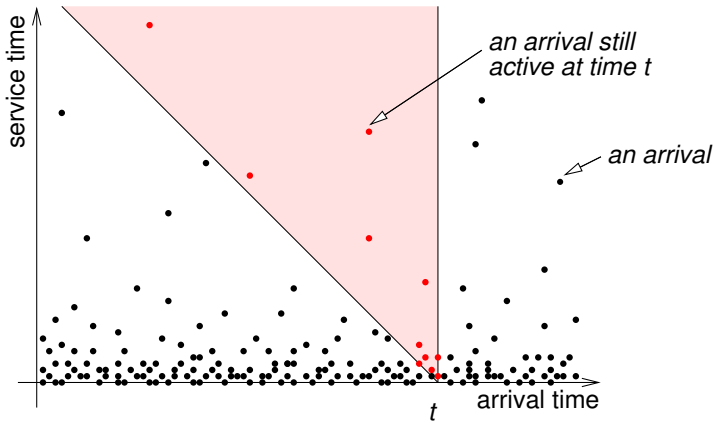


Figure 7.19: *Visualization of how an M/G/∞ queueing system is used to construct a self-similar time series.*

packets are transmitted continuously for the duration of the connection (which is the service time in the M/G/∞ model). Thus the total number of packets sent at time t is equal to the number of active clients at time t, that is, to $X(t)$.

The correlation between successive values of $X(t)$ is the result of clients that are shared between them. Significantly, the heavy-tailed distribution of service times means that some clients have very long service times. These then contribute to long-range dependence in $X(t)$. The power-law tail of the service times translates into a power-law decay of the autocorrelation of $X(t)$.

Implementing this model to generate the time series $X(t)$ is very easy. Based on the preceding description, it can be done as follows:

1. Start at $t = 0$ as the arrival time of the first client.
2. Select the client's service time d from a Pareto distribution.
3. Add 1 to all output values from $X(t)$ to $X(t + d)$. This represents the activity of this client.
4. Select an interarrival i from an exponential distribution.
5. Set $t = t + i$ as the arrival time of the next client, and return to step 2.

An example of the output produced by this method is given in Figure 7.20. The Poisson arrivals used $\lambda = 0.5$, that is, one arrival on average every two time units. The Pareto distribution had a minimal value $k = 1$, and a shape parameter $a = 1.2$. In the figure, we skip the first 10,000 time units to avoid the buildup period that starts from an empty system.

While the original model as described above is very synthetic, more realistic versions exhibit the same behavior. In particular, the Poisson arrivals can be replaced by another arrival process, in which interarrival times are independent but come from another distribution (that is, not the exponential distribution). In addition, the generation of work

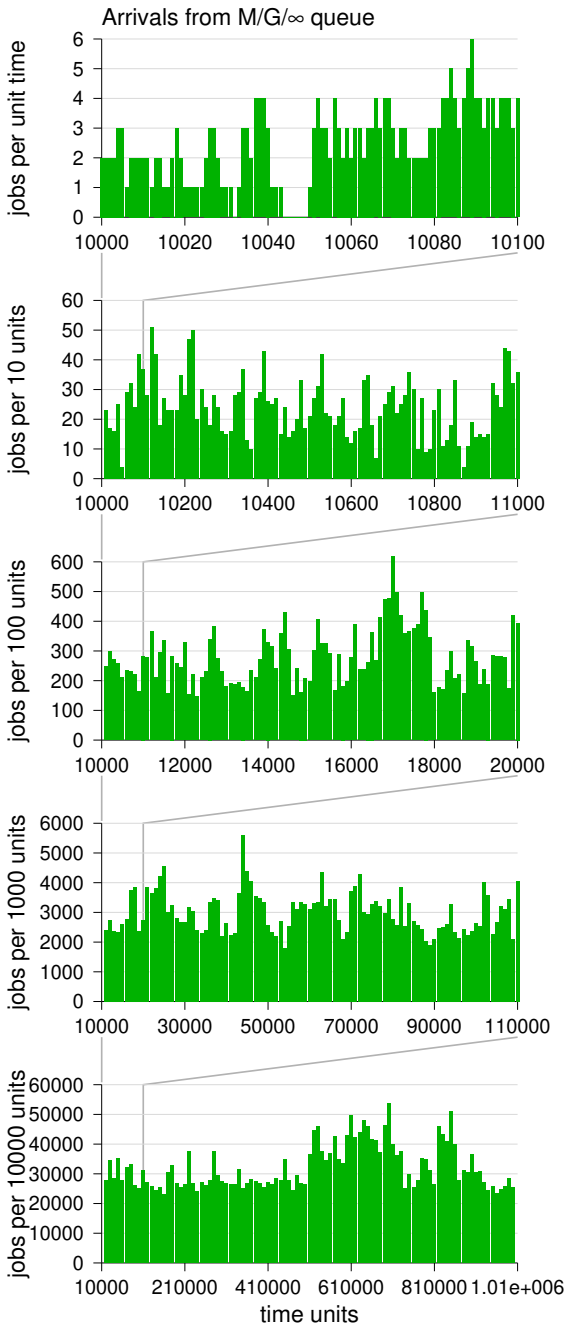


Figure 7.20: *Synthetic workload generated by an M/G/∞ model, shown at five different scales.*

during the session need not be uniform, but can be more general. The only essential element is the heavy-tailed distribution of session durations [731].

7.5.5 Merged On-Off Processes

The model of merged on-off processes is a generalization of the M/G/∞ model. That model can be interpreted as the merging of multiple processes, each of which produces a single connection with a heaty-tailed duration. The idea behind the merged on-off processes is very similar. Again the workload is generated by merging multiple processes. Again, each process has periods of activity that come from a heavy-tailed distribution. The difference is that each process alternates between *on* and *off* periods, rather than contributing a single *on* period [732, 155, 679, 304].

The crux of the model concerns the distributions governing the lengths of the *on* and *off* periods. If these distributions are heavy-tailed, we get long-range dependence: if a unit of work arrives at time t, similar units of work will continue to arrive for the duration d of the *on* period to which it belongs, leading to a correlation with subsequent times up to $t + d$. Because this duration is heavy-tailed, the correlation created by this burst will typically be for a short d, but occasionally a long *on* period will lead to a correlation over a long span of time. As many different bursts may be active at time t, what we actually get is a combination of such correlations for durations that correspond to the distribution of the *on* periods. But this is heavy-tailed, so we get a correlation that decays polynomially — a long-range dependence.

In some cases this type of behavior is built in, and is a direct result of the heavy-tailed nature of certain workload parameters. For example, given that web server file sizes are heavy-tailed, the distribution of service times will also be heavy-tailed (because the time to serve a file is proportional to its size). During the time a file is served, data is transmitted at a more or less constant rate. This rate is correlated with later transmittals according to the heavy-tailed distribution of sizes and transmission times, leading to long-range correlation and self-similarity [155].

This model is nice for several reasons. First, it has very few parameters. In fact, the self-similarity depends on a single parameter, α: the tail index of the distribution of *on* (or *off*) times. This should be chosen from the range $1 < \alpha < 2$, and leads to self-similarity with $H = \frac{3-\alpha}{2}$ [732, 679]. Moreover, the minimal α dominates the process, so different sources may actually have different values of α, and some may even have finite variance (that is, their *on* and *off* times are not heavy-tailed). Additional parameters of the model also have an intuitive physical meaning. For example, the number of merged on-off processes should correspond to the expected number of active users in the modeled system.

Like the previous model, implementing this one to generate a time series $X(t)$ is easy [679]. It can be done as follows.

1. Start at $t = 0$ as the beginning of the first *on* period.
2. Select the duration of the *on* period d from a Pareto distribution.

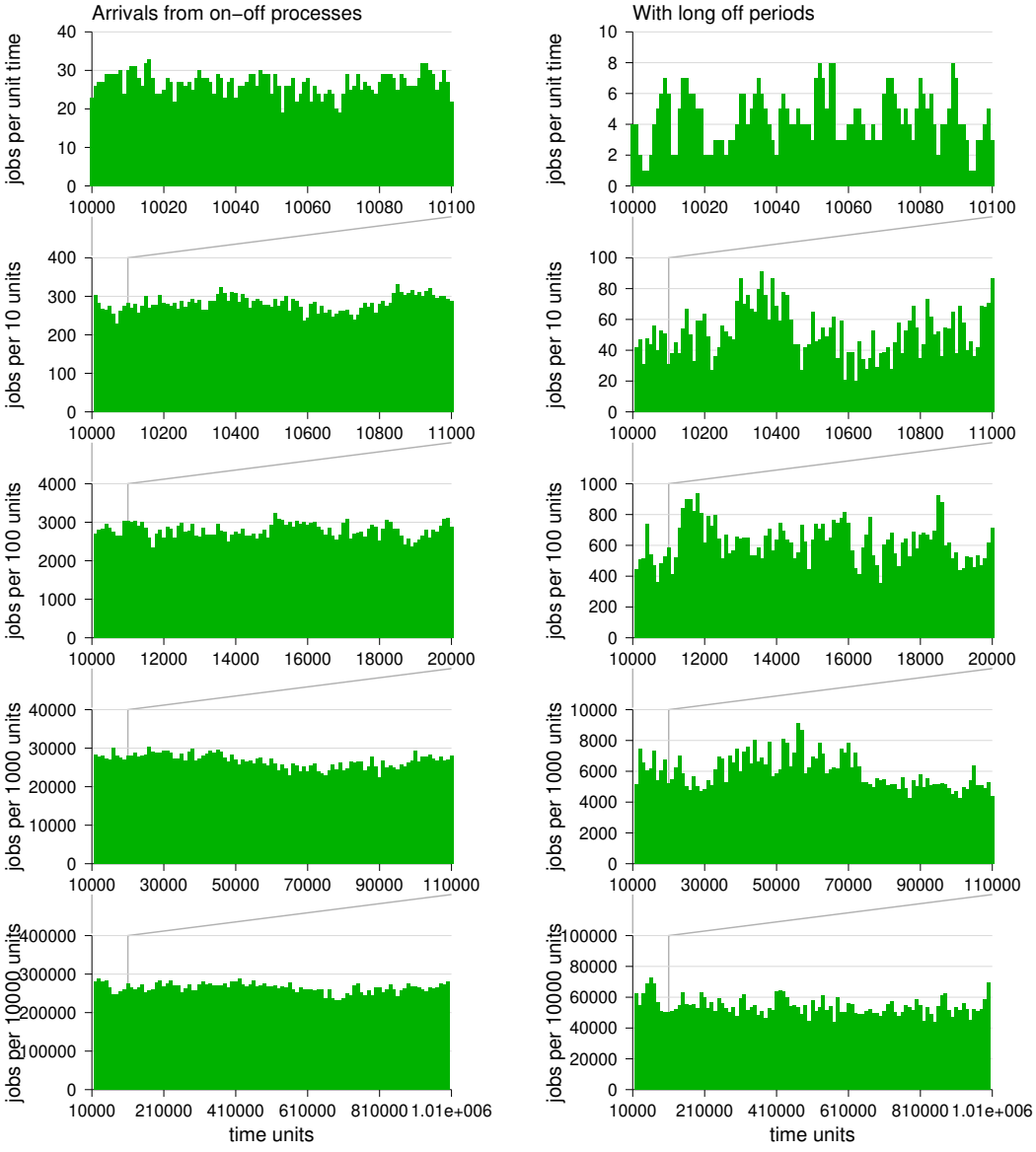


Figure 7.21: *Synthetic workload generated by merged on-off processes, shown at five different scales.*

3. Add 1 to all output values from $X(t)$ to $X(t+d)$. This represents the work generated during this *on* period.

4. Select the duration of the *off* period g from a Pareto distribution.

5. Set $t = t + d + g$ as the beginning of the next *on* period for this on-off process, and return to step 2. Continue to do so until the desired total length is obtained.

6. Return to step 1, and repeat the whole sequence of steps for another on-off process. Continue to do so until the desired number of processes have been modeled.

An example of the output produced by this method is given in Figure 7.21 (left), which used Pareto distributions with a shape parameter $a = 1.2$ for both the *on* and *off* periods. The number of processes being merged was 50. Again, the first 10,000 time units were discarded in the displayed figure.

When compared with Figure 7.20, the results of using merged on-off processes are less bursty. Indeed, using this procedure often leads to H that is smaller than expected according to the tail index α of the distributions being used [4]. The reason is that in an on-off process with heavy-tailed *on* and *off* times, the vast majority of *on* and *off* times are short. Therefore the average number of active users seen in any time unit is close to half of them — in our case, 25 of 50. This number is too low, because the theorem stating that heavy-tailed *on* and *off* periods lead to long-range dependence actually holds only asymptotically, when both the number of processes and the total duration observed tend to infinity [679]. Observing this directly may require periods of many hours, more than is realistic for network traffic which can be assumed to be stationary for only a couple of hours, and exhibits daily effects on longer scales [4].

However, there are several more practical ways to increase the variability relative to the mean, and thereby create more burstiness. One is to use longer off periods. This is shown on the right of Figure 7.21, inwhich 250 on-off processes were multiplexed, but the *off* periods from the Pareto distribution were multiplied by a factor of 50. A similar effect is obtained if only very few processes are used (e.g. five). Another option is to use a more diverse model in which different processes produce work at different rates during their *on* periods (that is, their behavior in step 3 is different). In fact, variable rates of activity in different *on* periods are also required to better model the realities of IP flows [753].

The most important parameter of an on-off processes model is the number of individual processes being multiplexed. According to Horn et al., at least 100 processes should be used to obtain a good approximation of the desired values of H [340]. However, this conflicts with data showing that bursts of activity do not result from the confluence of many small flows, but rather from the arrival of a few high-rate flows [592].

A variant of on-off models is to use a hierarchical model (more on such models in Chapter 8). For example, the LiTGen traffic generator models traffic in a hierarchical generative manner, by modeling user sessions, the web pages downloaded in each session, the objects that compose each web page, and the packets needed to retrieve each object [576, 575]. The distributions for the submodels are empirical distributions extracted from a trace, which typically have long tails, but the only ones that are really important to model correctly are the number of objects in a page and the packet interarrival times. At the bottom level, the packet arrival rate is made proportional to the object size: for large objects, the packet interarrival times are shorter. This feature is found to be crucial in reproducing long-range dependence similar to that observed in the original trace.

7.6 More Complex Scaling Behavior

Self-similarity is described by a single parameter: the Hurst parameter H. This parameter governs many different characteristics of the self-similar process. All finite dimensional distributions of the process scale according to H. As a result, all moments scale according to H. The autocorrelation also decays according to H. And all this is true at all times t. Both the measurement of self-similarity and its modeling depend on this fact.

But real processes may be more complex, and as a result may need more than a single parameter [567, 3]. One particular type of complexity that has been identified in the context of computer networking on WANs is that the scaling phenomena may be different at different scales. Thus the traffic is indeed self-similar at long time scales (in this context, long is more than a second), but not at short time scales (of no more than hundreds of milliseconds) [252, 251].

The identification and analysis of such effects are done using wavelets [6, 567, 663]. Wavelet analysis (as described in Section 7.4.5) is reminiscent of spectral analysis in the sense that the signal being studied is expressed in terms of a set of basis functions, or, rather, in terms of the coefficients of the different basis functions. In spectral analysis these basis functions are the sine and cosine, and the coefficients therefore identify the frequencies that make up the signal. Wavelets are also basis functions that have a limited band of frequencies, but in addition, they have a defined duration. Thus it is possible to express the fact that different frequencies apply at different times.

The resulting models are called multifractals. The essence of multifractals is that they have different scaling exponents at different times, as opposed to self-similar (also called monofractal) processes that have a constant scaling exponent H. The scaling exponent describes the derivative of the process at a given time, and hence its burstiness. The model is that

$$\lim_{\delta t \to 0} X(t + \delta t) - X(t) = \delta t^{H(t)}$$

where $H(t)$ is the local scaling exponent at time t. As $\delta t \to 0$, $H(t) > 1$ corresponds to low variability, whereas $H(t) < 1$ corresponds to high burstiness.

Although such models may provide a better fit to some data, the question of whether they are indeed justified and useful is still open [678, 6, 707].

8

Hierarchical Generative Models

The discovery of self-similarity, and the quest to model it, required a radical departure from previous practices. Before this discovery, workload items were considered to be independent of each other. Modeling could then be done by sampling workload items from the relevant distributions, subject to desired correlations between workload attributes. But self-similar workloads have an internal structure. Workload items are no longer distributed uniformly along time — they come in bursts, in many different time scales.

The next step is to consider whether it is only the arrivals that are bursty. Self-similarity says that workload items come in bursts, but says nothing about the nature of the items in a burst. Do the attributes of the burst items have any special characteristics? For example, do job sizes also come in "bursts", or are they just randomly selected from a distribution? And what about runtimes, memory usage, and so on?

The answer to these questions is that all workload attributes tend to come in bursts. It is common to see a burst of jobs with similar attributes, and then a burst of jobs with other attributes, and so on [239, 109]. Each burst is characterized by rather modal distributions, often concentrated around a single value. The wider distributions describing the entire workload are actually a combination of the modal distributions of the different bursts of activity. Thus the collective distribution does not capture the workload dynamics unless we postulate *locality of sampling*: instead of sampling from the whole distribution, sample first from one part, then from another, and so on.

A good way to produce locality of sampling is by using a hierarchical model: first select the part of the distribution on which to focus, and then sample from this selected region. This chapter suggests that such a model can and should be generative. This means that it should mimic and model the processes that create the real workload. Such models are therefore called *hierarchical generative models* (HGM).

The main example we focus on throughout this chapter is user-based modeling. This is based on the observation that each user tends to repeat the same type of work over and over again. When one user is very active, the workload therefore assumes the characteristics of that user's jobs. When another user becomes active, the characteristics of

the workload change. Thus by modeling the activity patterns of users we can create workloads with the desired locality of sampling.

Note that the same idea may apply to various other levels and types of workloads, even if human users are irrelevant. For example, packets may come from flows that are part of a connection, all of which occur at much finer time scales than explicit user activity. The important unifying theme is that the model not only imparts structure on the arrival process, but also leads to locality of sampling in all workload features.

8.1 Locality of Sampling and Users

Locality of sampling refers to the fact that workloads often display an internal structure: successive samples are not independent of each other, but rather tend to be similar to each other. This was demonstrated and quantified in Section 6.3. Here we show that this effect is associated with user activity.

The most extreme type of user activity that leads to locality of sampling is a workload flurry. The essence of this phenomenon is a surge of activity, typically by a single user, that completely dominates the workload for a relatively short period of time [249].

Figure 8.1 shows examples of large-scale flurries, in which the level of activity is up to 10 times the average. It also shows that the characteristics of such flurries are typically quite different from those of the workload as a whole. Specifically, flurries are usually composed of numerous repetitions of the same job, or of very similar jobs. Thus they do not span the space of attribute combinations of the entire workload, but rather concentrate at a certain point or at a small number of points.

Figure 8.2 shows that this sort of behavior is not unique to such large-scale flurries. Actually, many users tend to repeat the same type of jobs over and over again, and they do so over limited periods of time. The overall workload distributions are a result of combining the activities of many different users, each of which has a different characteristic behavior.

Additional support comes from the data displayed in Figure 8.3. This shows that the diversity in the workload grows with the duration of the observation window. More importantly, the number of different values observed in the logs is lower than would be observed if jobs were ordered randomly. This implies that there is a strong locality, and that only a small subset of the possible values are observed during short time windows.

Note that the number of active users also grows with the observation window. This can be interpreted as a causal relationship: longer windows provide an opportunity for more users to be active, and this in turn leads to more diverse workloads. However, this is merely circumstantial evidence. More direct evidence may be provided by creating a scatterplot showing the actual number of users in each week and the actual corresponding workload diversity (represented by the number of different job sizes). The results of doing so are somewhat mixed (Figure 8.4). On four of the checked systems, there indeed seems to be a positive correlation between users and workload diversity. On the other two this is not the case.

The thesis of this chapter is that the basic idea of locality of sampling as a result of

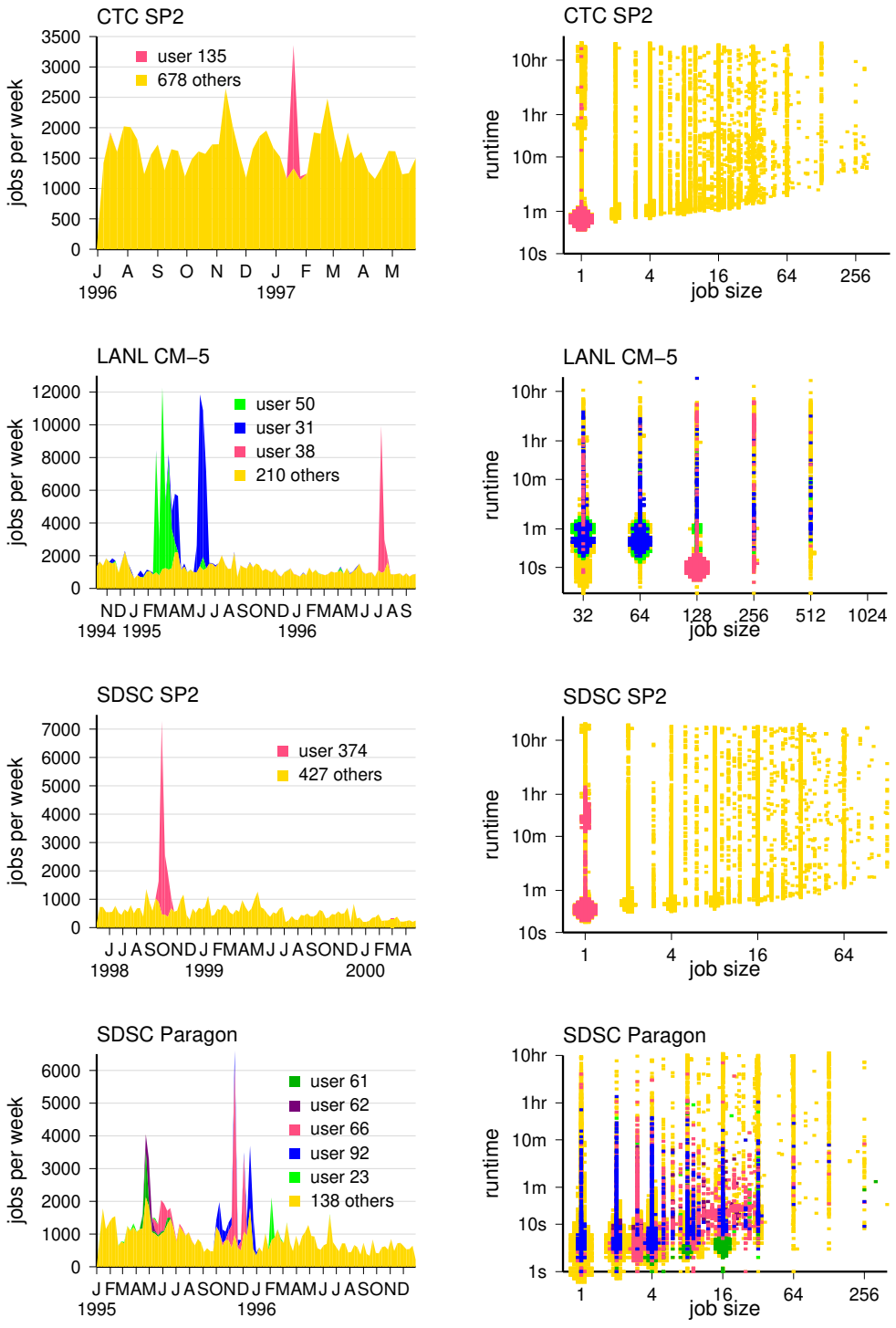


Figure 8.1: *The flurries of activity caused by single users are typically not distributed in the same way as the entire workload in terms of resource usage.*

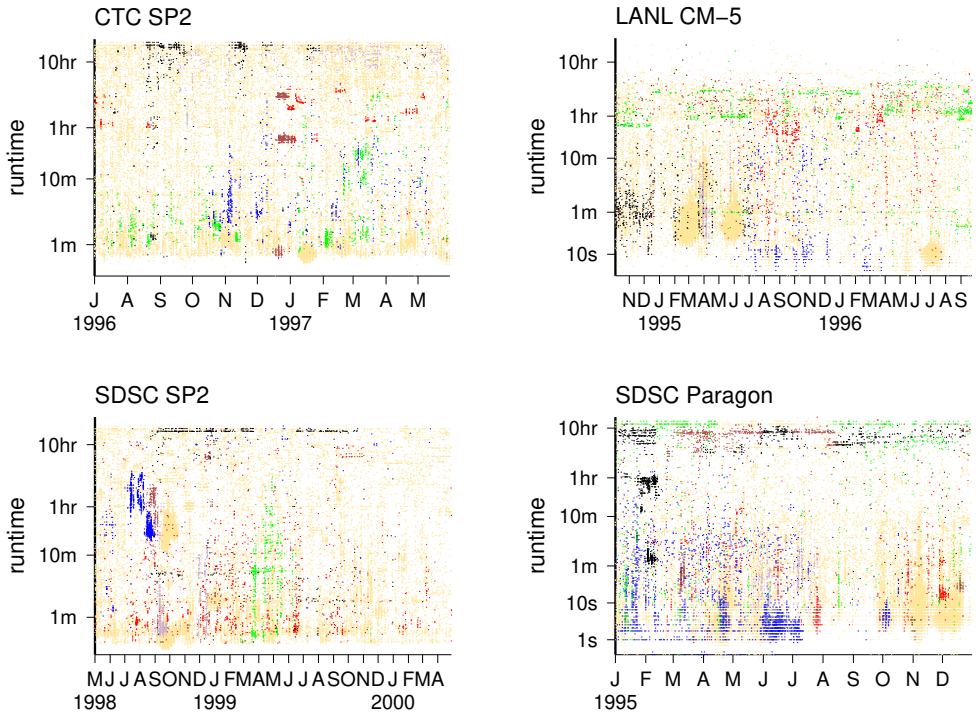


Figure 8.2: *The distributions representing specific highly active users are different from the overall distribution, both in the values sampled and in the distribution over time. In each log, six active non-flurry users are color-coded.*

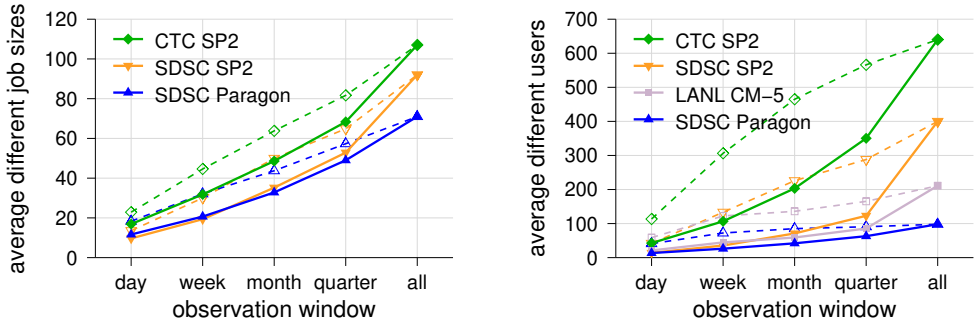


Figure 8.3: *Workload diversity as a function of the observation interval. With longer intervals, more values are observed, and more users are active. Dashed lines show the same data when the log is scrambled and locality is destroyed. In both cases, weekends are excluded for the "day" data point.*

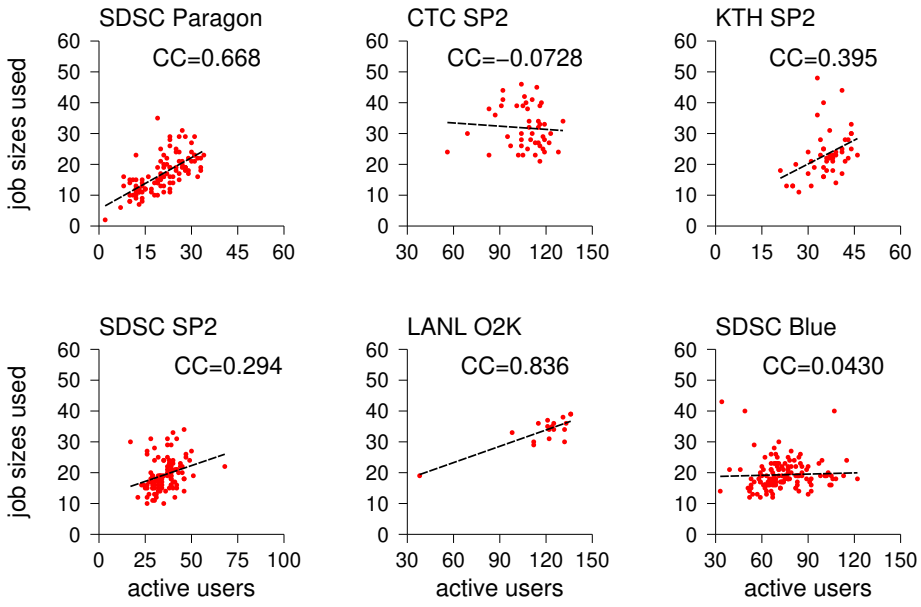


Figure 8.4: *Correlation (or lack thereof) between the number of active users and the workload diversity. Each data point corresponds to a single week from the log. The line segments are the result of applying linear regression.*

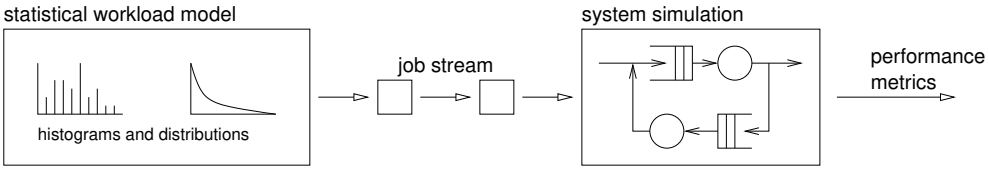


Figure 8.5: *Simple statistical workload modeling.*

localized user activity can be used in modeling. If we combine a model of the intermittent activity patterns displayed by users with models of the repetitive nature of the work they perform, we will get a workload that has locality of sampling. To accomplish this, we need a hierarchical workload model.

8.2 Hierarchical Workload Models

Straightforward statistical modeling of workloads involves sampling from selected distributions (Figure 8.5), which leads to the generation of a sequence of jobs that are independent of each other. These jobs are then fed to a system simulation, enabling the measurement of performance metrics. Alternatively, the distributions themselves may be used as input to mathematical analysis.

The problem with this approach is that the generated jobs are indeed independent

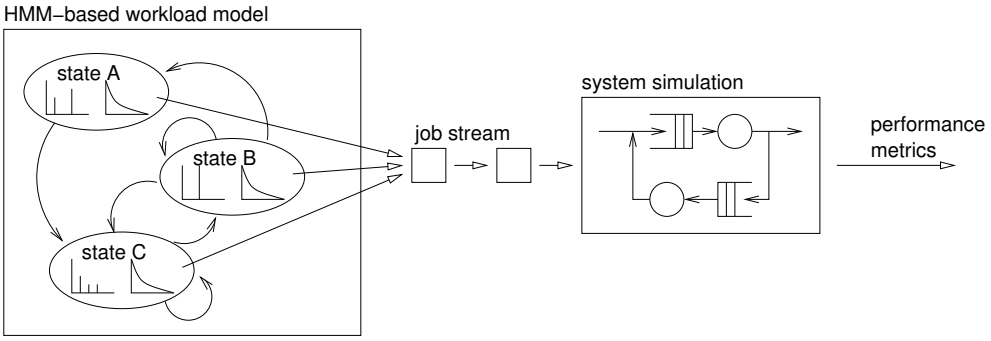


Figure 8.6: *Workload modeling based on a hidden Markov model.*

of each other. The generated workload therefore does not have any temporal structure, and, in particular, cannot display locality of sampling. To create an internal structure, we need a hierarchical model with at least two levels. The top level models the movement between different parts of the distribution, whereas the bottom level does the actual sampling. This leads to sequences of similar samples. The lengths of the sequences result from the details of the top-level model.

As in other cases, there are two ways to go about creating a hierarchical model. One is phenomenological: find the simplest abstract model that produces the desired behavior. The other is generative: try to model the process that actually creates the observed behavior in reality [369].

8.2.1 Hidden Markov Models

A specific type of hierarchical (or, rather, two-level) model that is often used is the *hidden Markov model* (HMM) [557]. This includes two parts. The first is a Markov chain, which provides the dynamics (i.e., how things change with time; for background on Markov chains, see the box on page 242). The other part is a set of distributions for each state in the Markov chain, that generate the outputs (in our case, the jobs or other workload items). This structure is depicted in Figure 8.6.

To generate a synthetic workload with such a model we do the following. The basis is a Markov process that visits the states of the hidden Markov chain. When a specific state is visited, a certain number of workload items are generated. The number of items and their characteristics (i.e., the distributions from which their attributes are selected) are specific to this state. After generating these workload items, we move to another state according to the transition probabilities of the Markov process.

The locality of sampling arises due to the fact that each state has different distributions of workload attribute values. Thus workload items generated when visiting one state will tend to be different from workload items generated when visiting another state. The overall distribution of workload attribute values is the product of the limit probability of being in each state and the output from that state.

A potential deficiency of an HMM-based model is that it may be too pure: at any

given time, only one state is active. This is similar to assuming that at any given time only one user is active. In reality, there is a set of active users, and the generated workload is an interleaving of their individual workloads. This interleaving does not lend itself to easy modeling with an HMM.

Note that HMMs are not only useful for capturing locality of sampling. They may also be used to model workloads that have an inherent structure [559]. For example, consider file system activity. Using files typically involves a mixture of open, close, read, write, and seek operations. But for any given file, these operations are typically ordered according to the following rules:

- The first operation is an open.
- The last operation is a close.
- Read and write operations are seldom mixed.
- Seek operations may come between other operations, but typically do not follow each other.

Selecting operations at random, even with the right probabilities, will lead to a nonsense workload that violates many of these rules. But with an HMM we can easily generate a stream of requests that satisfies them.

8.2.2 Motivation for User-Based Models

Real-world workloads incorporate myriad characteristics that may all have an effect on evaluation results. Modeling each of these characteristics individually is an arduous task. Using a generative model based on user activity is a natural approach that helps solve this problem.

The following subsections illustrate some of these characteristics, and how they are naturally modeled in a user-based model.

Locality of Sampling

Locality of sampling served as the motivating example leading to the use of HGMs (and user-based models in particular) in Section 8.1. Locality of sampling is built right into these models, because the workload is constructed by interleaving the activities of several simulated users, each of which displays low diversity.

Load Manipulation

It is often desirable to evaluate the performance of a system under different load conditions, e.g., to check its stability or the maximal load it can handle before saturating. Thus the workload model should not only model a single load condition, but should also contain a tunable parameter that allows for the generation of different load conditions.

(Note that we are talking about the average load, not the instantaneous load. Systems may always become overloaded for certain periods due to fluctuations in the load [600, 250].)

The most common approach to changing the load is to systematically modify the interarrival times or the runtimes. This is based on queueing theory, where the load (or utilization) is found to be the ratio λ/μ (i.e., the arrival rate divided by the service rate). For example, if a model generates a load of 70% of system capacity by default, multiplying all arrival times by a factor of $7/8 = 0.875$ will increase the load to 80%, while retaining all other features of the workload model. This is a workable solution for a stationary workload. However, if daily cycles are included in the model, such modifications influence the relative lengths of the jobs and the daily cycle, which is obviously undesirable (as discussed in Section 9.6.5).

In user-based modeling it is possible to achieve a modification of the load without changing any of the basic workload features. Instead, the load is manipulated by a high-level modification of the user population: adding more users increases the load, and removing users reduces the load [57, 108, 751]. Alternatively, load can be modified by changing the characteristics of the active users. For example, we may replace light users with heavier users.

Feedback and Throttling

Even when we do not manipulate the load explicitly, different load conditions may occur. In fact, fluctuations even occur within the same simulation. In real life, such fluctuations affect user behavior: when the system becomes overloaded, users tend to reduce their activity and back off.

Such effects are very important for reliable performance evaluations. To model them one needs a closed system model, where system performance feeds back into the arrival process. It is especially natural to do this in the context of user-based models, based on an explicit model of how users react to poor system performance. Such a model can include behaviors such as canceling jobs and aborting entire sessions.

Self-Similarity

Self-similarity has been found in many different workload types, most notably in Internet and LAN traffic [436, 540]. Similar results have been obtained for the workloads on file systems [304], parallel machines [670, 751], and more. Although self-similarity is basically a statistical property, it is hard to reproduce by a simple sampling from a distribution. Alternative models therefore construct the workload based on an on-off process: on and off periods are sampled from a heavy-tailed distribution, and then the on periods are filled with activity (Sections 7.5.4 and 7.5.5) [732]. In a sense, user-based modeling is an extension of this practice, which makes it more explicit. Importantly, at least in parallel job workloads it has been shown that self-similarity is indeed captured by the combined activity of multiple users, even if they are mixed [751].

Daily Cycles

Real workloads are obviously nonstationary: they have daily, weekly, and even yearly cycles [321]. In many cases this attribute is ignored in the modeling process, with the

justification that only the high load at prime time is of interest. This is reasonable in the context of network communication, in which the individual workload items (packets) are very small, but it is very dubious in the context of parallel jobs, which may run for many hours. Such long jobs are of the same order of magnitude as the daily cycle, and therefore necessarily interact with it. For example, it is possible to envision schedulers that delay long jobs to the less loaded night hours, while using the prime time resources for shorter interactive jobs [449, 248]. Such behavior cannot be evaluated if the workload does not include a faithful modeling of the daily cycle.

In a user-based generative model, the daily and weekly cycles are created by user sessions that are synchronized with these cycles. For example, Shmueli and Feitelson propose a model in which users have a propensity to leave the system around 6 PM and return around 8 AM the next day [616, 248]. Zilber et al. propose a model with four user classes, corresponding to combinations of users who are active at day or at night, and on weekdays or on weekends [763]. These and other approaches for modeling daily cycles were reviewed in Section 6.5.1.

Extrapolation

A recurring question about workloads is how to handle situations for which we do not have data. Using extrapolation with descriptive statistical models is dangerous, because we do not really know whether the workload will continue to act in a consistent manner beyond the range we have observed. But if we understand the mechanisms underlying the process that creates the workload, we can apply the same mechanisms in the new setting, and see what workload is produced. In this way the workload will naturally adapt to the new situation. Importantly, this can include a measure of self-regulation as is the case when feedback is included.

Generative models are especially useful if several changes are made at the same time. Consider an example where the scale of the system under study is larger than the systems from which we have data, and we also want to analyze the behavior of this system under an increased load. A parameterized generative model in which user behavior can be adjusted to reflect the system's scale can cope with such a scenario. But it would be hard to come up with appropriate modifications to a descriptive model.

Abnormal Activity

Finally, the workload on many types of systems sometimes displays abnormal activity. For example, in telephone systems Mother's Day is notorious for causing overload and congestion. On the web, events ranging from Live-Aid rock concerts to Victoria's Secret fashion shows have been known to bring servers and routers down. A targeted attack is also a unique form of activity that is typically orders of magnitude more intensive than normal usage. All of these are similar to workload flurries — large outbreaks of activity that can each be attributed to the activity of a single user (Sections 2.3.3 and 2.3.4).

by mixing normal and abnormal users, user-based modeling enables such singular behavior to be modeled separately from the modeling of normal activity [751]. This

leads to more representative models of normal behavior, as well as the option to control the characteristics of abnormal behavior that are desired for an evaluation.

Another special type of user is a *robot* — a software agent that creates workload automatically. Examples include the following:

- Scripts that are activated every day at the same time in order to perform cleanup operations (an example was shown in Figure 2.24).
- Programs that are activated at regular intervals to perform some monitoring task.
- Web agents that search for information or perform indexing operations.

With a user-based model, mixing in a controlled amount of robot activity is quite straightforward. All it requires is the definition of users that behave like robots.

Of course, this discussion above is not limited to long-range workloads based on human users. It is easily generalized to other situations as well. The only requirement is that the overall workload be viewed as the interleaving of multiple independent workload streams. This is a natural situation for environments such as networks and servers. Interestingly, it may also be relevant for other cases, that initially seem ill suited for this approach. For example, memory accesses are typically thought of in terms of a single continuous stream of requests. But in reality, they are an interleaving of processes at three levels. At the microarchitecture level, memory accesses are generated by multiple concurrent instructions that are executed out of order. At the application level there may be multiple threads that all generate independent streams of instruction accesses and data accesses. At the system level we have an interleaving between multiple applications and the operating system. And with multiprocessor and multicore machines, the various application and system threads really operate in parallel.

8.2.3 The Three-Level User-Based Model

The premise behind the three-level user-based model is that the best way to capture the internal structure of a workload is to model the process that creates the workload. If the workload is created by human users, we should create a model of the dynamics of users using the system. The workload is then generated by running this model, in effect simulating the process that generates the workload.

The basic building block of the model is the user. The idea of using "user equivalents" as the basic unit of load generation has appeared in the literature (e.g. in [57]) and is even at the basis of the TPC-C benchmark. It is used here to encapsulate a sequence of similar workload items that are then interleaved with those generated by other users.

The model of how user interaction with the system is generated divides this process into three levels:

1. *The user population*. The workload is not generated by a single user, but by an entire population of different users who are active at the same time. Moreover, the population changes with time, as some users stop using the system and others come in their place; this shift in the user population causes the workload to change

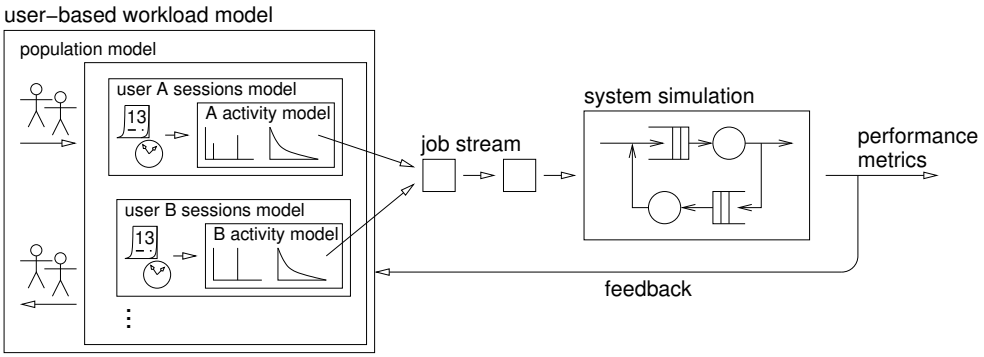


Figure 8.7: *User-based workload modeling.*

and leads to locality of sampling. The generated workload is the interleaving of the activity of all these users.

2. *User sessions*. Each user has a certain residence time in the system. During this residence time he or she engages in one or more sessions. The residence time of the user creates a long-range correlation in the produced workload, leading to self-similarity. Synchronization among the sessions of different users leads to the daily cycle.
3. *Activity within sessions*. During each session, a user generates a sequence of jobs. These jobs are typically similar to each other, leading to locality of sampling. And when the interaction between the user and the system is tight, a feedback loop may be formed whereby system performance affects the timing of subsequent submittals.

It is also possible to continue the modeling with additional, lower levels [336]. For example, an application may generate a sequence of requests from different servers, and each such request may be transmitted as a sequence of packets. But these lower levels are only relevant at a more detailed level of modeling.

This model is hierarchical because each level branches out to multiple instances of the next lower level. Moreover, these instances need not all be clones of the same model. This inclusion relationship is shown in Figure 8.7. At the top level there is only one population model. At the next level there are multiple user session models, one for each user; these may be different from each other, reflecting different user classes. At the bottom level are the session activity models, one for each session of each user. They can be the same for all the sessions of a given user, but still they may be parameterized because the activity within a session may depend on the day and time, for example.

An intriguing possibility made available by user-based modeling is to incorporate feedback: we can create a "smart" model in which users are not oblivious, but rather create additional work in a way that reflects satisfaction with previous results (assuming we can get a handle on the factors leading to satisfaction [181]). In fact, feedback can affect all three models: at the population level, dissatisfied users may elect to leave and cease using the system; at the session level, a session may be truncated short when the

system is not responsive, or else it may be extended if the work flows well; and at the activity level, jobs may be delayed until previous ones terminate and their results are examined. The notion of feedback and how it affects workload generation is discussed at length in Section 8.4.

8.2.4 Other Hierarchical Models

The three-level user-based model of interactive work by multiple users on the same computer system (e.g., a server or supercomputer) is but one of many hierarchical generative models. Many other such models have been proposed and used.

User Typing

An early example of a hierarchical model involved modeling the activity of a user seated at a terminal and working with a shared computer system [272]. The user would type at the terminal and the computer would respond. This was called "computer communication", referring to the communication between the user and the computer, rather than the communication among computers as we use the term today.

The bottom level of this model consisted of typing or displaying a single letter. Above it were bursts of letters that were typed or displayed in sequence. At the top level was a session composed of alternating bursts of characters by the user and the computer. The time from the end of a computer burst to the beginning of the following user burst is the think time.

Note that modeling user typing is not obsolete. Each key typed generates an interrupt on a desktop machine, and the sending of a packet in a networked interactive session. Indeed, the model used by Danzig et al. for telnet is very similar to the model outlined above [164].

World Wide Web Activity

World wide web access was the dominant form of Internet traffic from about 1995 to 2005. This load is generated by humans sitting in front of computers and clicking on links (today, a growing part of this load is generated by computer programs, which we ignore). Web activity may be analyzed in terms of a three-layer model similar to the one proposed earlier: the aggregate traffic, the sequences of sessions from individual clients, and the requests within each session [525]. But it is also possible to dissect the data in more detail, leading to some variant of the following multilevel model [36, 336, 446, 536, 576]:

1. The top level is the user session, starting when a user sits down and ending when he or she leaves.
2. Each session is composed of conversations with different Internet hosts, and reflects browsing patterns in which users first retrieve web pages from one server and then surf on to another server. To model this, one needs to model the number of conversations per session, their arrivals, and the distribution of their destinations.

3. Each conversation may involve multiple navigation bursts with the same host. The motivation for this level is that people usually retrieve several pages in rapid succession until they find the one with the information they really want to read. One then needs to model the size of the burst and the intervals between the downloaded documents.
4. Each downloaded document may have embedded graphics in it that have to be retrieved separately. This is modeled by the number of embedded objects and their sizes in transferred bytes.
5. At the lowest level, the retrieval of each object may require multiple packets to be sent. The arrival pattern of these packets is derived from the higher levels of the model, as is their number.

This description of web activity was suitable in the 1990s, but is less suitable now, because many web pages contain embedded objects that come from a variety of different servers (such as content distribution networks and ad servers), rather than all coming from the same server. This undermines the notion of a conversation in which multiple objects are retrieved from the same server.

Similar models may be used for other types of interactive Internet traffic (e.g., email access, P2P file sharing, or media streaming). The differences among these types are due to the absence of specific levels, such as the level specifying the structure of web pages [575, 151].

Another level that may be relevant is the TCP connection. With the original HTTP 1.0 protocol in the early to mid-1990s, each object was retrieved using a distinct connection: the connection was set up, a request was sent, a response received, and the connection was closed. This was obviously inefficient, so the HTTP 1.1 protocol allows multiple objects to be retrieved using a single TCP connection, rather than opening a separate connection for each one.

Web Search and E-Commerce

The previous subsection considered the mechanics of web usage. It is also possible to create hierarchical models based on the semantics of web activity. One special case of web activity is web search. It has been suggested that user search activity is actually hierarchically structured, with the following levels [385, 185]:

1. The user has a *research mission* he or she wants to accomplish. To achieve this aim, the user needs to collect various pieces of information.
2. Getting each piece of information is a *goal* in itself. To accomplish a goal, queries are submitted to a search engine and the results are examined.
3. At the bottom are the individual *queries*. In many cases a single query does not produce the desired information needed to accomplish a goal, so additional queries are needed.

Similarly, a three-level model has also been suggested for e-commerce [487]:

1. *User sessions* represent visits of users to the online commercial site.
2. Each session is composed of a sequence of *business functions*. These are requests to browse, search, and view items, as well as the commercial functions of adding an item to the shopping cart and actually buying it.
3. At the bottom are the *HTTP requests* between the user's browser and the site. The difference between these requests and the business functions is that each function is typically implemented by a multitude of requests (e.g., for graphical elements that are displayed as part of the web page).

Computer Communications

The web access models just discussed are oriented towards user browsing activity on the web. A related model may be applied to the data traffic as it is observed on the network. A simple model of this type is the packet train model [369]. This model postulates that packets are not independent, and do not arrive individually. Instead, they come in trains — sequences of packets in rapid succession one after the other, followed by an idle time until the next train. Although this is a phenomenological model, it is based on the fact that sending large datasets is done by partitioning them into multiple fixed-size packets.

Newer models may have more levels that are related to the mechanics of the communication protocols. For example, the levels may be related to connections, flows, and packets [252, 251], and can include detailed characteristics of the communication protocols (e.g., the flow control window size that determines how many packets are sent before waiting for an ack). In addition, the model can extend upward to include user behavior. For example, the top level may be user sessions, each of which contains multiple connections [522, 484, 710].

Hierarchical models such as these have become especially popular due to their possible use in modeling self-similarity. For this use, one postulates that the observed traffic is actually the aggregation of multiple on-off processes, with on and off times that come from a heavy-tailed distribution (see Section 7.5.5) [732]. Each of these processes therefore has two levels: one that creates the envelope of on and off periods, and another that populates the on periods with activity.

Database Systems

Database systems can be characterized as being composed of three quite different levels [11]:

1. At the top is the enterprise level, which includes the set of applications that embody the business logic of the enterprise. The workload at this level is concerned with information needs at the business level.
2. The second level represents the database management system, including the database schema with its tables and indices. The workload at this level may be represented as SQL queries.

3. The third level is the physical storage system disks. The workload here is the basic I/O operations.

File System Activity

The I/O operations on a file have also been shown to be bursty, and are well modeled by a two-level process [722]. The top level defines the arrivals of bursts and their durations. The bottom level models the arrival of I/O operations within a burst.

A hierarchical model can also be justified based on the observation that user sessions lead to the execution of applications, and applications perform I/O operations on files, with each application invocation only accessing a small part of the file system.

Execution of Parallel Applications

The performance of parallel computations may depend in critical ways on the details of what the application does, as this may interact in subtle ways with the architecture. It has therefore been proposed to model the internal structure of parallel workloads using a three-level hierarchical model [100]:

1. The applications that make up the workload.
2. The algorithms embodied in the applications.
3. The routines used to implement these algorithms.

8.3 User-Based Modeling

User-based modeling attempts to achieve all the benefits listed in Section 8.2.2 by using a three-level model of how users generate workload. In this section we describe these three levels, providing a detailed example for the construction of a hierarchical generative model.

8.3.1 Modeling the User Population

The top level of the workload generation model is the *user population model*. As its name suggests, it is involved with the modeling of the user population. The idea is that at a sufficiently long time scale the user population may be expected to change. The user population model mimics this change by modeling the arrivals of new users and the duration during which they use the system. Modeling durations induces a model of user departures, so that the population size will fluctuate around a certain value and will not just grow with time.

The user population model has three main objectives. One is to set and control the average population size. This is important because the population size dictates the level of load that is produced. Another is to facilitate a turnover of users. The changing population of users is one of the vehicles by which locality of sampling is produced, due

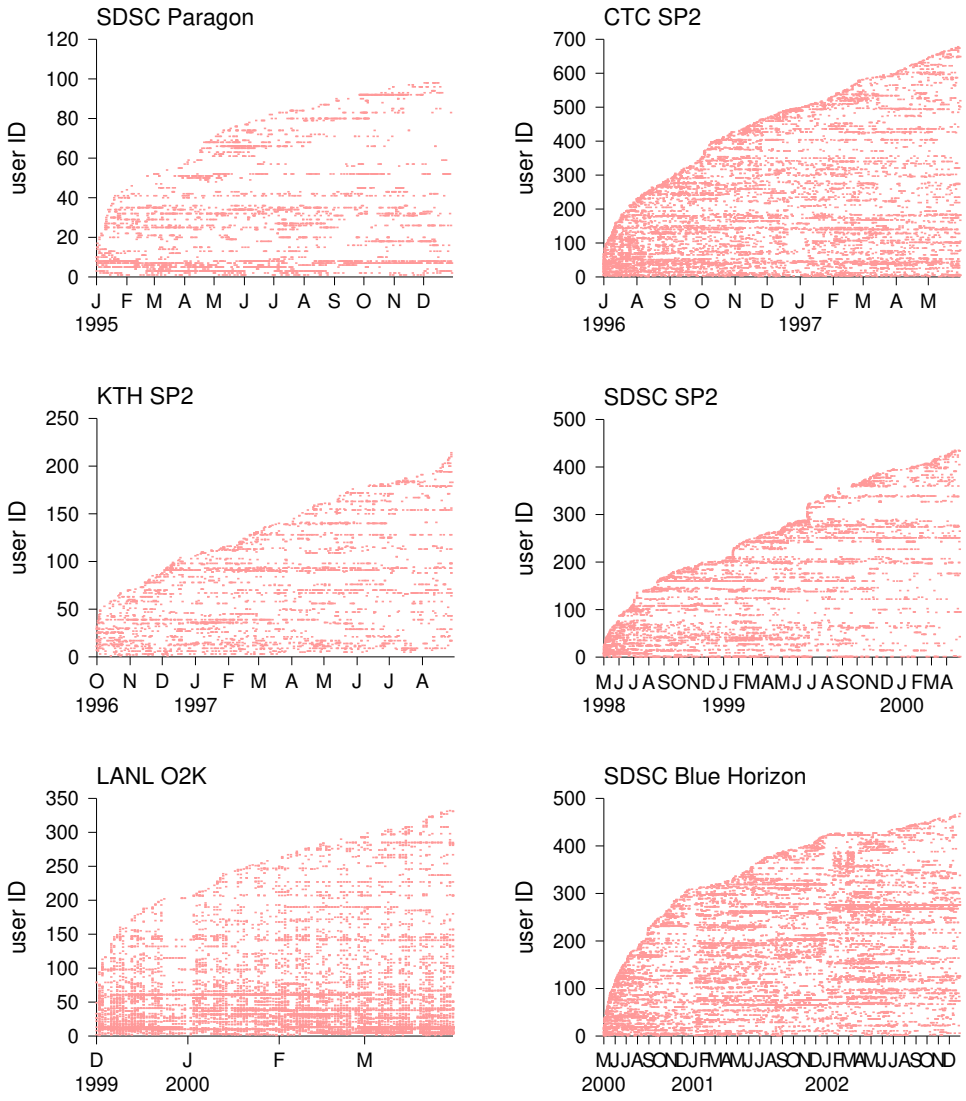


Figure 8.8: *Activity of users in logs showing new arrivals.*

to the different types of jobs that are created by each user's session model. The third objective is to generate self-similarity by allowing certain users to remain in the system long enough to induce long-range dependence.

User Arrivals

Locality of sampling is achieved by a changing user population. We therefore need to model the way in which new users arrive and residing users leave.

Figure 8.8 shows data regarding the arrival of new users. Actually this is a scatterplot

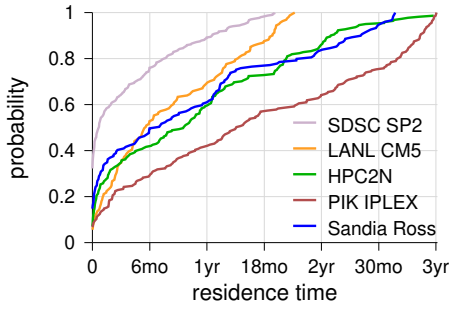


Figure 8.9: *Distributions of user residence times on different parallel machines.*

in which the X axis is time and the Y axis is user IDs; a point represents activity by that user at that time. The twist is to assign user IDs in the order of first activity. This leads to the growth profile seen at the left of each plot.

Initially, population growth seems to be very rapid. This growth reflects the first occurrences of users who were active when data collection commenced. Then comes a region of slower growth, which actually reflects the addition of new users. This growth has a more or less constant slope, so a model where user arrivals are uniformly distributed across time seems appropriate. In other words, user arrivals may be modeled as a Poisson process. While there are fluctuations that may indicate that a bursty or self-similar model would be more appropriate, there is not enough data to verify this.

Residence Time

The goal of the user population model is to create a fluctuating user population, with different users being active at different times. In addition, it creates a long-range correlation in the workload due to the activity of users who remain in the system for a long time. Heavy-tailed residence times lead to self-similarity [732].

The model should fit three characteristics of the original workload logs: the distribution of new user arrivals, the distribution of user departures, and the distribution of user residence times. We suggest modeling arrivals and durations, and checking that the induced distribution of departures also provides an adequate match. This approach is better than modeling user arrivals and departures, which would face the difficulty of deciding which user should leave each time — a decision that would affect the important residence time distribution.

The distribution of residence times is easy to extract from a log using the first and last appearance of each user. Naturally, care should be taken regarding users who seem active from the beginning of the log, or who remain active toward its end. To avoid such problems, we only use logs that are at least two years long, and delete from consideration all users who were active mainly during the first and/or last months.

Data for the distribution of residence times is shown in Figure 8.9. Two observations are clear. First, the distribution does not have a heavy tail; it may be better modeled as

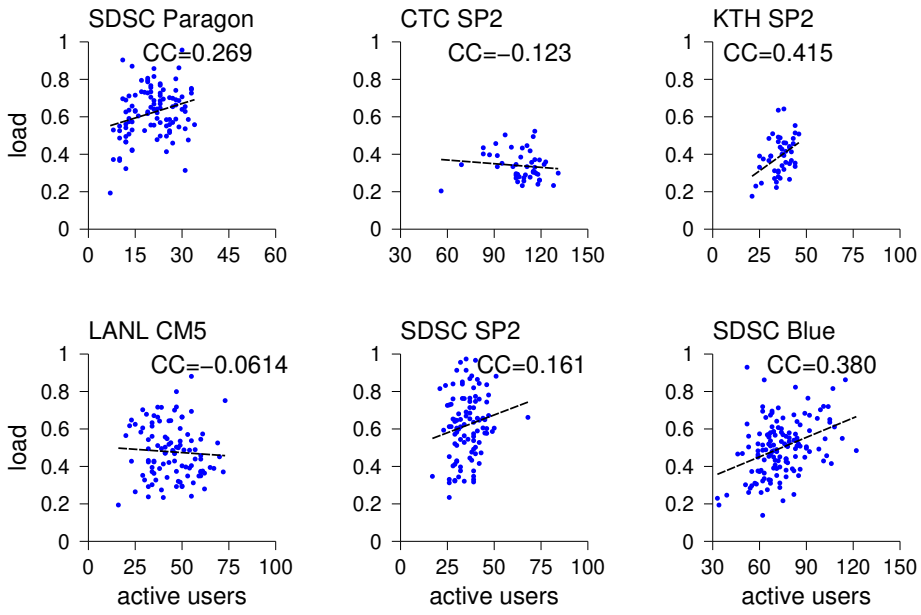


Figure 8.10: *scatterplots showing the number of active users and resulting load, using a resolution of one week.*

being roughly uniform, with added emphasis on short residencies. Second, the distributions seen in different logs are strongly affected by the lengths of the logs. This means that users with very long residence times are present. However, note that they may be dormant for extended periods in the middle. It may therefore be better to consider spurts of activity that come at large intervals as representing distinct users, even if in reality they are recurrences of the same user.

Population Size and Load

The main parameter of the user population model is the desired average population size. In principle, this can be used to regulate the level of load [57, 108, 751]. The problem is that different (simulated) users display very different levels of activity, so the effect of population size on load can only be expected to be reasonably accurate for long runs. On short time scales, big fluctuations may occur. Another problem is that, in reality, users may exercise self-regulation, and reduce their job submittal rate when the system becomes overloaded (such feedback-based behavior is discussed in Section 8.4).

Figure 8.10 investigates the correlation between the number of users and the generated load by showing scatterplots that compare these two variables for successive weeks. Of the six systems checked, four have some positive correlation between the number of active users in a given week and the load in that week, but the correlation is low. In the other two systems, there seems to be no appreciable correlation. In all systems, the clouds of points are rather diffuse, indicating the wide dispersal of values. The conclu-

sion is that it is not clear that in real workloads higher loads are a result of more users being active. Thus an alternative user-based mechanism for modifying system load is to change the characteristics of the users themselves (the options for load manipulation are discussed in Section 9.6.5).

8.3.2 Modeling User Sessions

The *user sessions model* is concerned with the pattern of activity displayed by a user during the period that he or she uses the system. It partitions the total duration of residence into sessions of intensive use, separated by times of idleness. The regularity of the pattern of sessions depends on the user's personality: some have very regular activity patterns, whereas others do not. In most cases, the majority of users work regular hours, and their sessions are correlated to the time of day. As a result they are also correlated to each other, and create a daily cycle of activity. Session times that are heavy-tailed also contribute to the generation of self-similarity.

User Personalities

The pattern of activity exhibited by a user has two main dimensions: the timing and extent of sessions, and the level of activity. In many cases, there is a limited number of user classes, such that all the users in the class display similar behavior. These classes can be found by clustering analysis, in which each user is represented as a point in a multidimensional space [108]. The coordinates of a user's point correspond to that user's workload attribute values (e.g., the average rate of submitting jobs, which is the reciprocal of the average interarrival time). The clustering then reveals different user types, with different patterns of activity.

For example, Haugerud and Straumsnes suggest the following types of users for a typical office or academic environment [321]:

- Standard users working 9 to 5 (i.e., working continuously during normal hours).
- Users starting earlier than usual, but then working normally.
- Users working normally, but staying later than usual.
- Users who only check their email from time to time.
- Users who work mostly at night.
- Users who are logged on all the time, including system processes.

These different user types are not deterministic — standard users do not all arrive exactly at 9:00 AM every day and leave promptly at 5:00 PM. Rather, they can be described by appropriate probability distributions. Thus standard users have a high probability of arriving at around 9–10 AM and a high probability of of leaving at 5–6 PM. For other user types, the distributions are adjusted according to their class of behavior. Workload patterns seen at different times can then be explained by different proportions of these user types.

A more complicated model was suggested by Zilber et al., based on an analysis of several parallel supercomputer workloads [763]. Noting that users sometimes modify their behavior, they start by identifying five types of sessions, rather than users:

- Normal daytime interactive work between about 9 AM to about 5 PM each day, representing normal working habits. In the study, 43% of all sessions were of this type. They are typically short sessions of just a few short jobs.
- Interactive sessions during night hours. In the study, 29% of user sessions were of this type. They may represent "night owls" who work on a shifted schedule. The jobs in these sessions tend to be longer than those during the day.
- Interactive sessions on the weekend. 21% of user sessions were of this type.
- Highly parallel batch sessions. Although only 4% of the sessions were of this type, they accounted for the majority of large jobs utilizing half the machine or more. These jobs typically ran for several hours, much longer than the jobs of the interactive sessions. Most of these sessions consisted of only a single job.
- Batch sessions dominated by very long jobs. Only about 3% of the sessions were of this type, but again, they accounted for a much larger fraction of the utilized resources. This was the result of having multiple jobs that were much longer than the jobs in any of the other session types — with half of the jobs running for more than two days.

Zilber et al. then went on to identify four different user types, each characterized by a different mix of session types.

Both these studies focused on the variability among human users under normal conditions. In addition, it is possible to envision other special types of user behaviors, such as

- A worker facing a deadline, resulting in a hysterical sequence of random and intense sessions.
- A robot (i.e., a software agent or periodically activated script) performing regular bursts of completely deterministic activity.

As the above examples show, the user personality dictates the distribution of sessions: when they start each day, how many there are (e.g., does the user break for lunch?), and how long they are. These characteristics should persist throughout the residence time of the user, or at least across several sessions, to enhance locality of sampling. Combining the sessions of all the active users should lead to the desirable overall activity pattern. For example, assuming that most users work normal hours, such a model will lead to more active users during work hours and less at night and on weekends.

In addition to having different personalities, users also have different levels of overall activity, possibly based on a Zipf-like distribution [764]. Thus two users of the same type need not have an identical activity profile — only the "envelope" of their activity is similar, but the contents may be quite different. Such variability is discussed in Section 8.3.3.

Realistic models that embody these ideas need to be based on data regarding user sessions, data that is often not explicitly available in logs. It is therefore necessary to develop methods to extract user sessions from the logged activity data.

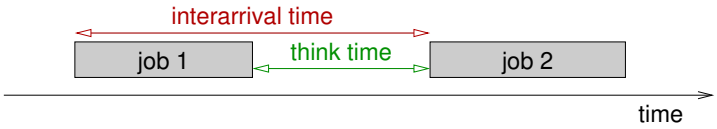
Recovering Session Data

In many cases workload data includes a record of individual user actions, but not of user sessions. It is therefore necessary to recover data about sessions in order to be able to model them.

As a concrete example, we will consider a system in which a user submits jobs for processing. A session is then a sequence of job submittals, and a break in the sequence is interpreted as a break between sessions [34, 618, 74, 614, 601, 385]. However, jobs always have some gaps between them, even if they are part of the same session. Therefore the question is one of defining a threshold: short gaps, below the threshold, are assumed to be gaps within the same session, whereas longer gaps are taken to represent a gap between sessions. Note that, although our discussion is couched in user-based models, the same ideas apply to practically all hierarchical models. For example, the threshold used to define a session is analogous to the maximum allowable inter-car gap of the packet train model of communication [369].

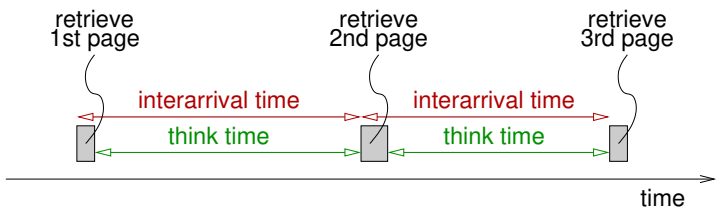
Practice Box: Interarrival Times and Think Times

When extracting session data, an important consideration is whether to look at interarrival times or at think times. The difference between the two is the actual duration of the work being done by the system. The interarrival time is the interval from the arrival of one job to the arrival of the next, whereas the think time is the interval from the termination of the first job to the arrival of the next; it derives its name from the notion that this is the time in which the user thinks about what to do next:

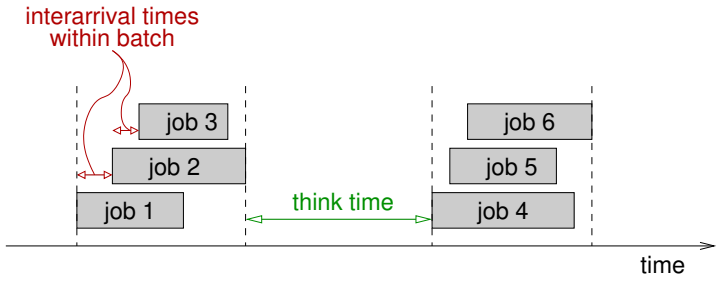


Looking at interarrivals is justified by claiming that they represent the intervals between successive actions taken by the user [749]. Looking at think times is justified by the fact that they represent the user's part in the interaction with the system; a long interarrival may still be part of the same session if the user was waiting all this time and reacted quickly once the system responded.

In interactive settings the distinction may not be so important. For example, in web browsing the actual retrieval of web pages typically takes no more than a second or two, but once the page is displayed the user may take some time reading it and deciding what link to click next. Thus the difference between the interarrival time and the think time is small. (Recall that we are defining these quantities from the system's point of view. In this case the system is the web server, so arrivals are requests to retrieve web pages and terminations are completions of serving such requests.)



A slight complication may arise in batch systems, or, in general, when work can be done asynchronously in the background. In such a scenario the user may submit several jobs concurrently. One possible interpretation in this case is to consider interarrival times between the jobs that make up a single batch, and think times between the batches [614, 749]:



However, the main problem is handling long jobs. Consider a case where a user's activity includes a very long job, say one that runs overnight. It would be wrong to assign jobs that were submitted alongside it and jobs that were submitted the next day soon after it terminated to the same session. The bottom line is therefore that using interarrival times is preferred [749].

End Box

To investigate possible relationships between gaps and sessions, we plot a histogram of interarrival times for the sequences of jobs submitted by each user. In other words, given a log of activity by multiple users, first partition this log into subsequences belonging to different users. Then find the distribution of interarrival times for each such subsequence. The sum of all these distributions, for the specific case of interarrivals between parallel jobs, is shown in Figures 8.11 and 8.12. It seems that the body of the distribution is limited to something between 10 and 20 minutes, although it has a significant tail, possibly with some spikes. Using similar data from parallel supercomputer logs, Zilber et al. have suggested using a threshold of 20 minutes on think times [763]. Nearby values, such as 15 minutes or 30 minutes, could be used as well. Zakay has suggested a threshold of one hour on interarrival times [749].

The problem with selecting a threshold value is that the distribution appears monotonic. Conceptually, we may imagine it to be composed of short intervals between jobs in the same session, and long intervals between sessions, but we do not see such a bimodal structure [385]. However, an underlying bimodal structure may still exist, and may be uncovered by using the EM algorithm to fit the data to a mixture of two lognormal distributions [282]. A good fit may be obtained even if there is substantial overlap between the two distributions, and the point of equal probabilities of being in either can then be used as the threshold.

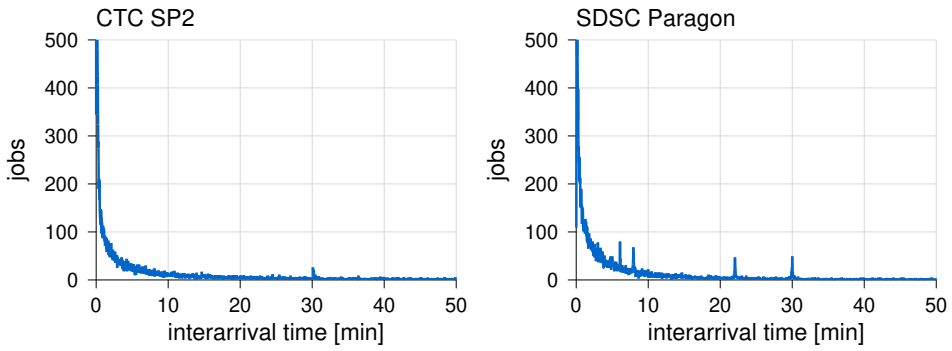


Figure 8.11: *Histogram of interarrival times between job submittals by the same user, for all users in each log of parallel jobs.*

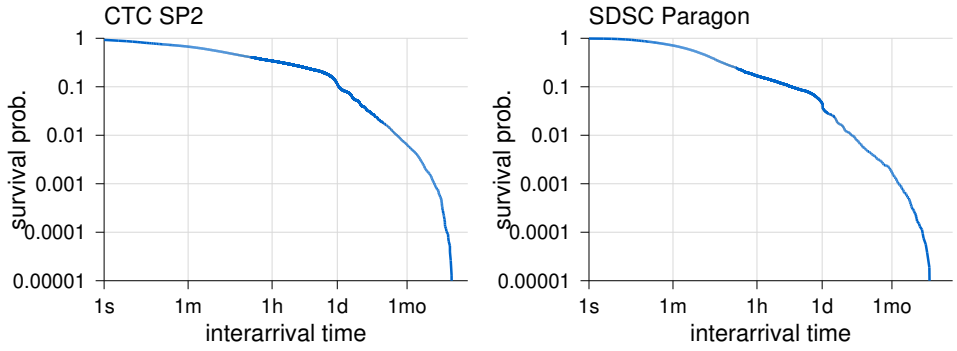


Figure 8.12: *LLCD graphs of interarrival times on parallel machines indicate that, although very long interarrivals may occur, the distribution does not have a heavy tail.*

An alternative for setting the threshold is to decide arbitrarily on a certain percentile. For example, we could decide that the longest 10% of the values represent session breaks. However, this is ill advised because it may lead to unreasonable results. Looking more closely at the data as in Figure 8.13, we find that for the CTC data, the 20-minute threshold comes at 59.0% of the tabulated interarrivals, and for the SDSC Paragon data at 74.6%. Setting the threshold according to a probability (e.g., defining the tail to be the top 10% of the samples) leads to a threshold of about one day for CTC (that is, every interarrival shorter than a day is still considered part of the same session), and just over five hours for the SDSC Paragon — values that seem quite high for the notion of a continuous session. The reason for these high values is that the distribution of interarrivals is actually log-uniform, at least for the CTC log. In any case, blindly using a percentile without checking the consequences is unjustifiable.

A completely different approach is to use clustering, in which jobs that are adjacent to each other form clusters that represent sessions [722, 74, 509]. This technique also

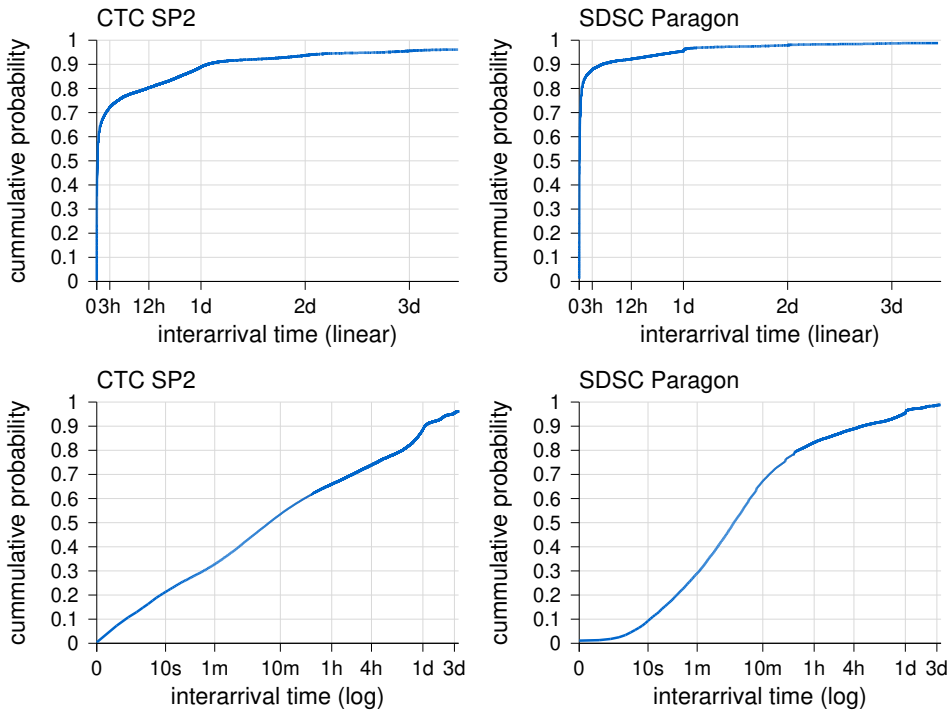


Figure 8.13: *The distribution of interarrival times between job submittals by the same user is (approximately) log-uniform.*

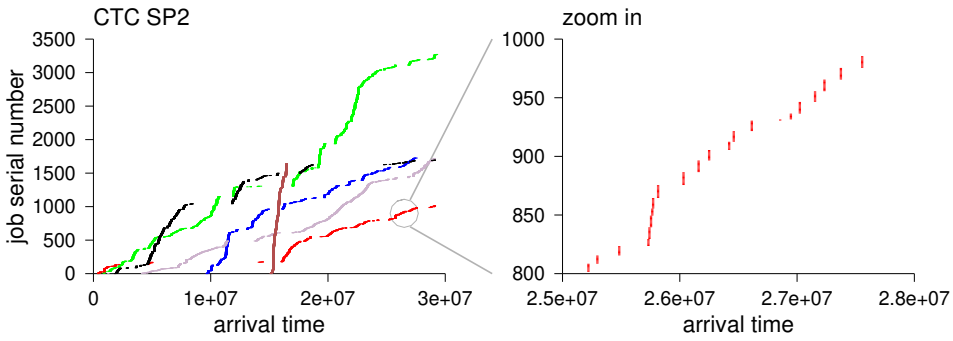


Figure 8.14: *Depiction of arrivals by active users (same ones as in Figure 8.2).*

has the advantage of tailoring the threshold for each user separately, rather than assuming that a single threshold is suitable for all users [483].

For intuition, consider a graph of jobs as a function of time. An example is shown in Figure 8.14: the X axis represents time, and the Y axis is serial numbers, so jobs are represented by dots snaking from the bottom left to top right. But this is not a

uniform, continuous sequence; rather, it is made up of small steps, in which a sequence of consecutive jobs come in rapid succession. These are the sessions we are looking for.

The clustering used to identify the sessions is a one-dimensional, agglomerative, single-link approach. Initially all the jobs are independent clusters, and the distances between them are their interarrival times. We then repeatedly unite the two clusters with the shortest remaining interarrival. This will obviously create clusters that are composed of subsequences of jobs, which is what we want.

The problem, of course, is to know when to stop. If we look at the distribution of interarrival times, interarrivals within a session are expected to be smaller than those between sessions. To find the threshold automatically, it has been suggested to sort the interarrivals and look for the point where they suddenly become bigger. This can be achieved in either of two ways.

The first is finding when the second difference becomes significantly different from 0 (e.g. larger than 2) [722]. More formally, denote the arrival times by $t_1, t_2, \ldots$. The interarrival times are then $h_i = t_{i+1} - t_i$. Denote the sorted interarrivals by $h_{(i)}$. The first difference of this sequence is $d_i^1 = h_{(i+1)} - h_{(i)}$, and the second difference is

$$\begin{aligned} d_i^2 &= d_{i+1}^1 - d_i^1 \\ &= h_{(i+2)} - 2\,h_{(i+1)} + h_{(i)} \end{aligned}$$

A high second difference implies that the first difference grew significantly relative to previous first differences. This in turn implies that the interarrival time grew dramatically more than its normal growth.

The second approach is based on computing the standard deviation of the interarrival times. Again, sort the interarrivals from the short ones to the long ones. For each one, calculate the quotient of the interarrival divided by the standard deviation of all the shorter interarrivals [509]. Assuming the distribution of interarrivals is bimodal, the first long interarrival can be easily identified by its leading to a relatively large quotient.

Another procedure that can be used is to consider the effect of the threshold on the sessions that are found. Specifically, count the number of single-job sessions, two-job sessions, three-job sessions, and so forth, as a function of the threshold value [324, 487, 356]. As the threshold is increased, longer intervals are considered as part of the same session rather than a break between sessions. As a result the number of individual jobs will decrease, because they will be grouped into longer sessions. As this grouping happens, the number of two-job, three-job, and longer sessions will grow. But at some point the results will stabilize, and the numbers of sessions of different lengths will not change as much as before. This point can be used as the threshold. Arlitt and Menascé independently show that for web and e-commerce sessions this point happens at less than 20 minutes [34, 487].

These and other ideas have been applied in various contexts, such as web browsing and web search. Examples of threshold values that have been put forth are given in Table 8.1.

Context	Threshold	Reference
general web surfing	10–15 min	[324, 552]
	25–30 min	[113, 601, 298]
	2 hr	[500]
web search	5 min	[623]
	30 min	[191, 374]
	1 hr	[618]
mobile web access	5 min	[576]
e-commerce	30 min	[487]
Wikipedia edits	1 hr	[282]
blog access	30 min	[192]
telnet activity	20 min	[164]
whiteboard activity	30 min	[183]
parallel supercomputers	20 min	[763, 615]
	1 hr	[749]

Table 8.1: *Examples of using a global threshold to define sessions.*

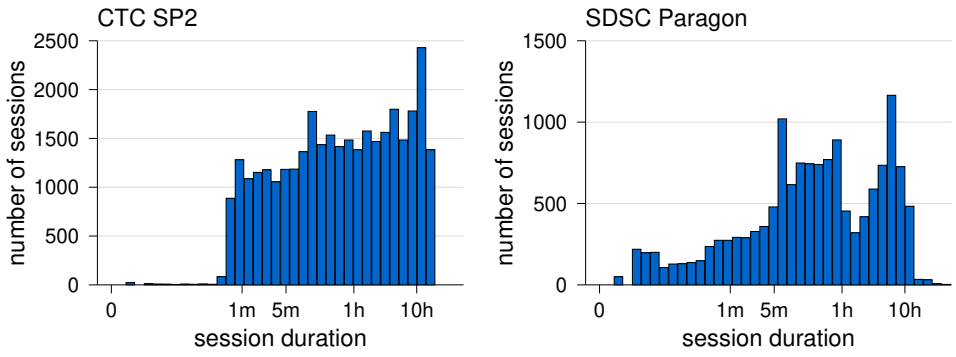


Figure 8.15: *Distribution of the durations of sessions recovered from parallel logs.*

Session Durations and Inter-Session Gaps

Once we decide how to break the sequence of jobs submitted by a user into sessions, we can tabulate the durations of these sessions. Figure 8.15 shows the resulting histograms. The good news is that the recovery of sessions seems to be reasonable, as the bulk of the distribution ends at between 10 and 20 hours. (A possible record holder is a 21-hour session reported from a Wikipedia editing marathon [282].) The bad news is that the distribution is hard to characterize. Notably these distributions do not seem to have a significant tail, let alone be heavy-tailed.

Similar data is obtained from a Unix server, in which login sessions are tracked explicitly by the system (Figure 8.16). The high number of 0-length modem sessions is

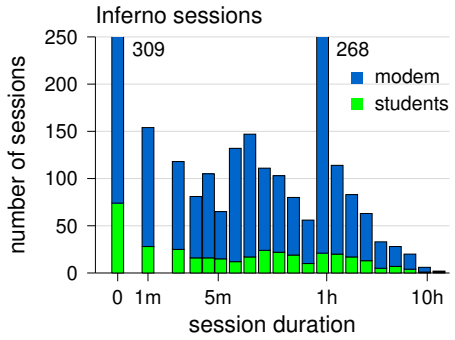


Figure 8.16: *Distribution of the durations of login sessions on a Unix server (excluding staff).*

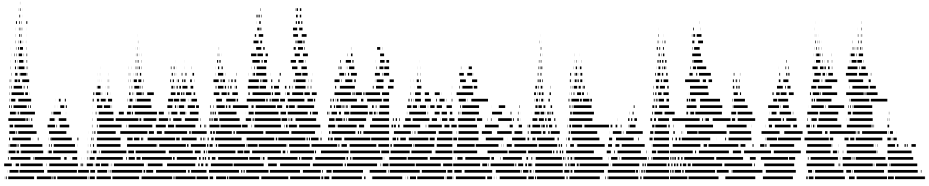


Figure 8.17: *Sessions tend to overlap according to the daily cycle (individual sessions are represented by line segments, drawn at different heights only for visibility, with the longer ones on the bottom). Data for about 3 weeks from the CTC SP2.*

probably the result of sessions that suffered from connection problems. The high number of 1-hour sessions could be the result of automatically closing connections that had not been active.

In other domains, however, the distribution may be different. For example, data from DSL lines (which provide home Internet access) shows that sessions are typically short, in the range of up to 30 minutes [458]. However, there are also many 24-hour sessions, which probably correspond to machines that are connected continuously, but need to reconnect once a day.

The inter-session gaps are simply the tail of the distribution of interarrival times, as shown in Figure 8.12 and 8.13. Although the tail extends to more than a month, it does not seem to be well modeled by a power law; a better model is the log-uniform distribution.

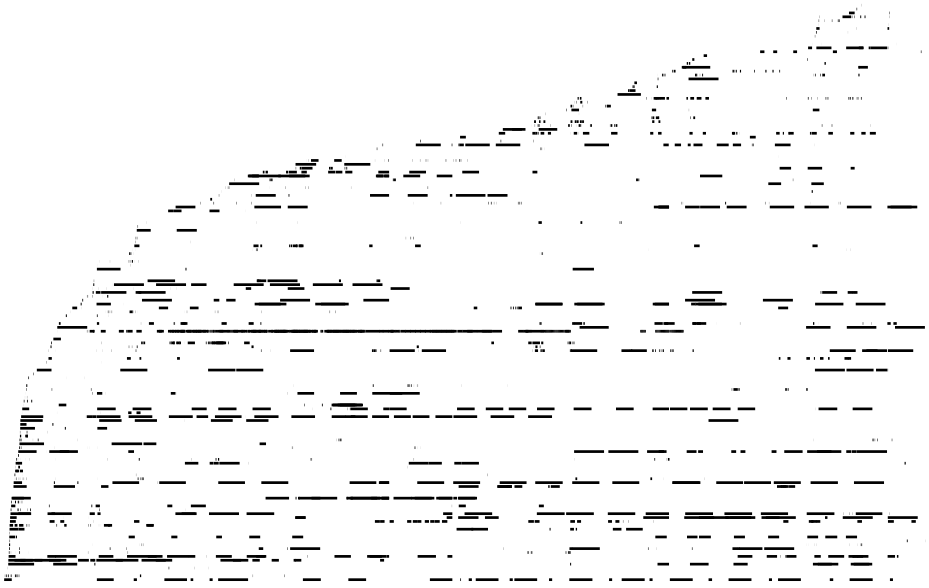


Figure 8.18: *The same data as in Figure 8.17, except that the vertical dimension is used to distinguish among users. Users are numbered by first appearance in this dataset.*

Correlation among Users

The goal of the user activity model is to divide the residence time of users into sessions. The correlation of sessions with time of day leads to the daily activity cycle. Indeed, we find that sessions tend to overlap each other, creating a daily cycle (Figure 8.17).

However, when we look at the activity of individual users (Figure 8.18), we see a mixed picture. Although some users exhibit regularity in their sessions, none do so across the whole span of time plotted here (only about three weeks). Moreover, many do not display any regularity. However, it must be remembered that we are dealing with the execution of jobs on a supercomputer. It is possible that the same users exhibit much more regular activity on their personal workstations.

User Freshness

A special effect that is sometimes observed is a change in behavior that is induced by how long the user has been using the system. It appears that new users tend to use the system more heavily than seasoned users [309]. However, the effect wears out rather quickly. It has therefore been suggested that a user's first week of activity be modeled separately, but subsequent weeks may share the same model.

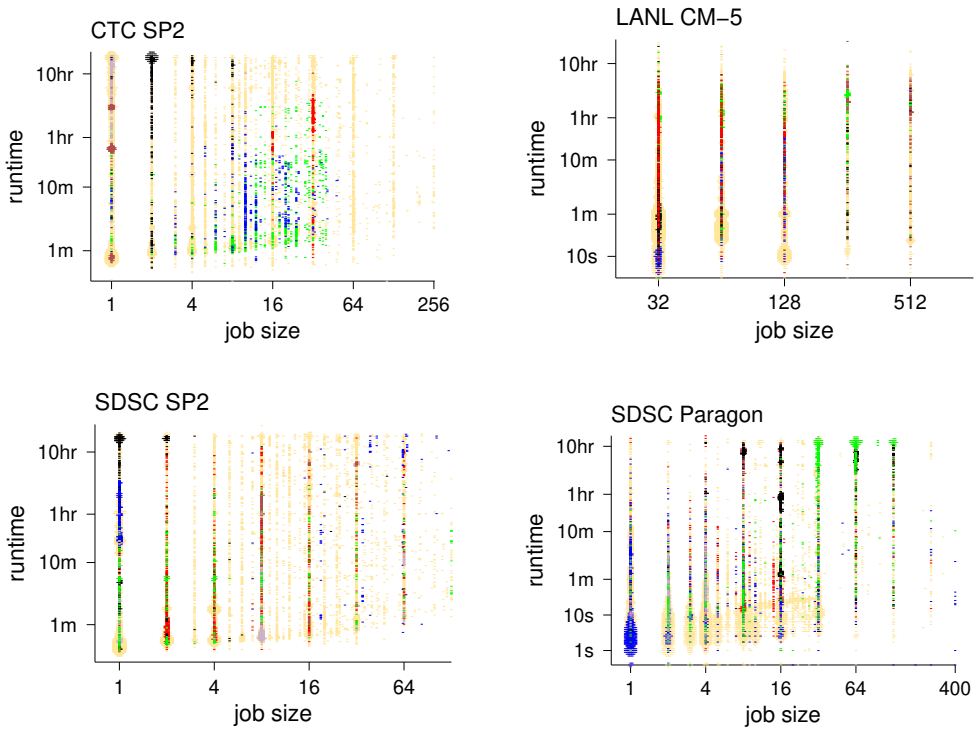


Figure 8.19: *The distributions representing specific highly active users are different from the overall distribution; in many cases, the jobs of a specific user cluster together. In each log, the six most active users are color-coded.*

8.3.3 Modeling User Activity within Sessions

The population and session models are responsible for characterizing the structure of the workload. The actual sequence of jobs is produced by the *activity model*. Jobs run by the same user in the same session are often repetitions of one another. This directly leads to locality of sampling. By explicitly creating dependencies among jobs (e.g., a simple chain in which the submittal of each job depends on the termination of the previous one [230]) a level of feedback is obtained. This can be combined with think times between jobs to prevent excessive regularity.

Anatomy of a Session

The activity model is somewhat similar to models commonly used today — it generates a sequence of jobs by sampling distributions. The difference is that the distributions are expected to be more modal, and can be different for each simulated user [258], thus creating an effect of locality of sampling when the user is active, but the correct overall distribution when all users are considered over a long simulated period.

The distinction between the distributions describing the activity of different users is

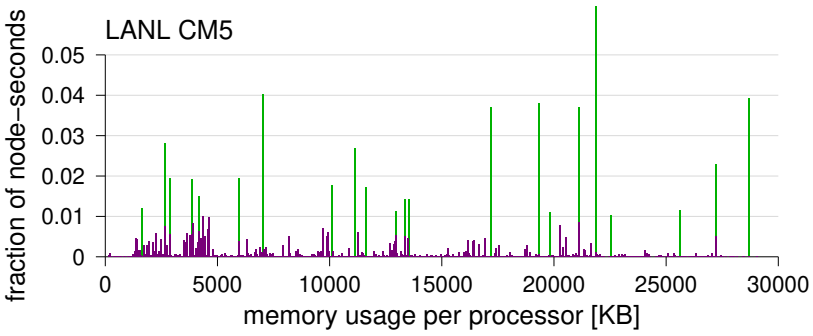


Figure 8.20: *The distribution of per-processor memory usage on the LANL CM-5, using a linear scale and buckets of 10 KB. Jobs are weighted by runtime and degree of parallelism to produce the distribution that would be observed by random sampling. The emphasized discrete components of the distribution are attributed to specific users in Table 8.2.*

illustrated in Figure 8.19. This shows scatterplots of parallel jobs on the well-known coordinates of job size and runtime. The jobs generated by different active users tend to cluster together at different locations, indicating that they are different from each other and from the overall distributions observed in the full log.

The modal nature of workload distributions and its user-based origin are demonstrated in Figure 8.20. This shows the distribution of memory requirements (measured on a per-processor basis) of jobs running on the LANL CM-5 in 1996 [231]. Using a fine granularity of 10 KB, it is obvious that the distribution is highly modal: it is composed of a number of very high discrete components, and very low "background noise".

Table 8.2 contains information about all the discrete components that represent more than 1% of the total node-seconds. For each one, users who individually contributed more than 1% are identified. In all but one case, individual users contribute the lion's share of these discrete components of the distribution.

The data in Figures 8.19 and 8.20 indicates that the distributions of workload attributes, when confined to a single user session, tend to be modal. But this data does not say anything about the temporal structure of sessions. For this we need to look at the distribution of think times, defined to be the interval from the termination of one job to the submittal of the next job in the session.

Tabulating think-time data from parallel supercomputers leads to a surprising result (Figure 8.21): many think times are negative! This simply means that the next job is submitted before the previous one terminated, so the term "think time" is actually inappropriate (Figure 8.22). Thus sessions for this type of workload should include the concurrent submittal of multiple jobs by the same users. In other workloads such behavior is rare and sessions only include a linear sequence of jobs.

KB per			Node sec	
proc	User	of	%	of
1640	usr1	6	1.18	1.19
2650	usr3	10	1.96	2.71
2660	usr4	13	1.41	2.81
	usr3		0.88	
2900	usr5	7	1.37	1.93
3880	usr6	12	1.37	1.93
4180	usr7	8	0.42	1.51
	usr3		0.46	
4340		7		1.01
5950	usr5	4	1.55	1.94
7010	usr3	2	4.02	4.02
10120	usr9	4	0.70	1.77
	usr5		1.07	
11150	usr9	4	1.08	2.68
	usr5		1.60	
11600	usr10	2	1.73	1.73
12950	usr2	2	1.11	1.12
13380	usr3	2	1.44	1.44
13530	usr11	2	1.43	1.44
17180	usr12	2	3.69	3.71
19330	usr5	3	2.81	3.81
	usr9		0.94	
19810	usr12	1	1.09	1.09
21120	usr5	3	2.85	3.70
21890	usr9	4	2.04	6.21
	usr5		4.08	
22550	usr10	2	1.02	1.02
25630	usr10	2	1.14	1.15
27220	usr13	2	1.76	2.28
28700	usr10	2	2.21	3.94
	usr1		1.73	

Table 8.2: *Single-user contributions to discrete components of the distribution shown in Figure 8.20 that are above 1% of the total (adapted from [231]). Column 3 gives the total number of users contributing to this component. Column 5 gives the total fraction of node-seconds in this component, and column 4 gives the fraction contributed by the user specified in column 2.*

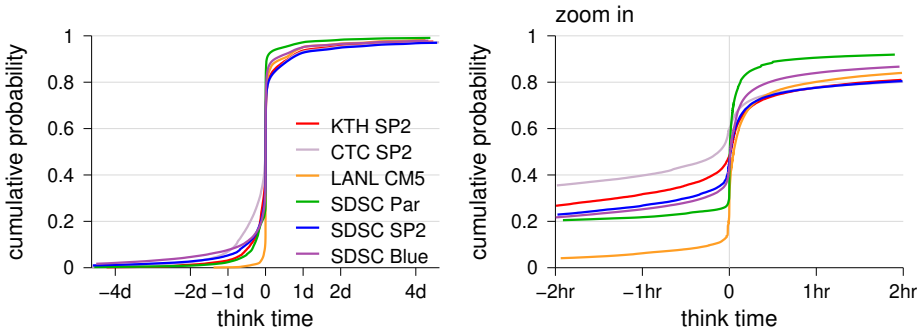


Figure 8.21: *Distribution of think times on parallel machines.*

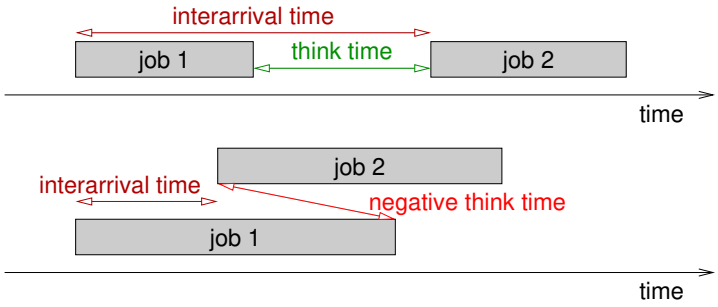


Figure 8.22: *Negative think times occur if a new job is submitted before the previous one terminated.*

Parameterized Users

In our models we want users who are repetitive, but different from each other. A simple way to achieve this is to create a parameterized user model, which generates a sequence of repetitive jobs. Then we can create multiple users, but give each one different parameters. To do so, we need to define the distribution of parameter values.

The simplest approach is to create users who only run a single job over and over again, at least within the same session. This is essentially the same as the model for creating locality of sampling described in Section 6.3.5 [239]. It is simple because we can use the global distribution directly, and do not need to compute another distribution for parameter values.

As a concrete example, consider a user model that is to generate jobs with a modal distribution such as the one shown in Figure 8.20. Each user session has a single parameter: the amount of memory-per-processor that its jobs should use. This is selected randomly from the global distribution. The effect is to generate sequences of jobs that have the same memory requirement, leading to the desired modal distribution and to locality of sampling.

An obvious problem with this approach is that it is extremely simplistic, and tends

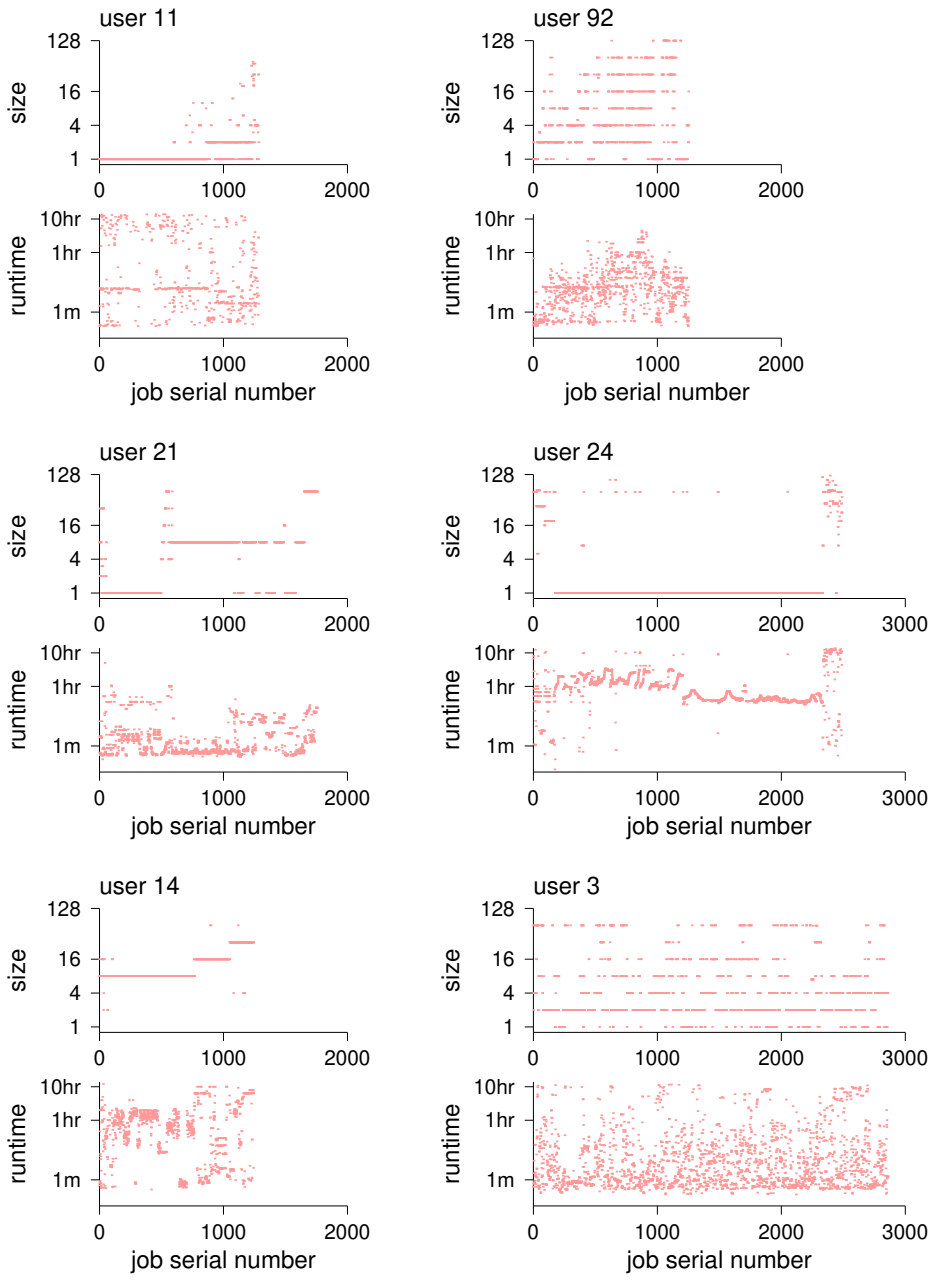
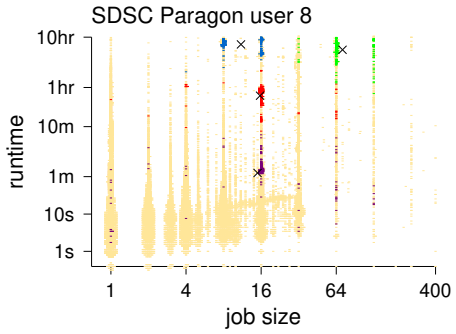


Figure 8.23: *Attributes of the jobs submitted by active users of the SDSC SP2 (the same as those depicted in Figure 8.19). Some, but not all, exhibit sequences of very similar jobs.*



Cluster centroids:

Clust.	*Size*	*Runtime*
0	11.0	25830
1	15.7	2472
2	71.9	20204
3	15.0	71.0

Figure 8.24: *The jobs submitted by user 8 in the SDSC Paragon log, clustered into four clusters. Cluster centroids are indicated by* $\times$*s.*

to reduce the variability present in each user's behavior way beyond what is observed in practice — in fact, it is reduced to a single set of values. This problem is illustrated in Figure 8.23. While some users, e.g. user 24, submit extremely long sequences of jobs that are indeed all very similar to each other, others, such as user 3, submit jobs with very diverse characteristics.

It is therefore appropriate to consider a more general approach, in which each user (or session) generates jobs from a distinct distribution. In this case, the distribution parameters have to be set so that the combined effect of all the users will lead to the desired global distribution.

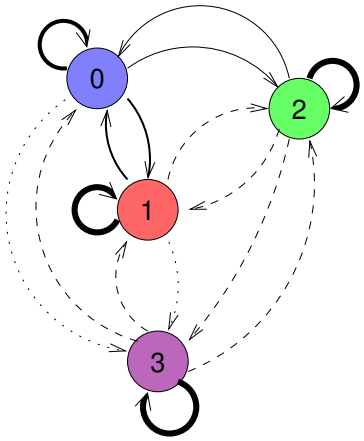
User Behavior Graphs

There are actually two problems with the parameterized model just suggested. One is that it assumes that all jobs are the same, overly reducing the variability in the workload, even for a single user (as noted above). The other is that because all jobs generated by the user are the same, there is no temporal structure. User behavior graphs are designed to solve the second of these problems, but also help with the first.

The idea of modeling users to capture the structure in the workload, and specifically of using a Markovian user behavior graph, is in fact quite old [258, 104]. The basis of this technique is to cluster the workload generated by a single user and then find a small number of representative jobs. These are then used to create a Markovian model. The states of this model are the representative jobs. The transitions are the probabilities that one job type appear after another. (For background on Markov chains, see the box on page 242.)

User behavior graphs are especially suitable when users display some repetitive activity, as in an edit-compile-test cycle. As an example, consider the activity of user 8 in the 1995 portion of the SDSC Paragon log. As shown in Figure 8.24, these jobs can be clustered into four clusters in runtime$\times$job-size coordinates. The clusters are characterized by their centroids, which are given in the accompanying table.

The resulting user behavior graph is shown in Figure 8.25. The line thickness reflects



Transition probabilities:

	To			
From	0	1	2	3
0	0.55	0.27	0.14	0.04
1	0.20	0.71	0.05	0.04
2	0.12	0.06	0.75	0.07
3	0.05	0.05	0.08	0.81

Figure 8.25: *User behavior graph for user 8 from the SDSC Paragon log, based on the clusters of Figure 8.24.*

the probabilities associated with the different arcs. The strong self-loops on all four states indicate that there is a high probability that a job from a certain cluster will be followed by another job from the same cluster, indicative of locality of sampling; the lowest such probability is 0.554 for cluster 0, and the highest is 0.814 for cluster 3. Note that the arcs between pairs of states are not completely symmetric. For example, a job from cluster 3 has a probability of 0.051 of being followed by a job from cluster 1, but the probability of the other direction is only 0.037.

Although user behavior graphs provide more variability than the single-job models of the previous subsection, they nevertheless still tend to reduce the variability present in each user's behavior way beyond what is observed in practice. As seen in Figure 8.2, the work of individual users cannot always be represented by discrete clusters. This can be solved by using an HMM-style model, in which each state generates jobs from a distribution, rather than only repetitions of the centroid of the cluster (or even samples from the cluster [440]).

In addition, using a single user behavior graph does not facilitate the differentiation between users who behave in distinct ways (e.g. one who consistently prefers browsing through an e-commerce site versus another who prefers searching using keywords). Therefore multiple different user behavior graphs are needed, effectively leading to a multiclass workload model. This complicates the modeling considerably, because each user (or class) needs a separate model with multiple parameters describing the different states and the transitions among them.

Special Users

User-based modeling provides the option to add special users to model special conditions that are known to occur in practice. One example is workload flurries. Another is robots, that is, software agents.

Let us focus on workload flurries for the sake of concreteness. Workload flurries

are surges of activity, typically with well-defined characteristics and by a single user (see Section 2.3.4). They are similar to flash crowds in that they embody a workload that is significantly different and more intense than normal for a limited time span. The existence of such flurries may affect the performance of the system, either just because of the increased load, or due to some specific interaction between the flurry jobs and other jobs or the system behavior [696].

Once the possible existence and effect of flurries are acknowledged, an analyst may want to study their specific effects on a particular system design. A basic requirement for such an experimental evaluation is to have two types of workloads: a base workload without any flurries, and alternative workloads with different numbers of flurries. A simple way to achieve this is to define the set of jobs in each flurry to be a "user". Then the base case is obtained by considering the workload with these special users removed. The effect of flurries can be studied by not only including them but by also injecting repeated instances of these users to increase the prevalence of flurries [751]. This is an instance of user resampling.

8.3.4 User Resampling

A major issue with workload modeling in general is the question of representativeness: does the model indeed represent reality? Using the right distributions (including heavy-tailed distributions) and including correlations (with locality and self-similarity as special cases) are meant to make models "better" in the sense that they will be more representative. Alternatively, using log data directly or using benchmarks strive for representativeness by avoiding modeling altogether.

User resampling is a technique that attempts to combine the best of both worlds [750, 751]. The idea is to partition a workload log into sub-logs representing the activities of individual users, and then to recombine these sub-logs in different ways to create new workloads. Thus all the characteristic behaviors of users are retained even if the analyst using this workload is not aware of them. This includes both the internal correlations and structure exhibited by users, and the diversity between users.

At the same time, resampling provides the following important benefits over using the log as is:

- The option to create multiple statistically similar workloads. Assuming that the resampling of users is done randomly, doing the resampling again will lead to a different workload (that is, the same basic blocks will appear in a different order and possibly also a different number of times). This can be useful when calculating confidence intervals for performance metrics. Essentially, it is like bootstrapping at the user level.
- The option to create longer workloads than the original log. By continuing to resample (with replacement) from the pool of users, the generated workload can be as long as desired. This can be important in enabling simulation results to converge.

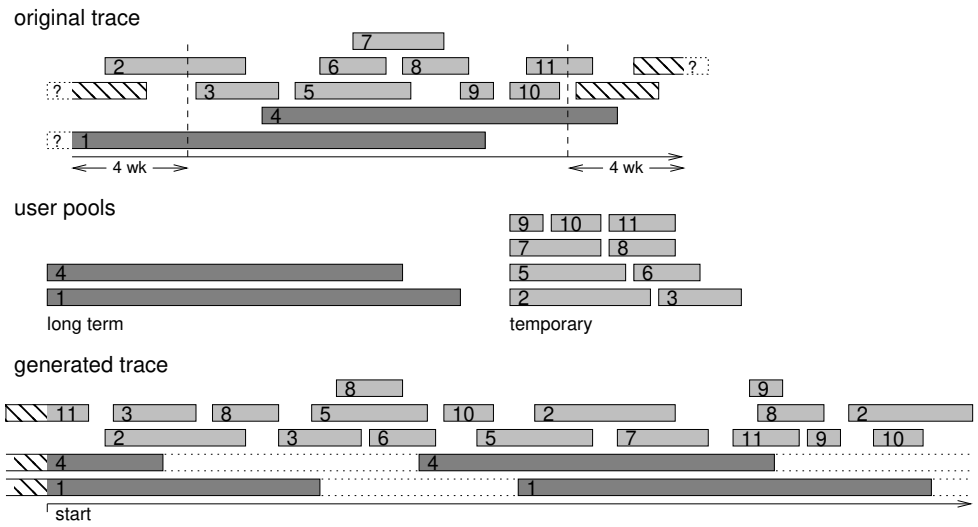


Figure 8.26: *How user resampling generates new workloads based on data from an existing log. Each rectangle represents the complete tenure of some user.*

- The ability to manipulate the workload, and, in particular, to change the load. Performance evaluations often seek to characterize the performance as a function of system load. But with a single workload log, we only have data for a single level of load. Resampling solves this problem by allowing us to change the average number of active users in the system, thus creating loads that are either lower or higher than the original load. At the same time, it also tends to produce a more consistent workload with reduced fluctuations in the load, thereby reducing the confounding effect of such fluctuations on evaluation results.
- The ability to give special treatment to select users. For example, certain users suspected as being robots can be eliminated from the resampling process. Conversely, users with special rare activity (such as those generating flurries) can be oversampled, in order to investigate the effect of these rare behaviors.

The idea of resampling has a long history in statistics, where it is used, for example, in bootstrapping to estimate the accuracy of sample statistics [205, 206]. But it has been used only in a very limited manner in connection to workloads. One example is the Tmix tool for generating network workloads [725, 331]. This tool does resampling not of the complete activity of each user, but of individual exchanges over the network.

User-based resampling has been suggested for parallel system workloads [750, 751]. A given log of system activity is partitioned into sub-logs containing the jobs of different users (Figure 8.26). These users are then classified into two classes: the long-term users who are active for a large fraction of the trace, and temporary users who are only active for a relatively short period. Users who appear only close to the beginning or the end of the traced period are discarded.

Given the pool of long-term users and the pool of temporary users, generating a new

workload proceeds as follows. To initialize, select at random a representative number of users of each type. Create the initial workload by combining the activity of all these users, where each user starts from a random week during his or her logged activity. Everything is done at the resolution of weeks, so jobs always start on the same day of the week and the same time of day as in the original log. This ensures that the daily and weekly cycles of activity are similar to those of the original workload, and that the different users remain synchronized with each other.

After initialization, select at random a certain number of new temporary users to add in each week of the generated workload. Randomize the actual number of new users, with the average being the number of new users per week in the original workload log. As for long-term users, when such a user ends it is regenerated after a certain interval. Thus the number of long-term users is essentially constant in the generated workload, as it should be if that number indeed reflects those users who are active all the time.

In this description of resampling, the activity of each user is reproduced exactly as in the original log: exactly the same jobs are submitted, at exactly the same times. An alternative is to do a second level of resampling within each user. Thus the user's activity can be partitioned into individual sessions, and resampling these sessions will produce a new sequence each time the user is used in the generated workload. It is also possible to add a feedback effect, such that the timing of successive job submissions depends on system performance.

8.4 Performance Feedback

A user, and especially an interactive user, cannot be expected to be oblivious to the behavior of the system. At the most basic level, successive jobs often depend on each other and constitute a well-defined workflow. Thus the next job will not be submitted until some time after the previous one had terminated. Because of such dependencies, system performance has an immediate effect on precisely when users submit new jobs.

Moreover, it is reasonable to assume that the user's future behavior will also be linked to the system's performance. If the system is responsive, the user will continue to use it and will generate more work. But if it is sluggish, the user will reduce the amount of submitted work and may even give up completely. Thus the workload generation model cannot be oblivious, but rather should be coupled to the system performance with a feedback loop. Importantly, this is generally a stabilizing negative feedback effect.

This notion of feedback may also be described in terms similar to the supply-and-demand curves commonly used in economics (Figure 8.27). One curve shows the response time as a function of the system load: this is the supply curve, and indicates what the system can provide. For low loads, the response time (cost) is low, because abundant resources are available; at high loads there is much contention for resources, and cost is high. The other curve shows the load that would be generated by users (the demand) under different performance conditions. If the system is responsive, users will be encouraged to submit more jobs and increase the load. But if the system is already overloaded,

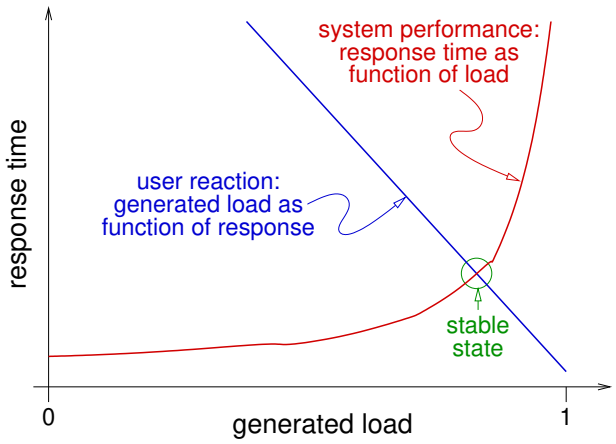


Figure 8.27: *Supply-and-demand curves for computational resources.*

and the response time is very high, users will tend to submit less additional jobs. The intersection of the two curves identifies the expected steady state of the system.

Figure 8.27 can also serve to point out gaps in our knowledge. The system performance curve is approximately that of any open queueing system, and similar curves are also common in simulation results. But the correct shape of the user behavior line, drawn here as a straight line, is actually a mystery.

The applicability of feedback is very broad. It is an obvious component of the activity model, in which it affects the timing of activities within the session. This is true both for fine-grained interactive work (including word processing, programming, or online gaming [81, 116]) and for more coarse-grained work such as the submittal of a set of jobs that depend on each other. But feedback may also have an effect at higher levels. It may have an effect on the user sessions model, for example when bad performance may cause a user to abort the session. Conversely, if a user has a certain amount of work that just has to be done, bad performance may cause sessions to be extended to make up for the delays. In the extreme case feedback may also influence the population model, as when bad performance causes users to give up on the system.

It is important to note that the inclusion of feedback in a workload model is not merely an academic curiosity. Feedback is, in fact, very important. Its importance stems from the fact that it may lead to a throttling of the load, as when a high load leads to bad performance and hence to reduced generation of new load. This is a self-regulating effect that reduces the risk of system saturation, and may be regarded as part of online behavior [241]. Thus models with feedback may lead to dramatically different results regarding system behavior under load; specifically, oblivious models will tend to generate pessimistic predictions, whereas models with feedback will be more optimistic regarding system robustness [614, 601, 688, 552, 752].

Evidence that such feedback effects indeed occur in practice is shown in Figure 8.28, which displays workload data from several parallel supercomputers. Each workload log is divided into week-long segments. For each such segment, the number of jobs is

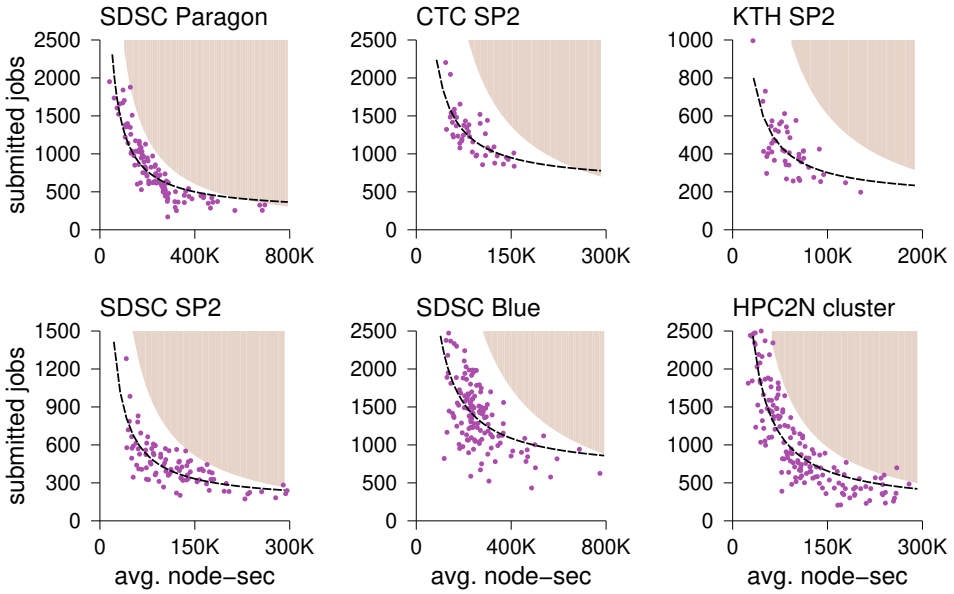


Figure 8.28: *The inverse relationship between the average parallel job resource usage, measured in node-seconds, and the number of jobs submitted per week. The fit lines are based on a linear regression of* Y *with* $1/X$*. The shading shows the forbidden area, i.e., combinations that represent more than the available resources on the machine.*

plotted against the average job resource usage, expressed in node-seconds. In all cases a negative correlation is observed, and more specifically, the data seem to fit an inverse relationship. This is conjectured to be the result of self-regulation by the users: either many lightweight jobs or fewer heavy jobs are submitted, but never many heavy jobs. Note that in some cases, notably the SDSC Paragon and the HPC2N cluster, this is closely related to the limit of available resources. But in others, notably CTC SP2, KTH SP2, and SDSC Blue, it is not.

Background Box: Feedback in Internet Traffic

Feedback effects are not unique to human users who have opinions regarding the quality of service they receive from the system. It is also used to regulate system load automatically.

The most prominent use of feedback to regulate load is the congestion control mechanism used by implementations of the TCP protocol [364, 531]. Congestion control was added to TCP in 1986, after several episodes in which bandwidth dropped by three orders of magnitude and all traffic essentially came to a stop. These episodes were traced to overloaded conditions that led to packets being dropped. As TCP provides a reliable connection, it uses a timeout to identify packets that have not been acknowledged, and then retransmits these packets. However, when the network is overloaded, such retransmissions have the undesirable effect of increasing the load even more and making the situation worse. The innovation of congestion control was to take the opposite approach: when packets are

dropped, the window of allowed unacknowledged packets is reduced, so as to reduce the load on the network.

As a result of congestion control, the timing of packet arrivals in a trace of Internet traffic contains a signature of the load conditions that existed on the network when the trace was recorded [265]. It is therefore not a good idea to use such traces directly to evaluate new designs. Instead, it is necessary to model the source of the communications (e.g., the application that is sending and receiving data). Furthermore, it is typically not necessary to model the feedback effects. Instead, one should use an implementation of TCP as part of the evaluation, and let it generate the feedback as needed for the specific conditions that exist in this evaluation.

End Box

Incorporating feedback into system simulations has profound implications. One major complication is that doing so undermines the basic notions of "load" and "equivalent conditions". With feedback, the load on the system is no longer an input parameter of the evaluation — instead, it becomes a metric of the performance. Thus identical loads cannot be used any more to claim that different simulations (e.g. with different system configurations) are being executed under the same conditions, as is needed in scientifically valid experimentation. Instead, being executed under the same conditions must be interpreted as *serving the same workload-generation process*.

The role of system load is not the only thing that changes. Other performance metrics and their interpretations change as well, as summarized in Table 8.3. Simply put, this is a result of modifying the underlying character of the system. With an oblivious workload, the system is open, as if the workload comes from an infinite population that is not affected by the system itself. But with feedback, we shift toward a closed system model, in which new jobs only replace terminated ones. As a result the response time metric should be replaced by the throughput metric, which further explains why "equivalent conditions" can no longer be interpreted as running exactly the same jobs, but rather as facing the same workload-generation process. We also obtain a more direct (albeit still debatable [181]) assessment of user satisfaction.

Dependencies and Think Times

The simplest way to incorporate feedback into a workload model is to postulate dependencies between jobs. Instead of assigning jobs arrival times in an absolute time scale (this job arrives at 5:37 PM), assign them arrivals relative to previous jobs (this job arrives 43 seconds after the previous one terminated). The relative delay is the think time, and can be modeled using the distributions shown in Figure 8.21.

It is important to note that the assignment of the new arrival is done relative to the previous job's *termination*, not relative to its *arrival*. This is the difference between think times and interarrival times, and is what allows feedback to have an effect. Modeling interarrivals is just a different way to model the arrivals themselves, and does not include any feedback from the system. Modeling think times, in contradistinction, implicitly takes the feedback from the system into account, because the think time is only applied

Metric	*Oblivious*	*With feedback*
load	compared with offered load to find the threshold beyond which the system saturates	measured to find the maximum utilization achievable — a performance metric
throughput	if not saturated, throughput is completely determined by the workload	achieved throughput is the main metric of system performance
response time	response time is the main metric of system performance	quoting response time is misleading without taking into account the throughput and user frustration
user satisfaction	assumed to be measured by the response time or slowdown	measured directly by the feedback model, e.g., in the form of frustrated user departures

Table 8.3: *Performance metrics under oblivious workloads and workloads with feedback.*

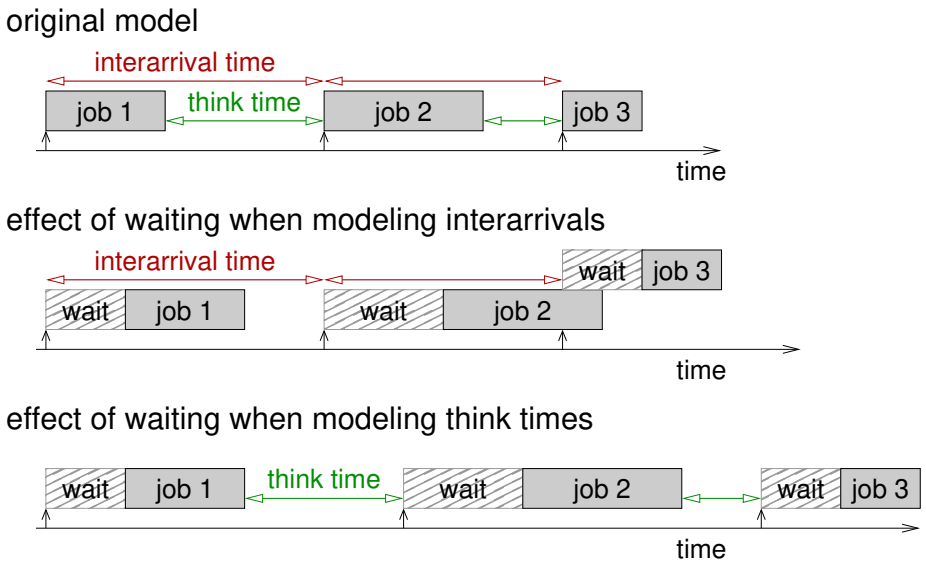


Figure 8.29: *Modeling think times instead of interarrival times implicitly includes feedback from system-induced wait times into the workload, and shifts the job arrival times. Modeling interarrivals may cause a job to be submitted before a previous job terminated, because the previous job was delayed for a long time.*

after the job terminates, which includes not only the runtime but also all waiting times imposed by the system (Figure 8.29).

(The above notwithstanding, if several jobs depend on a single previous job, and do not depend on each other, it may be advisable to also retain their interarrival times [752]. This avoids situations where in a simulation jobs that were originally separated become concurrent.)

The problem with modeling the effect of feedback using think times is that accounting logs used as data sources do not include direct data regarding the dependencies among jobs. Specifically, the problem occurs when some jobs overlap each other, meaning that job B was submitted before job A had terminated. Job B then obviously does not depend on A, but what previous job does it depend on?

One way to recover such dependencies is to match each job with the last job (by the same user) that had terminated before this job arrived [350]. This is similar to the approach used in the session-identification model described earlier, with one important difference: it avoids negative think times such as those found in Figure 8.21. This procedure may be applicable in storage systems, but it may be questionable in batch systems where jobs can be excessively long.

An alternative is to postulate that overlapping jobs actually represent batches of independent jobs, and that feedback only affects the intervals between such batches [614]. This still can be interpreted in two ways, depending on when we consider a batch to have ended: is it the termination time of the job that was submitted last, or the time at which all jobs in the batch have terminated? Zakay attempts to resolve this issue by comparing the distributions of think times produced by the two approaches, concluding that using the termination of the last submitted job is preferable [749].

In any case, a drawback of limiting feedback to the intervals between successive jobs is that it only affects the submittal rate; the general structure remains unchanged. Thus if the workload model includes a long sequence of jobs submitted by the user one after the other, and in a particular simulation these jobs take a long time to run, the entire sequence may be extended to unreasonably long periods. To avoid such scenarios, we need to model user behavior explicitly.

Effect on User Behavior

A more general model allows feedback to affect user behavior in more complicated ways [752].

One example is a user behavior model in which multiple inputs are used by the user. Thus additional submittals may depend not only on the performance experienced by recent jobs but also on various system parameters. For example, the user may consider the current queue length as a relevant factor, because a long queue indicates that future jobs are likely to suffer higher delays.

Another example includes tolerance in the user behavior model. With such a model, if the system performance deteriorates too much, the user ceases to submit new jobs and gives up. This can be temporary, as in the early termination of a session, or permanent, as when a user completely ceases to use a system.

The question, of course, is what model to use. Satyanarayanan and co-workers have suggested (but not verified) that user tolerance to delays be modeled using an exponential scale [593]. The context was the Coda distributed file system. This system tolerates disconnected operation by hoarding files on the mobile client at those times when it is connected to the main server. The issue was to balance user input about such caching with transparent operation. The suggested model was that if the expected time to down-

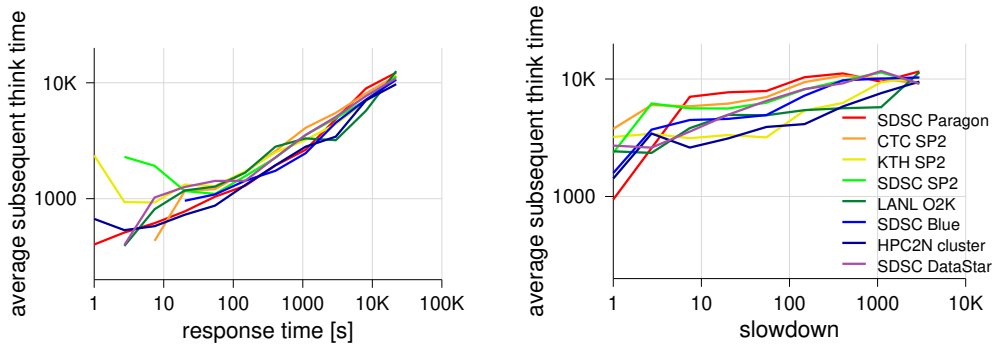


Figure 8.30: *Relationship between a job's performance, as measured by the response time or the slowdown, and the subsequent think time.*

load a file was less than a certain threshold — given that file's predefined priority, its size, and the current bandwidth availability — the file would be cached without asking the user. The specific equation for the threshold was $\tau = 6 + e^{pri}$ seconds.

Another model, for web download time tolerance, was proposed by Cremonesi and Serazzi [153]. They recommend that two thresholds be used, and suggested that users will be satisfied with downloads of up to 10 seconds but will not tolerate downloads that take more than 30 seconds. Moreover, these thresholds also depend on the stage of the interaction: the values quoted above are valid for the first three interactions with a site, then fall sharply by half by the fifth interaction, and continue to drop more slowly with additional interactions.

The problem with such models is that they are not based on real data, but on assumptions. At the same time, they are better than oblivious models in which the decision to abort is random and unrelated to performance, as in the model proposed by Tay and others [504, 688]. Still, experimentation with actual human users is sorely needed to faithfully characterize user behaviors and perceptions.

An alternative that is based directly on readily available log data is to look at the relationship between performance and think times [615]. The idea, due to Shmueli, is to analyze the data at the job level for each user. Thus we first partition the log into independent sequences of jobs by the different users. Then, for each user, we consider the activity pattern at the job level. In particular, we check the effect of each job's performance on the ensuing think time, that is, on the interval of time until the same user submits another job. Results of doing this are shown in Figure 8.30. The performance metrics, response time and slowdown, are binned in a logarithmic scale, and the average subsequent think time is computed for each bin, but excluding think times longer than 8 hours.

As the graphs show, when considering the response time metric, the average following think times grow in a consistent manner. The shift in behavior is quite pronounced, with short think times following short response times, and much longer think times following long response times. This implies that the think time distribution is not fixed —

rather, different distributions should be used based on the preceding response time, or at least the distribution should be parameterized based on the response time. In addition, long response times increase the likelihood of very long think times that indicate a session break.

Interestingly, the effect of slowdown on subsequent user behavior is much less pronounced. Slowdown is response time normalized by runtime: if we denote the time that a job waits in the queue by t_w, and the time it actually runs by t_r, then the response time is $r = t_w + t_r$, and the slowdown is

$$sld = \frac{t_w + t_r}{t_r}$$

This is expected to quantify how much slower the system appears to be, relative to a dedicated system; if the job does not wait at all, the slowdown is 1, and the more it waits, the higher the slowdown. It may therefore be expected that users will be more sensitive to slowdown than to response time. For example, if a 10-minute job waits for 5 minutes before running, its response time is 15 minutes and its slowdown is 1.5. But if a 1-minute job waits for 14 minutes, for the same total 15-minute response time, we expect this to be much more annoying to the user, and this is reflected in the slowdown which is 15. However, the data in Figure 8.30 indicates that slowdown is a much worse predictor of subsequent user behavior than response time [615].

Using the above, Shmueli also proposed a model for the effect of feedback on sessions. Setting the threshold representing session breaks at 20 minutes, the probability of not breaking the session (that is, the probability that the ensuing think time is shorter than 20 minutes) was found to depend on the response time according to

$$\Pr(\text{continue}) = \frac{0.8}{0.05\,r + 1}$$

where r is the response time in minutes [616]. But this needs to be combined with data about session starts and other aspects of user behavior in order to create a realistic model [752].

Another especially interesting option is to include learning in the model. Evidence from production systems indicates that the workload evolves with time, as users learn to use the system [346, 240]. This idea can also be applied to new users who start using an existing system. However, doing so requires a more careful analysis of the distributions of jobs submitted by individual users.

9

Case Studies

This chapter presents an assortment of published results pertaining to computer workloads. They range from a specific measurement or observation to a complete workload model for a particular domain. Regrettably, in many cases the available data is sketchy, and based on a single study from a specific system, rather than being based on a meta-study that summarized many original studies conducted under diverse conditions.

Likewise, many of these studies are not the final word in the sense that they focus on a single element of the workload, and not on the complete picture. A comprehensive model would require the identification of the most important parameters, their characterization, and a characterization of the interactions between them (e.g. [763, 671]).

Thus there are many opportunities for additional studies on workload characterization and modeling.

9.1 Human User Behavior

A chief concern regarding computer workloads is their dependence on changing technologies. Will measurements performed today still be relevant next year? One area that is excused from such considerations is that of workloads that reflect basic human behaviors. The issue here is the patterns of how users generate new work when they interact with a computer system. At a very basic level such patterns reflect human behavior regardless of the system and are therefore largely invariant.

This does not imply, however, that all human users are the same. On the contrary, human users are very variable in terms of temperament and abilities, which is reflected in their computer-related behavior [727, 181]. Moreover, they may sometimes behave in unpredictable ways [211]. This just adds to the natural variability of the workload.

9.1.1 Sessions and Job Arrivals

The patterns of user sessions and the resulting job arrivals form the basis of user-based workload modeling, which was discussed at length in Section 8.3. To recap, its main attributes are as follows:

- Different users create sessions with different characteristics, but the largest group is made up of "normal" users working normal hours.
- Session lengths are typically limited, and the distribution ends abruptly at around 12–15 hours.
- Intersession gaps can be long, and their distribution has a long tail (but not a heavy tail).
- Sessions by different users tend to be synchronized to some extent, because most users observe a daily cycle of activity.

The question is how to model all these attributes together. For example, given session starts, one can model either session ends (thereby inducing a distribution of lengths) or lengths (thereby inducing ends). So which should it be?

The answer seems to be that all three — session starts, session ends, and session lengths — should be combined, reflecting a model of user behavior rather than a model of the statistics of session attributes. It is indeed necessary to model session starts, but they also depend on previous ends. And ends depend, among other things, on lengths. Putting this in the perspective of user behavior, we can suggest that session starts depend on the following:

- The day of the week (more on weekdays, except for gaming where there is more activity on weekends [327]).
- The time of day (mainly at the beginning of the workday, maybe again after lunch).
- The user's profile (some come early, some late, some work nights).
- When the user ended the previous session (did he or she go to sleep late?).

And session ends depend on these factors:

- How long the session has been going on (users get tired).
- The time of day (users tend to leave in the evening).
- The service provided by the system (bad service leads to user abandonment).

It would be hard to create a statistical model that handles all these issues correctly. Yet, when using a generative user-based model one should verify that the induced distributions of session attributes match the empirical data.

Little if any work has actually been done so far on modeling all these aspects of user behavior. One issue that has been modeled, however, is user reaction to system performance [615, 616]. One question asks exactly what metric of performance really affects user behavior — for example, does response time capture performance as it is perceived by human users, or should we use slowdown instead? As shown toward the end of Section 8.4, response time is a much better predictor of subsequent user behavior than slowdown.

The next question asks how exactly does the response time of submitted jobs affect the user's decision to terminate a session. To answer this, define session breaks as intervals of more than 20 minutes between the end of one job and the submittal of the

next job. Then calculate the probability of a session break as a function of the last job's response time. The result is that the probability of continuing the session drops quickly as the response time grows longer and can be modeled as

$$\Pr(\text{continue}) = \frac{0.8}{0.05\,r + 1}$$

where r is the response time in minutes [616]. Thus the probability of continuing to submit jobs in the same session is 0.8 when response is instantaneous, and it drops to 0.2 if the response time is one hour.

This discussion considers only the activity envelope — when people start to work and when they stop. In many contexts another important issue is the sequence of activities that are performed. In some cases the most important characteristics in determining the sequence are locality and the repetition of previous activities. In others, such as e-commerce systems, it is the actual sequence of actions (e.g., "buy" comes only after "put in cart"). These issues are discussed later in the relevant contexts.

9.1.2 Interactivity and Think Times

When modeling user behavior, interactivity implies a closed system model. But in the complete model we typically have three levels of arrivals, as described in Chapter 8:

1. New user arrivals. This is part of the user population model.
2. Session starts. This is part of the individual user model and may depend on the user type (e.g., normal daytime worker or some other type).
3. Job arrivals within a session.

Importantly, the first two levels are typically modeled using open system models, whereas the third implies a closed system model [614, 601, 552]. The closed model admits two possible behaviors. If all is well, the user continues to work with the system. In this case one needs to model think times as described next. But if the system is congested, the user may abort individual jobs and even the whole session. This was discussed at length in Section 8.4, and appears again in Section 9.1.4.

Interactive user behavior was one of the first phenomena to be studied empirically. Think times of users during interactive sessions were found to be independent but not exponential, and, in particular, to have a longer tail than an exponential distribution. Coffman and Wood, in 1966, suggested a two stage hyper-exponential model [146]:

$$f(x) = 0.615 \cdot 0.030e^{-0.030x} + (1 - 0.615) \cdot 0.148e^{-0.148x}$$

The mean of this distribution is 23.1 seconds. Fuchs and Jackson reported a lower value of 5.0±3.1 seconds [272].

Data from IBM mainframe systems indicated that user think time was correlated with system response time [184, 425]. Thus if the system response time was reduced from, say, 3 seconds to subsecond values, so was the user think time. This is part of the reason why reduced response times improve user productivity.

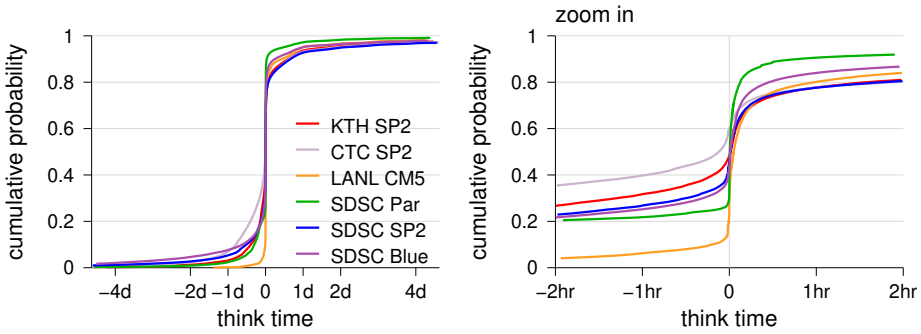


Figure 9.1: *Distribution of think times on parallel supercomputers.*

More recent data from windowing systems, tabulating the number of events generated per second, found an upper limit of about 20–30 events per second (events are either keystrokes or mouse clicks) [594]. This corresponds to a minimal time between events of about 33–50 ms. But this time probably reflects a user's maximal typing speed — which is done more or less automatically — or a system's sampling rate of the mouse, more than it reflects thinking before submitting the next job. Indeed, it is reasonable to place a boundary of about one or two seconds between such automatically spaced events and events that require cognitive action.

In terms of modeling interactive user activity, this distinction means that actually two distributions are needed [272]. One is the distribution of short gaps within sequences of events, such as typing a line of text or performing a sequence of operations with the mouse. Of course, different input devices might actually require different distributions, and HCI researchers have devised precise models of how much time various actions will take [617]. These times can be very short from a human perspective, but still very long for a 2–3 GHz computer. The other is the distribution of actual think times, when the user is contemplating what to do next.

Think times in interactive sessions are by definition positive, because a user cannot start a new task before receiving a response from the previous one. But in batch environments, the interval from the end of one job to the beginning of the next can be negative (that is, the second job is submitted before the first one terminates). The distribution of such think times as observed in parallel supercomputers is shown in Figure 9.1. Between 10% and 60% of the jobs have negative think times — meaning that they were submitted without waiting for the previous job to terminate — with values around 30–40% being typical. This result may be interpreted as showing that jobs are submitted in batches: within each batch the jobs are submitted asynchronously, without waiting for previous jobs to complete, but such synchronous waiting does occur from one batch to the next [614]. Modeling such an activity pattern therefore also requires two distributions: the distribution of intersubmittal gaps within a batch, in which the user does not wait for a previous termination, and the distribution of think times that occur between batches,

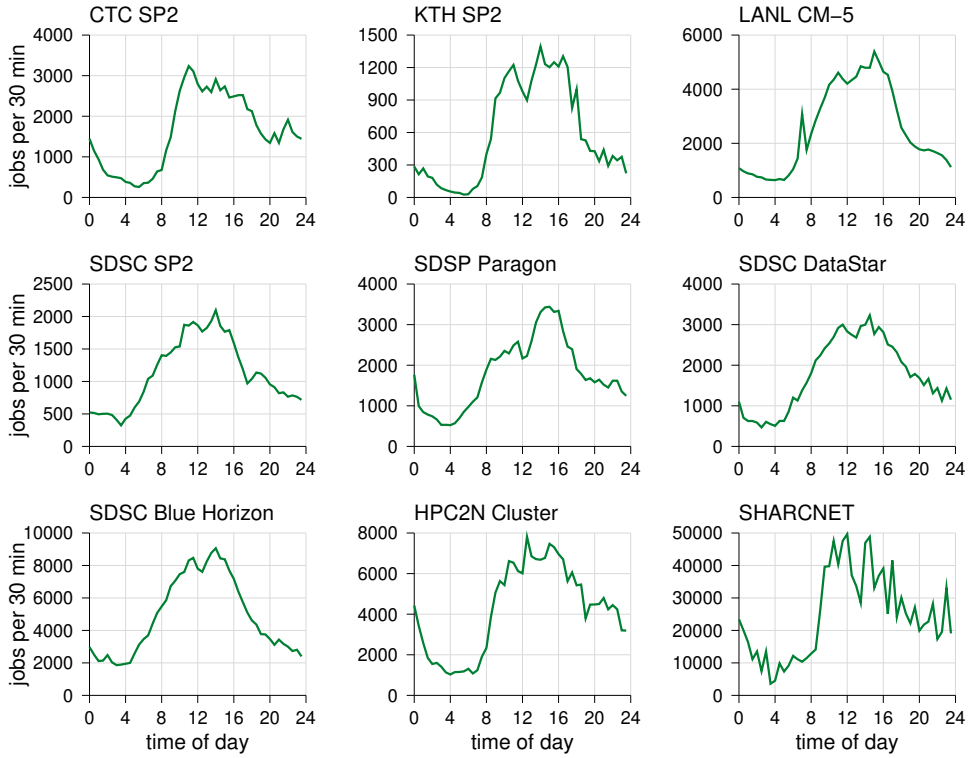


Figure 9.2: *Daily arrival patterns at parallel supercomputers.*

when the user waited for a termination and looked at the results before submitting more jobs (see box on page 399).

Another aspect of interactivity is the typing itself. Fuchs and Jackson provided data from early timesharing systems, indicating that when users send data to the computer the characters are typically sent individually, with intervals of 1.5±0.83 seconds between them (the line speed was such that each character took 0.092 seconds to transmit) [272]. Later data from telnet sessions analyzed by Danzig et al. again showed that characters tend to be transmitted individually — two thirds of all packets contained only one byte of user data [164]. The intervals between such packets were distributed largely in the range of 0.1–1 seconds. These results were corroborated by Paxson and Floyd, who further showed that the distribution of interpacket times (and hence, the distribution of intervals between typing successive characters) is Pareto [540].

9.1.3 Daily Activity Cycle

Another salient characteristic of user behavior is the daily activity cycle (e.g. [327, 321, 525, 63, 763, 747, 758, 382, 192]). Activity patterns typically peak during the day in the morning and afternoon and are low at night. Several examples of data from parallel supercomputers are shown in Figure 9.2. Although the details vary, they all exhibit

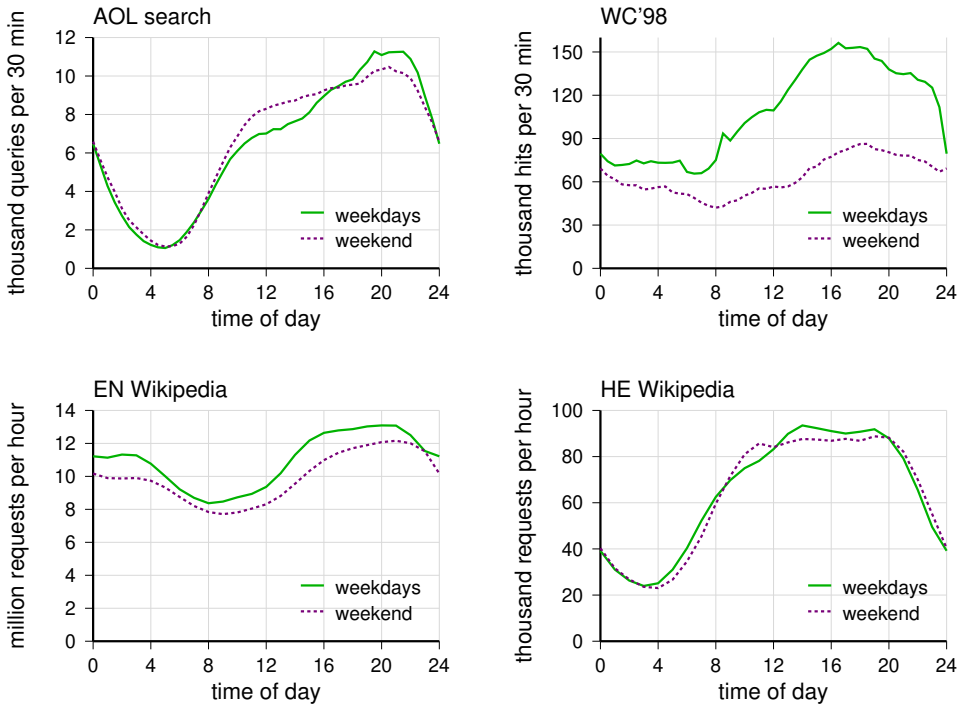


Figure 9.3: *Daily arrival patterns at large websites.*

high levels of activity during normal working hours, possibly with a small dip for lunch. Activity in the evening is much lower, and it reaches a minimum in the pre-morning hours, around 4 or 5 AM. In fact, this is so consistent that it can be used to sort out problems with recording the correct time zone for each log.

Daily cycle patterns at large-scale websites are shown in Figure 9.3. The data from the AOL search log includes 28.7 million queries (including requests for additional pages of results) submitted over a three-month period in 2006 by more than 650,000 users. This shows a different pattern from the parallel supercomputers considered in Figure 9.2: the highest activity occurs in the evening hours, between 7 and 10 PM, when people are free at home. On weekends this difference is much less pronounced, because people are free to search the web also during normal working hours.

The data shown from the World Cup site of 1998 includes nearly 143 million hits during a one-month period before the tournament actually began, in order to avoid flash crowds that occur during games. Although it also displays a daily cycle, it is much less pronounced. The reason is most probably that the daily cycle depends on location. Thus if a website has an international user base, as the World Cup site certainly has, the resulting traffic will be the superposition of several daily cycles with different phases [116]. A similarly weak pattern occurs for the English Wikipedia; the data shown is for the year 2012, probably using Greenwich time. But the Hebrew Wikipedia has a much

more pronounced daily cycle, probably because the vast majority of Hebrew speakers are concentrated within a single time zone.

Calzarossa and Serazzi have proposed a model for the arrival rate of jobs at a workstation [102]. This is a polynomial of degree 8, so as to allow for several peaks and dips in the level of activity. Simpler models have also been proposed (e.g., dividing the day into three periods with low, high, and medium loads [131]). These are reviewed in Section 6.5.1.

9.1.4 Patience

An important aspect of user behavior in interactive systems is patience: how long is the user willing to wait? Traditionally, it has been common to quote about 2 seconds as the limit beyond which users get annoyed with the system (e.g., when waiting for a dial tone on the phone). But, of course, this is not a sharp bound, but a distribution. And it also depends on the system type.

Direct data about patience is scarce. Data from call centers indicates that the time callers are willing to wait for an answer has a hyper-exponential distribution [584]. Shneiderman, in writing about the desired response times of interactive systems, notes that the expected response time varies with the task [617, chap. 10]. Thus when typing or moving the cursor the response must be immediate, around 50–150 ms. Simple frequent tasks should elicit a response within a second, common tasks can take 2–4 seconds, and complex ones 8–12 seconds. Delays above 15 seconds are considered disruptive. Embley and Nagy make similar observations [211], with typical times ranging from less than 100 ms to echo a character, through 2 seconds for an editing task, 4 seconds for a (complex) search, and up to 15 seconds for login or logout. Extending this, a study of peer-to-peer file sharing found that users are willing to wait hours or even days for large media files that are downloaded in the background [309]. At the other extreme, in online gaming users are sensitive to delays as short as 150 ms [81]. Likewise, Quake players (a first-person shooter game) were much more sensitive to interference than users performing word processing or Internet search tasks [313].

Indirect sources of information about user patience include data regarding aborted downloads, and specifically, the distribution of time until the download is aborted. Such data, collected for multimedia downloads [310], indicates a log-uniform distribution starting with 60–70% abortions after a mere second and going up (more or less linearly in log scale) to 95–100% at 100 seconds. Somewhat similar data is provided by a study of live streaming using a P2P system [438]. When the streaming event (a baseball game) began, many users could not immediately obtain sufficient data to play the video and aborted their sessions. Thus the distribution of short session durations, specifically for those sessions that did not have sufficient data, can be used as a proxy for the user patience distribution. This distribution grew rapidly between 10–20 seconds, had a wide mode at about 50–100 seconds, and then a tail that continued beyond 3 minutes.

At a lower level in the protocol stack, one can also identify and study aborted TCP connections [582]. Assuming communications between a web browser and a web server, connections that are aborted by the client (that is, the browser, representing the user) can

be identified by a combination of two criteria: first, that the FIN or RST packet signifying the end of the connection is first sent by the client, and second, that this is done within about one round-trip time from the last data sent by the server. The second criterion is needed to weed out cases where the connection is persistent, and the client closes it after a certain timeout, which does not reflect any impatience by the user. Data collected in this way indicates that most aborts occur within the first 15–20 seconds, and that aborts are much more likely for slow servers (meaning servers that provide only a low bandwidth).

An interesting question is what actually causes users to abort. In the context of networking, it has been suggested that users may be sensitive to two types of performance problems [744]: either the cumulative service rate is too slow, or the instantaneous rate of progress is too slow. These models assume the user is cognizant of the transfer's progress, and not only of the elapsed time. For example, a bar that increases in length showing what fraction of the transfer has been completed may cause users to abort if they become impatient and see that only a small fraction has completed. An indication of download rate may cause users to abort if they see that the rate is low and is insufficient for the data being transferred.

Data from parallel supercomputer workloads in the interactive studies cited earlier also indicates that users are willing to wait more than a few seconds. This analysis looked at the think times (intervals until the user submitted another job) as a function of the previous job's response time [615]. Response times of a few minutes led to think times that were also only a few minutes long, indicating the user was still waiting for the result. With much longer response times, the probability of a short think time dropped, but not to zero.

A completely different type of data is provided by web search logs. An analysis of the number of results viewed shows that most users only skim the first page of results, but the distribution is heavy-tailed with some (very few) users looking at hundreds or even thousands of results (see Section 9.4.6 and Figure 9.29). However, this finding may be due to robot activity.

In a related vein, Chambers et al. look at gaming servers and define patience by the number of times a user is willing to try to connect to a busy server [116]. They find that three out of four users are unwilling to try again (but this may be tainted by software that automatically tries a different server each time). Of those who do try again and again, the number of attempts is geometrically distributed.

9.1.5 Mobility

With the advent of Wi-Fi technology it has become possible to connect to the Internet in a wireless ad-hoc manner. The movement of human users then affects the characteristics of the workloads they generate, because they may roam from one system to another during an ongoing activity.

The degree of user roaming between access points may depend on user context. For example, Kotz and co-workers analyzed the wireless network installed at Dartmouth College in its early days; they found that most students did not roam much and that most

of their connections were made in residences [414, 328]. Physical infrastructure also has an effect: if there are many overlapping access points, network cards may reassociate more often while trying to maintain a connection to the strongest signal available. This creates many artificially short sessions in place of a single long one that more faithfully represents the user's actual behavior.

The patterns of movement on a campus are as may be expected: they correspond to students' daily schedules and follow the directions of roads and walkways [328, 402]. The speed of movement and pause times follow a lognormal distribution. At the same time, most users actually stay put most of the time, spending most of their time at well-defined hotspots. Typical hotspots for voice over IP users were a hotel restaurant, library, and the school of engineering; for laptops, the most typical location was residential buildings.

Based on these findings, Kim et al. suggest a user mobility model that can generate synthetic tracks that mimic the real ones [402]. This model has the following three components:

1. Users are assigned either a "stationary" or "mobile" character. In the dataset analyzed, 46% were mobile.
2. Stationary users are assigned a stationary hotspot location, a connection start time, and a duration, based on distributions extracted from the data.
3. Mobile users are assigned an initial mobile location and a start time. They then move among mobile hotspots according to the following procedure.
   (a) The next destination is selected based on probabilities from a location transition matrix.
   (b) The number of waypoints to pass on the way is selected from a distribution.
   (c) The waypoints are selected from the rectangle defined by the current location and the destination, and sorted by distance from the current location.
   (d) The speed of movement is chosen from the speed distribution.
   (e) Upon arrival, the user pauses for a duration chosen from the pause time distribution, and then repeats these five steps.

Karamshuk et al. describe human mobility models for opportunistic networks [396]. The idea is to model the spatial, temporal, and social patterns of human activity, based on the realization that human relations are the motivation for individual movements. By predicting movements one can use them to forward messages over disconnected networks. Likewise, Gluhak at al. realize the need for including human behavior in the loop of experimentation on the Internet of things, because the location, mobility, and activity of the "things" are related to humans [288].

9.1.6 Runtime Estimates

A rather particular form of human user behavior occurs when using large-scale supercomputers. Such systems often require their users to provide estimates of job runtimes.

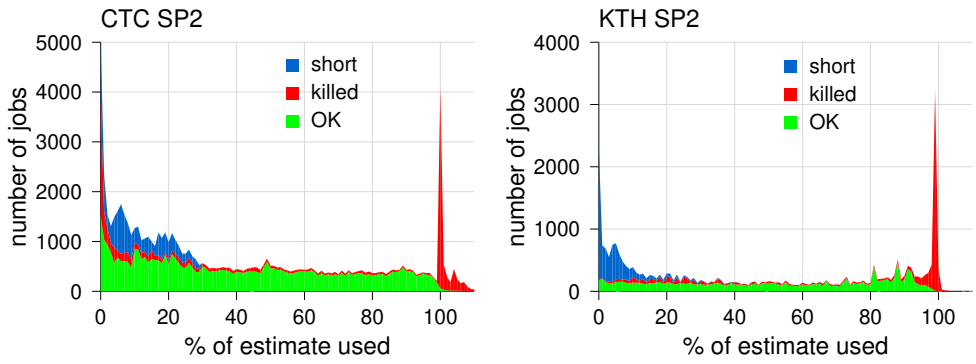


Figure 9.4: *Histograms of how much of the estimated runtime was actually used.*

These estimates are used by the system scheduler to plan ahead and to decide when to run each job [443, 507]. The naive expectation is that users would be motivated to provide accurate estimates in order to facilitate better system performance. The availability of accounting logs allows us to compare the estimates to the actual runtimes. The results show that they are typically far from accurate. But it is especially interesting to consider how they differ.

Figure 9.4 shows histograms of what percentage of the estimated time was actually used. Perfect estimates should lead to a strong peak at 100%, and nothing else. The data shows that a peak at 100% indeed exists, but unfortunately it is composed almost exclusively of jobs that overran their allocation (which was based on the estimate) and were therefore killed by the system. An additional peak occurs at very low percentages; a large fraction of this is either jobs that were also killed for some reason, or very short jobs (here defined to be less than 90 seconds long), which are probably jobs that failed in some way upon startup. The remaining jobs exhibit a rather flat histogram, implying that user runtime estimates do not provide much information about how long jobs will actually run. However, they do provide a usable upper bound: in all the logs, only a small fraction of the jobs were killed because they exceeded their estimates.

To study the effect of the accuracy of user runtime estimates on performance, estimates with varying levels of accuracy are needed. One way to obtain them is to start out with the real job runtimes t_i, and apply some function that creates estimates e_i with the desired accuracy. Given the above observations about the histograms of Figure 9.4, the following four-step model has been suggested [507]:

1. With probability of 10% return $e_i = 0.99 \times t_i$. This represents the jobs whose estimates were too low and were therefore killed by the system.
2. Otherwise create an estimate of $e_i = t_i/u$, where u is uniform in the range $(0, 1]$. This leads to the flat histogram where the ratio t_i/e_i is uniformly distributed.
3. If $t_i < 90$, multiply the estimate by 10. This makes short jobs use only a small part of their estimate.

4. If the estimate is outrageous, truncate it to some upper bound (e.g. 24 hours).

The problem with this model is that it actually provides rather good estimates that do not reflect reality: a quarter of them are off by only 33% or less, and another quarter by less than a factor of 2. Thus short jobs of a few minutes can typically be easily distinguished from long jobs of several hours. Another problem is its randomization, and in particular, the fact that there is no correlation between runtime and accuracy. Such a correlation actually does exist, for two reasons [694]. First, long jobs generally enjoy much more accurate estimates. The reason is that all large-scale systems impose limits on job length. Naturally, these limits also apply to the estimates. Thus the closer the runtime is to the limit, the smaller the range that the estimate can come from. For jobs that are practically at the limit, the estimate must also be at the limit, and hence be very accurate.

The second reason is that user estimates are inherently modal. Users are human, and humans simply tend to use round numbers. It is very rare to see estimates such as 37:21 minutes — normal users will tend to round up this time. Thus it was found that only 15 discrete values account for around 80% of the estimates in all the logs. These values are 5, 10, 15, 20, and 30 minutes, and then 1, 2, 3, 4, 5, 6, 8, 10, 12, and 18 hours. Because these estimate values are so popular, jobs whose real runtimes are near one of them tend to have more accurate estimates, whereas jobs whose runtime is in between them will have inaccurate estimates.

Based on these observations, a better model has been proposed to generate synthetic runtime estimates that mimic those chosen by real users [694]. This is somewhat involved, but the five main components are as follows:

1. Start with the 20 most popular estimate values. These include the maximum allowed runtime and other "round" values as listed above.
2. The number of additional estimate values to use is based on the number of jobs for which we need estimates: the more jobs, the larger the number of different values that will eventually be used. The values themselves are calculated from a formula.
3. Next, define the sizes of the different modes (i.e., how many jobs will use the most popular estimate, how many the second most popular estimate, and so on). The model is to use an exponential distribution for the first 20 and a power law for the rest.
4. Then we need to decide the mapping: which estimate value (from steps 1 and 2) is used by how many jobs (from step 3). The maximal allowed estimate is made the most popular. The remaining 19 popular values are mapped to large modes in an order that resembles the data from logs. The other values are mapped randomly.
5. Finally, we need to assign a specific estimate to each job such that the correct number of jobs will end up with each estimate value. To achieve this we create a sorted vector of all the required estimates. Thus if an estimate of, say, one hour has a popularity of 3%, and we intend to create a workload with 100,000 jobs, then we will have 3,000 instances of one hour in the vector of estimates. In parallel,

we create a sorted vector of the actual (modeled) runtimes of all the jobs. Now iterate over the runtimes from the highest to the lowest, and for each one select an estimate that is larger or equal to it.

A deficiency of this model is that it does not include locality. In real workloads successive jobs tend to be similar to each other. In the current context, this means that they will typically have the same runtime estimate, rather than having random estimates (albeit from a modal distribution).

Although the particular issue of user runtime estimates is perhaps of limited interest, its analysis does identify human behaviors such as the tendency to use rounded values. These behaviors may be expected to occur in other cases where user input is sought.

To read more: In parallel job scheduling, the discovery that user runtime estimates are inaccurate led to another unexpected discovery: it seemed that inaccurate estimates actually lead to better performance! This led to a line of interesting experimental work that eventually showed that inaccurate estimates cause holes to be left in the schedule, and these can then be filled with short jobs, effectively leading to an implicit shortest-job-first effect that improves the average response time. Thus the truth of the matter is that accurate estimates are actually better, because they facilitate using shortest-job-first explicitly [242, 507, 765, 697, 695]. The original study was also tainted by using "statistically bad" estimates that, as noted earlier, still contain significant information. A better model for bad estimates is that more of them use popular modal values and thus lose information [694].

9.2 Desktop and Workstation Workloads

Desktop systems and workstations were among the most commonly used computing platforms for many years, only recently being replaced by mobile devices such as tablets. The arrival of jobs in such systems is governed by user behavior, which was covered in the previous section. Here we focus on the characteristics of the jobs themselves, which are plausibly the same or similar also on mobile devices.

An important aspect of desktop workloads is the total load they place on the system. In fact, many studies show that typical desktop workloads leave the vast majority of the resources idle most of the time. For example, reports regarding average system idle time range from 75% [510] to 91% [418]. On desktop machines, which are often networked together, the spare cycles on one machine may be used by applications initiated on another. load balancingSuch behavior lies at the base of load-balancing schemes [51, 320], grid systems [30], and large-scale volunteer computing efforts such as SETI@home [26]. Alternatively, the findings regarding idle time provide strong motivation for power-saving schemes, both on desktops and even more so on mobiles. A particular challenge is to achieve such savings without affecting interactive performance. At the same time, evolving modern workloads are using more and more resources, including auxiliary processing power such as the GPU and memory space in multilevel caches.

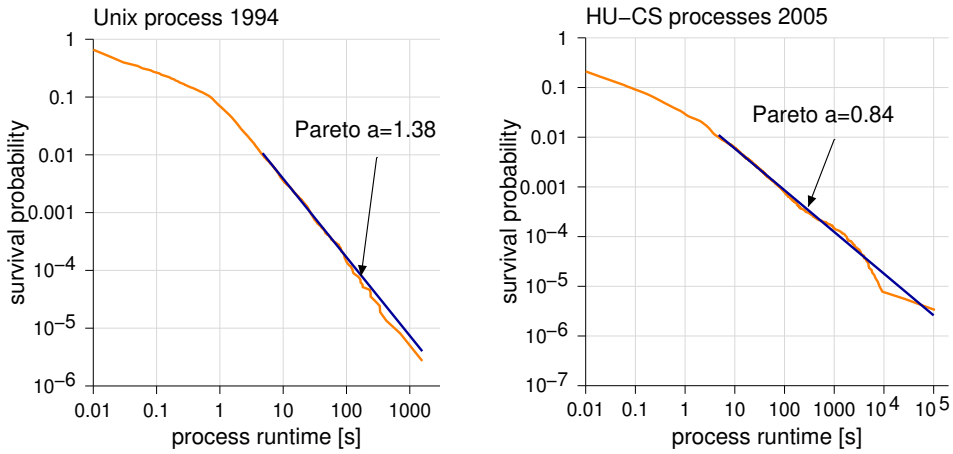


Figure 9.5: *LLCD plots showing heavy-tailed process runtimes.*

9.2.1 Process Runtimes

Maybe the most basic characteristic of a process is how long it runs. Early measurements in the 1960s quickly revealed that the distribution of runtimes is skewed and has a coefficient of variation greater than 1. This led to the proposal that hyper-exponential models be used [580, 146, 717].

More recent work on the tail of the distribution has identified it as being heavy, and providing a good fit to a Pareto distribution [435, 320], As illustrated in Figure 9.5. These graphs show that the probability that a process run for more than τ time decays according to a power law:

$$\Pr(T > \tau) \propto \tau^{-\alpha} \qquad \alpha \approx 1$$

This means that most processes are short, but a small number are very long.

The finding that process runtimes are Pareto distributed (or at least, their tail is) has important implications for load balancing (when computations can be performed on any of a set of machines). Load balancing has to contend with three main questions:

1. The first question is whether there is enough of a load imbalance to justify active migration of processes to improve the balance. In particular, an underloaded machine and an overloaded machine need to be paired up, which requires machines to know about each other [52, 496].
2. Migrating a process from one machine to another costs considerable overhead to transfer the process state (typically its address space and possibly some system state as well). The question is whether the expected performance gain justifies this expense.
3. Migrating a process that terminates soon after is especially wasteful. This raises the question of selecting the best process to migrate.

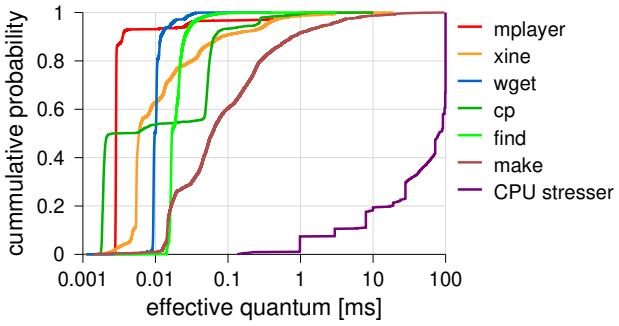


Figure 9.6: *Distributions of the effective quanta of different applications, when run on a Linux system with a quantum time of 100 ms (from [219]).*

The distribution of runtimes enables us to answer the latter two questions. Specifically, if we condition the probability that a process will run for more than τ time on how much it has already run, we find that a process that has already run for t time may be expected to run for an additional t time: the expectation actually grows with how long the process has already run! (The derivation is given in Section 5.2.1.) This makes the long-running processes easy to identify: they are the ones that have run the longest so far. And by knowing their expected remaining runtime and the load on the two machines, we can also calculate whether the migration will lead to improved performance or not [320, 435].

Studies such as the above require data about process runtimes. On Unix systems this information is typically obtained using the lastcomm command. The lastcomm command reads its data from system log files, and prints it out in a format that is more convenient for humans to read. Regrettably, in the process it reduces the second resolution of job termination times to a resolution of minutes. To retain the original resolution, one needs to write a program that will parse the log file directly.

Note that the process runtime refers to the total time required by a process (indeed, this is sometimes referred to as the process lifetime for added emphasis). This total time is accumulated in numerous short bursts of CPU activity, interspersed by system activity (handling interrupts or system calls) and possibly the execution of other processes. Obtaining data about the length of CPU bursts requires precise instrumentation, using tools such as KLogger [219]. Using such tools, it is found that typical bursts are very short, and in particular they are much shorter than the operating system's time quantum (Figure 9.6) — so much so as to make the operating system quantum irrelevant. This is so because desktop programs are typically interactive and perform many I/O operations, with only a fraction of a second of computation between successive I/O operations.

The only case where the operating system quantum limits the time that a process will run is if the process performs pure CPU work, and even then about half of the quanta are cut short by interrupts or system daemons activity. (Daemons are system processes that are activated periodically to handle various administrative tasks.) However, when a pure

CPU process (CPU stresser in Figure 9.6) runs concurrently with other applications, its effective quanta will often be reduced to the range of those of the other applications. For example, when running concurrently with a movie viewer such as MPlayer or Xine, the movie viewer will require the CPU for a short time whenever a new frame is to be shown. If the frame rate is 25 fps, this means that it will run every 40 ms (assuming the system supports alarms at such a resolution). As a result all other applications (such as the CPU stresser) will never run more than 40 ms before being interrupted.

9.2.2 Application Behavior

Application behavior may be characterized in several different dimensions. This section focuses on those that are the most relevant for the study of a machine's microarchitecture and memory hierarchy, namely the instruction mix, branching behavior, memory behavior, and the division into phases of computation.

Instruction Mix

All modern microprocessors have multiple execution units of different functional types. To achieve good utilization, the mix of execution units should match the mix of instructions in applications. Thus a very basic workload attribute for architecture studies is the instruction mix.

As a simple example, consider processors that support the FMA instruction with a specialized functional unit. This instruction performs a floating-point multiply-add, which is a basic component in many linear algebra computations. It contributes significantly to the processor's peak rating, because each completed instruction is counted as two floating-point operations. However, if an application does not contain multiplies and adds with a suitable relationship between them, this functional unit cannot be used and the achievable computation rate is reduced. Similar problems may occur with SIMD-like instructions for multimedia processing [673].

The FMA is a unique instruction that does not exist on every processor. In general, one of the key differences between computer architectures is the instructions they provide. Thus the instruction mix exhibited by a program actually depends on the architecture for which it was compiled. To avoid this issue it is common to consider classes of architecture-independent generic instruction types rather than specific instructions [345, 387]. These may include the following:

- Integer arithmetic (addition, subtraction, and logic operations).
- Integer multiply.
- Integer division.
- Floating-point arithmetic (addition and subtraction).
- Floating-point multiply.
- Floating-point division.
- Load (from memory to a register).

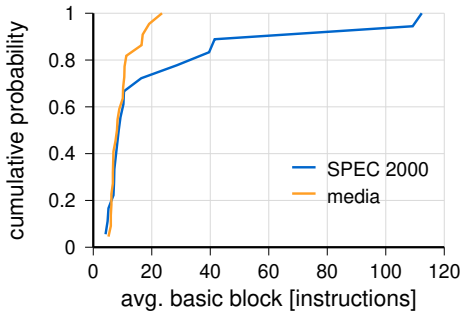


Figure 9.7: *Distribution of basic block sizes in two benchmark suites (data from [387]).*

- Store (from a register to memory).
- Branch instructions.

Sometimes fewer classes are used, because some of these classes can be grouped together.

The most popular way to obtain a representative instruction mix is by using a benchmark suite, that is, a set of real applications that are considered representative of applications in a certain domain. The most popular benchmark suite for general-purpose computing is SPEC CPU from the Systems Performance Evaluation Consortium [655]. Note that program behavior also depends on input, so representative inputs are also needed [202, 204], and indeed, benchmarks specify not only the applications to run but also the reference inputs to use. Benchmarks are discussed further in Section 9.2.4.

An alternative is to create a model of program behavior suitable for architecture studies, including the instruction mix, number of operands, and register aging [200]. A related approach is to synthesize test cases that approximate the behavior of the full benchmarks, and thereby also their performance characteristics [64, 387]. In either case, it is important to note that we are interested in modeling the correct dynamic instruction counts, not the static ones. Static instruction counts are the number of times each instruction appears in the compiled code. Dynamic counts are the number of times each instruction is executed when the program runs and may depend on the piece of code that is considered. Joshi et al. suggest maintaining separate instruction counts for each basic block of the code, and moreover, to maintain separate counts depending on the identity of the previous basic block to run [387].

Branching Behavior

The branching behavior of applications includes information about several program attributes: the sizes of basic blocks between branch points, the graph of the flow from one basic block to the next, and the probabilities of taking the different branches.

The distributions of basic block sizes in two benchmark suites are shown in Figure 9.7. The distributions show the average sizes for the different benchmarks in each suite.

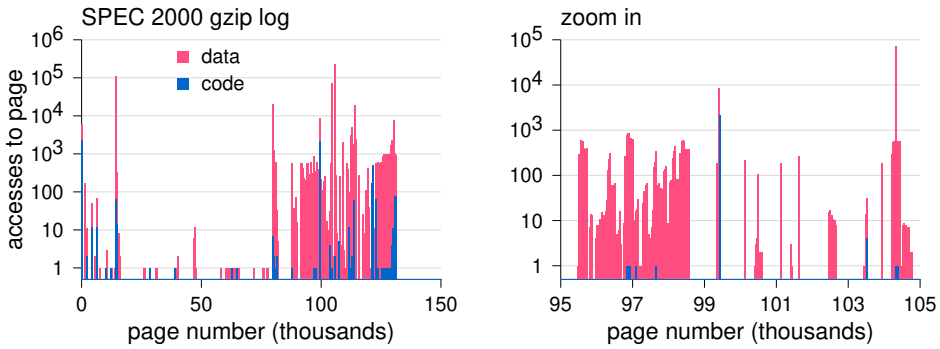


Figure 9.8: *Distribution of accesses across the address space.*

This shows that typical average basic block sizes are small, ranging from 4 to 11 instructions. However, some basic blocks may be much bigger, and in some benchmarks even the average is much bigger — as high as 112 in the applu benchmark and 109 in the mgrid benchmark from the SPEC CPU 2000 suite [387].

The connectivity of basic blocks shows less variability (at least for these two benchmark suites). Specifically, the average number of successor basic blocks ranges from 2 to 11 in all the benchmarks studied [387].

The probability of taking a branch is typically far from $\frac{1}{2}$, which lies at the basis of all branch prediction schemes [637]. In some cases, prediction can be done statically. For example, the branch that controls a loop is typically taken, because loops are typically executed more than once. Thus branch instructions can be modeled by their taken rate, and either high or low rates lead to good predictability.

However, in some cases the taken rate does not provide the full picture. For example, a branch that is alternately taken and not taken is completely predictable, whereas one that is randomly taken with the same probability of 50% is not. Therefore the taken rate should be complemented by the transition rate, which quantifies the probability that the branch transitions from being taken to not being taken [322]. Branches with either very low or very high transition rates are highly predictable, regardless of their taken rate. Branches with high or low taken rates are just a special case, because an extreme taken rate necessarily implies a low transition rate.

In addition, dynamic information may also add to predictability. An interesting observation is that branches may be correlated with each other [223] (e.g., if different branches actually include the same predicates). Such cross-branch correlation should also be included in models to enable branch prediction schemes to be evaluated.

Memory Behavior

Computer programs do not typically use their whole address space. Rather, only distinct regions or segments are used, and different regions are used different numbers of times.

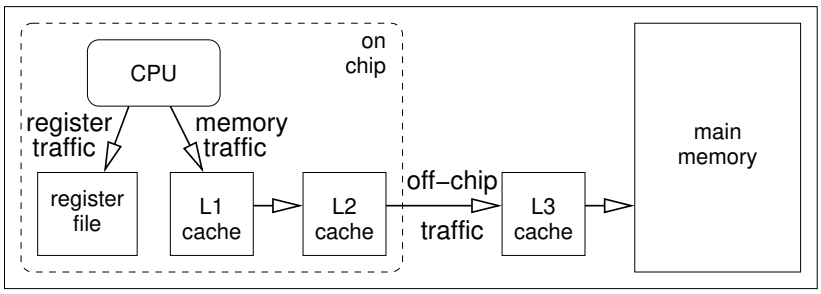


Figure 9.9: *Data accesses by a system's CPU lead to different types of memory workloads depending on the target.*

This is illustrated in Figure 9.8 for the gzip benchmark from the SPEC 2000 suite, using 4 KB pages as the basic unit of memory[1].

The main attribute of memory references is their locality. This has central importance in the context of paging schemes [171, 174, 648] and caching [306, 476, 743], because locality enables the high cost of fetching a memory page or line to be amortized across many references.

Locality is the result of two phenomena that may occur in a reference stream. The first is that references to a given set of locations tend to be grouped together, rather than being spread uniformly across the whole execution of the application. This is related to the issue of program phases discussed later. The other is skewed popularity, in which some memory locations are much more popular than others, implying that they are reused much more often [381, 217, 411] (recall Figure 9.8).

The degree of locality observed in a reference stream depends on the locus of observation (Figure 9.9): the reference stream emitted from the CPU may have significant locality, but after it is filtered by successive levels of caches the degree of locality may be reduced. This affects the use of traces to study memory behavior. The problem is that traffic between the processor and the L1 cache cannot be observed. Therefore, to trace the true memory behavior of an application, caches should be disabled. When this is done the external memory traffic exhibits the full locality of the application. But such data is actually only useful for caching studies. If one is studying bus and memory performance, one should use a reference stream that has indeed been filtered by the caches [730]. However, this choice is somewhat ill defined, because the degree and character of the filtering depend on the cache configuration.

One of the simplest and most common ways to characterize locality is using the LRU stack distance [649, 648]. Examples of the stack distance distributions of several SPEC 2000 benchmarks are shown in Figure 9.10. Stack distance distributions can also be used to create a generative model of memory references. The idea is to prime the stack with all the memory addresses, and then generate successive references by simply

[1]This data is from the Brigham-Young trace repository. Surprisingly, it indicates that instruction fetches and memory reads and writes may be targeted at the same page. This may be an error because instructions are usually placed in a separate memory segment.

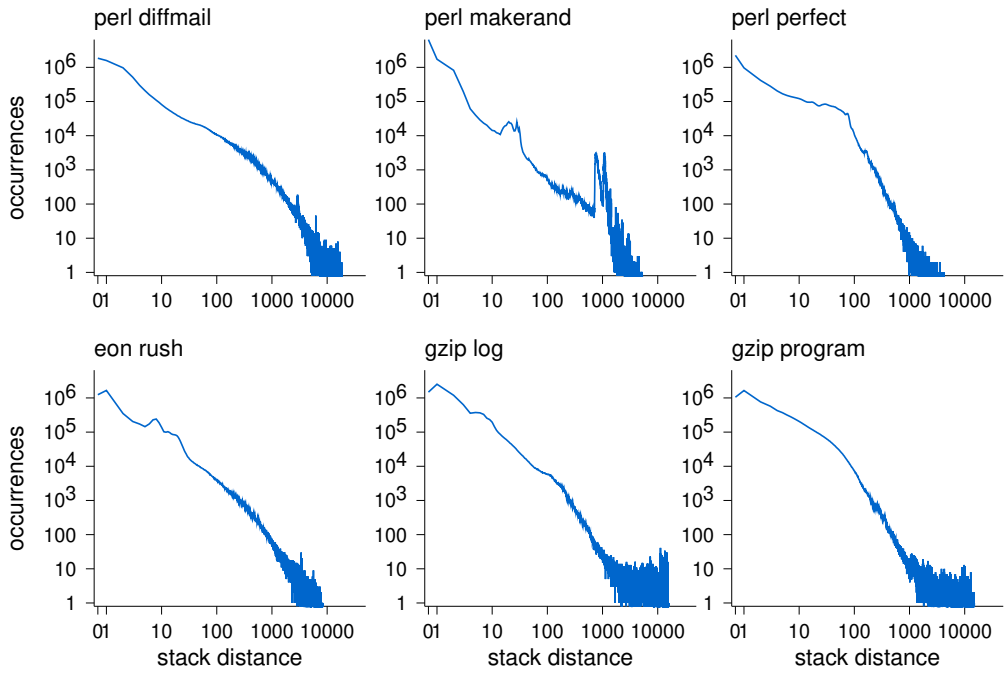


Figure 9.10: *Examples of stack distance distributions from SPEC 2000 benchmarks, using 4 KB pages.*

selecting them from the stack according to the distribution. Of course, when an address is thus selected, it is also moved to the top of the stack to maintain the stack semantics. This and other metrics and models were discussed at length in Section 6.2. For example, the fractal model by Thiébaut et al. [683, 684] tends to produce better locality, because it models the jumps between successive addresses using a heavy-tailed distribution, so most of them are very small. In addition, Phalke and Gopinath suggest a model based on inter-reference gaps for temporal locality [548].

Another way to characterize locality is by using the working set — the set of all unique memory locations that were referenced within a certain window of time (or rather, within a certain number of instructions) [171, 174, 172]. The smaller the working set, the stronger the locality of the reference stream. Although the notion of a set does not imply any order or structure on the memory locations, Ferrari has proposed a generative model that is designed to emulate working sets [257]. There has also been work on combining working sets with skewed popularity, leading to the identification of a core subset of the working set that attracts the majority of references [217, 216, 218].

For example, consider a program fragment that includes two loops traversing two large arrays, as follows:

```
for (i=0; i<lenA; i++) {
    A[i] = 13;
}
```

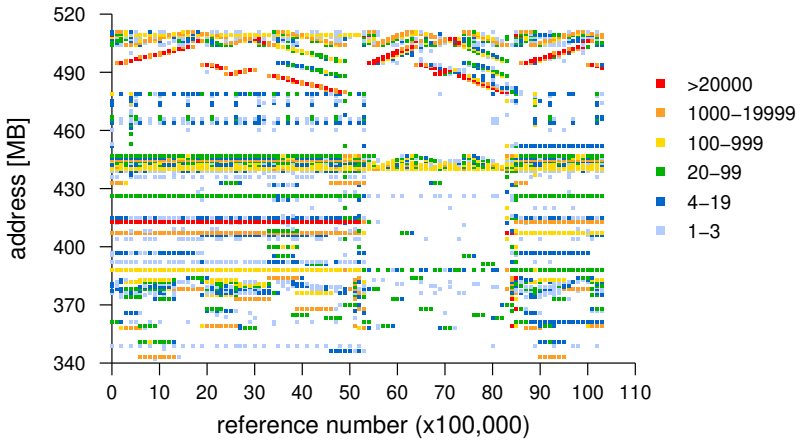


Figure 9.11: *Memory accesses map of the gzip SPEC 2000 benchmark with the log input. Note phase transitions, most notably around 5.4 and 8.3 million references, and diagonal lines signifying sequential traversal of addresses in the range of 480–500 MB. Colors represent number of accesses to each block of 1 MB within a window of 100,000 references.*

```
for (i=0; i<lenB; i++) {
    B[i] = 0;
}
```

In the first loop, i and lenA are reused repeatedly, and the elements of A are accessed once each in sequence. In the second, i and lenB are reused, but this time the elements of B are accessed once each. Now consider the size of the working set that would be found. Obviously, since the vast majority of the unique memory locations referenced are the elements of A and B, the size of the working set will be directly proportional to the window size! This will only change if the window includes much more than these two loops, and then it would depend on the additional code too.

The working set model envisions a slow shift of locality. But real data tends to be more structured than random sampling from such a set (Figure 9.11). A comprehensive model should therefore include phenomena such as the following:

- Distinct phase transitions where the access pattern changes abruptly [152, 172, 543].
- Sequential access to successive addresses, as would be generated when traversing large data structures [152, 287].
- Strided access with various stride sizes, as would be generated when traversing multidimensional arrays [387].

In particular, to get a better handle on locality properties, it is best to first partition the execution into distinct phases, and to consider each phase independently [61, 456]. This

leads to a model based on a combination of macro models and micro-models [152, 172, 543]. The macro model describes how the program moves from one locality to another, typically using a Markov chain formalism. The micro-models describe the references performed within each locality. Courtois and Vantilborgh explain how the characteristic working set sizes of an application result from the working set sizes in the different phases, weighted by the limiting probabilities of being in the different phases [152]. Peris et al. extend this by considering overlaps between localities, and also show how to factor in the state that remains in the cache when a phase is repeated [543].

The main implication of recognizing program phases is the realization that most page faults (and cache misses) happen during phase transitions [172]. For example, one study showed that phases occupy 98% of a program's runtime and transitions only 2%, but the transitions account for some 50% of the page faults. As a result modeling multiple contending applications can make do with only two classes: jobs that are in a phase and jobs that are in transition between phases. Such a model indicates that a good multiprogramming level is such that two or more jobs are rarely in transition at the same time.

Nevertheless, it is also important to consider the internal structure of individual phases. Phases may actually be nested, with a major phase of the computation including several subphases, which may also be repeated [456, 411]. In addition, one must distinguish between the *reuse set*, which is those elements that are indeed reused extensively (i, lenA, and lenB in the earlier example), and the other elements, which are transient in nature [216, 411]. The reuse set provides a better approximation of the actual memory capacity needed by the application, because the memory used to store transient elements can be reused.

The working set and reuse set only capture the volume of data that is used by an application. Another important characteristic is the *rate* at which data is accessed. For example, consider two applications running on two cores and sharing a cache. If one of the applications accesses its data at a higher rate, it will tend to appropriate much of the cache space, because the slower application's data will typically be selected for eviction. Thus, to enable modeling of the contention between applications that share memory resources, Koller at al. suggest the following three additional model parameters [411]:

1. The *flood rate*: the rate at which an application floods the cache with memory lines that are not part of its reuse set. This reflects the application's compulsory miss rate, and the pressure it applies on cache space allocated to competing applications.
2. The *reuse rate*: the rate at which an application accesses data in its reuse set. If the reuse rate of one application is lower than the flood rate of another, its core data will tend to be evicted from the cache, leading to poor performance for this application.
3. The *wastage*: the space required to store nonreusable data and prevent the eviction of reused data.

An often overlooked aspect of memory behavior is that part of the data is not stored in memory at all, but in registers (Figure 9.9). The problem is that the usage of registers is governed by the compiler, so the workload may be affected not only by the application being run but also by the compiler that was used and the level of optimization. Register workloads are important for architecture studies, especially for issues such as register renaming and instruction dependencies. For example, one property uncovered by studying register workloads is that register instance lifetimes are heavy-tailed [198].

Program Phases

Many applications are composed of multiple phases, such that the application behaves consistently within each phase, but the behavior in one phase is quite different from the behavior in other phases. Another common structure is one of repetitive phases, in which a certain activity pattern is repeated multiple times.

The importance of phases stems from the fact that, if we identify them, we can evaluate a representative part of each phase, rather than evaluating the entire program [422, 613]. This can save considerable effort. Another reason to be interested in phases is that the phase transitions themselves may affect the workload. For example, a phase transition may lead to increased paging activity as a new working set of pages is loaded into memory as described earlier [172, 543].

Identification of program phases based on memory referencing behavior was discussed by Batson and Madison [61, 456]. In a nutshell, their procedure was based on identifying periods with consistent memory referencing, meaning that the same locality was referenced repeatedly. In addition, they required that such periods be long enough, to prevent the identification of short transient periods as phases. An alternative is to identify phases, or rather shifts in program behavior that indicate a phase transition, based on reuse distances. The reuse distance is the number of different references made between two successive references to the same address. Short reuse distances are used to characterize locality. But large reuse distances can be used to identify a return to an activity pattern that occurred a long time in the past (e.g., in a previous phase) [182].

These procedures detect phase transitions in a trace that records program execution. Another method to detect phase transition points in the application itself is to "feed" the application synthetic data that is very regular. If the input contains many identical repetitions, the program's behavior will likewise contain many identical repetitions. This enables the instructions that mark phase transitions to be identified automatically even in the binary executable, without access to the source code [611]. They can then be used to delineate phase transitions when the program is applied to real inputs.

Parallelism

An important aspect of modern applications is parallelism. Parallelism allows applications to accelerate their computation by using multiple processors, cores, or hardware contexts provided by the platform. However, this comes at the price of contention for

shared resources such as L2 or L3 caches. Parallelism therefore rarely provides linear speedup with the number of cores being used.

For many years desktop applications provided little if any parallelism [43]. But when parallelism is in fact used, it is important for applications to complement each other's resource requirements. Thus the system scheduler is required to foster an appropriate job mix [228, 762, 224]. This mix is based on assessing the intensity with which each application uses the different resources, coupled with its sensitivity to competition from the other applications.

Because there are tremendous numbers of desktop applications, evaluations typically depend on representative benchmarks. One such benchmark is the PARSEC suite (standing for Princeton application repository for shared-memory computers, but actually representing a collaboration with Intel) [75]. This suite is a set of 12 applications, ranging from image processing, animation, and video decoding through data mining to portfolio analysis. Importantly, these applications use advanced algorithms and techniques that represent emerging workloads and challenge contemporary architectures. Each of the benchmark programs comes with six input sets, including small inputs for use in development and testing, and larger inputs with different sizes for actual evaluations. Together, the suite attempts to cover diverse conditions, including the following:

- Different levels of using parallel programming primitives such as locks, barriers, and condition variables.
- Different levels of inherently sequential work that limits the achievable speedup.
- Different sizes of data working sets and different degrees of locality in accessing the working sets.
- Different levels of sharing cache lines by threads running on different cores.
- Different amounts of off-chip traffic to memory.

9.2.3 Multimedia Applications and Games

Multimedia applications are an important type of applications that are common in both mobile and desktop settings. Their prevalence and unique requirements even affect processor design [179], e.g., the addition of SIMD extensions to the instruction set by Intel and other manufacturers [541, 542]; other architectural enhancements have also been proposed [673].

Indeed, when comparing multimedia workloads with general-purpose workloads several differences become apparent [199]:

- Multimedia programs are more computationally intensive. This means that the ratio of computation to memory operations is higher.
- Multimedia programs have a higher degree of locality. In particular, the data exhibits higher spatial locality due to streaming, which leads to high predictability. It may also exhibit higher temporal locality caused by working on small parts of the input. Likewise, the instructions also exhibit both higher spatial locality and higher temporal locality.

- Multimedia applications tend to suffer from hard-to-predict branches.

Because of their unique characteristics, special benchmarks have been devised for multimedia workloads. Two such benchmarks are the Berkeley multimedia workload [633] and MediaBench [430].

Games are a special case of multimedia applications in which graphical output is combined with interactive user input. This input is typically quite intensive and continuous, so issues such as think time and patience discussed earlier become irrelevant. The workload imposed on the system can be studied at three levels:

1. Player behavior: the dynamics and characteristics of using the mouse, keyboard, joystick, etc.
2. Programming API: the sequence of graphic requests issued by the application. The number of requests per frame gives an indication of the scene complexity.
3. Hardware: the instructions, branching, and data used by the CPU and GPU to perform the actual computations, as well as bus usage and I/O activity.

The latter two are specific to graphics rendering and the specialized hardware that has been developed to support this activity [493, 574]. As a result, specialized benchmarks have been developed for this application area. One of the most commonly used is 3DMark05 [620].

9.2.4 Benchmark Suites vs. Workload Models

The evaluation of desktop systems, workstations, and servers is dominated by benchmark suites. The best-known benchmarks for computer systems are those defined by the Standard Performance Evaluation Corporation (SPEC) [655, 329]. In the context of microarchitectural studies, the relevant benchmark is SPEC CPU. This is actually a suite of complete applications, each with one or more representative inputs. There are also other benchmark suites, such as the MiBench suite, which is targeted for embedded systems [316], and MediaBench, which is targeted for multimedia systems [430].

The reason for using a whole suite of benchmarks is that microarchitectural workloads are very complex and hard to characterize. Achieving a representative instruction mix is not enough — we also want representative branching to enable an evaluation of the branch prediction unit, representative memory behavior for the analysis of data and instruction cache miss rates, and correct dependencies so as to be able to asses the achievable instruction-level parallelism. Moreover, all of these issues interact with each other. For example, the behavior of the branch prediction unit may affect the reference stream as seen by the memory unit. Instead of trying to model all this, researchers tend to simply use real applications — and because applications are actually different from each other, they use many of them. The hope is that by using enough long-running applications all the important workload attributes will be captured with sufficient fidelity.

The fact that it is hard to achieve representative microarchitectural behavior leads to a real tension [745]. On one hand, confidence in the evaluation depends on running many long applications. But when simulating a new architecture this process can take weeks

or even months for a single configuration, because cycle-accurate simulations are orders of magnitude slower than direct execution. Trying to compare several configurations then becomes practically impossible. This has led to devising different ways to reduce the amount of simulations needed, hopefully without undue impact on the accuracy of the results.

It might be useful to consider the details of such arguments. The SPEC CPU 2006 benchmark suite consists of 30 benchmark programs: 12 integer benchmarks and 18 floating-point benchmarks. Together, these programs encompass 3.3 million lines of code. Each has at least one, and often several, reference inputs used for actual evaluations. In addition there are smaller test inputs to verify functionality, as well as train inputs for use in feedback-directed optimization. When executed, the dynamic instruction counts are in the billions, with tens of billions being typical. Running all the benchmarks on all their inputs involves the execution of a trillion instructions, give or take an order of magnitude. When all this is simulated rather than being run on actual hardware, the cost grows considerably. However, the end result of all this work is often condensed into a single number, say the achieved instructions per cycle (IPC) averaged over all the benchmarks. Couldn't this result be obtained with considerably less effort? And to what degree is it meaningful at all?

The simplest way to reduce the effort is to run just part of each benchmark [745]. Typically, researchers fast-forward, say, 15 billion instructions (to skip any non-representative initialization phase), then run another billion or two as a warmup phase (to get the contents of the memory hierarchy into a representative state), and then collect statistics over the next billion or two.

However, using such arbitrary numbers risks having unknown interactions with the structures of individual benchmarks. A better alternative is to judiciously select which parts of each benchmark to run. The common terminology in this context is to use "representative slices". One approach to do this is to identify program phases explicitly, and to select a representative slice from each phase [422, 613]. Another is to use fractal sampling in order to cover the whole program [380]. Naturally such selections must be done with an eye for their effect on evaluations.

A more systematic way to reduce the work is not to run all the benchmarks with their full reference inputs for each possible configuration. Instead, one can use reduced inputs to quickly find the sweet spots in the design space, and then use the full inputs to evaluate only these select configurations. For example, a specific subsetting for SPEC 2000, called MinneSPEC, achieves this by a combination of using the SPEC test or train input in some cases, and a truncated version of the reference input in others, e.g., for the compression benchmarks [406, 203].

Another interesting study used the MiBench benchmark suite, which has 1000 possible inputs for each benchmark program [316, 126]. Here the problem is not just the size of the inputs, but their number. The study showed that using three inputs at random could bias evaluation results and lead to selecting a substantially suboptimal design point [86]. But by judiciously choosing three representative inputs, one can achieve essentially optimal results. A good way to make this choice is to perform a one-time evaluation of an arbitrary design point with all 1000 inputs, and then select the inputs that gave the

minimal, median, and maximal results. As these three inputs exhibit widely different behaviors they provide good coverage and can then be used to evaluate all other design points effectively.

Yet another approach is to use a subset of the benchmarks, rather than a subset of the inputs. The full set of benchmarks may actually include redundancy that just wastes evaluation time [284, 388]. However, just using an arbitrary subset of the benchmarks is again not a justifiable approach, because we might accidentally select redundant ones and miss important ones [138]. Instead, one can characterize the different benchmarks based on inherent characteristics such as their instruction mix, branching behavior, and working set sizes, cluster them, and use a single representative of each cluster [388]. This may be further improved by using principal component analysis and representing the different benchmarks based on these principal components prior to the clustering [746].

At the same time, there are also claims that benchmarks should actually be extended. Even a large benchmark suite simply cannot cover all the types of applications that exist and are in everyday use. The diversity is simply too great [203]. For example, it has been shown that DOE applications require more memory bandwidth and many more integer operations per FLOP than what is common in the SPEC FP benchmarks [508]. Thus it seems that these benchmarks are not correctly balanced for a large class of HPC applications. Likewise, desktop applications may not be as well represented by SPEC int as we would like [432]. A third example comes from the MiBench benchmark for embedded systems. It has been shown that programs in this benchmark are different from SPEC programs, and moreover, categories within MiBench (automotive, consumer devices, security, networking and telecom, and office automation) are also different from each other [316].

An alternative to benchmarks is to use statistical modeling [201, 197, 64]. This approach is based on creating a statistical model of a benchmark or application, and then using this model to create synthetic workloads that are much shorter than the original. Moreover, the model also includes the effect of the input. If the model is indeed valid, simulating the execution of the short synthetic workload will lead to performance metrics that are close to those that would be achieved by simulating the full benchmark. But the dynamic instruction count, and thereby the simulation time, are reduced by up to five orders of magnitude [387]. In addition, this allows for evaluations based on proprietary applications, without disclosing the actual code of these applications.

To obtain a valid model one needs to profile the benchmark or application. A common starting point is to create a Markov chain model of control flow, where the states are basic clocks and transition probabilities reflect branching behavior. Given that this is a Markov chain, it is easy to calculate the limiting probabilities of the different states, namely what fraction of the time the model will spend in each basic block. The joint ratio of this distribution of probabilities gives an indication of the locality of the code, essentially relating it to the adage that programs spend 90% of their time in 10% of their code.

Each basic block is profiled in terms of instruction mix. For each instruction, the distribution of dependency distances of its operands is recorded. This distribution re-

flects the number of cycles since the input values of this instruction were generated, and is important for modeling dependencies in out-of-order processors. In addition, load instruction addresses are generated based on the desired probability for cache misses, at all cache levels. Common access patterns such as strided access are also taken into account.

9.2.5 Predictability

All the previous program attributes — runtime, branching behavior, memory behavior, and phases — may be predicted to some degree [175]. The same goes for other types of program behavior considered later, such as I/O behavior [289]. This is extremely important for the evaluation of adaptive systems, which attempt to learn about their workload and adjust their behavior accordingly [239].

Predictability comes at several levels. At the more global level is the predictability of the complete application behavior. This stems from repeated execution of the same program on the same or similar input, which is a common type of user behavior. For example, as I worked on this book I repeatedly executed the LaTeX and BibTeX commands, with very small incremental changes to the input files between successive executions. The successive execution profiles in terms of runtime, I/O operations, and memory usage were therefore very similar. Moreover, there was significant regularity in the sequences of distinct applications that were used (e.g. LaTeX, BibTeX, LaTeX, LaTeX). Such sequences can be modeled using Markov chains in which the different states represent different applications — in effect creating a user behavior graph [258, 175, 104].

Considerable predictability also exists within the execution of a single application. The predictability of program behavior results from the simple fact that some code fragments are executed repeatedly. For example, this may be the case with functions that are called many times and with loop bodies that are executed in multiple iterations. Often, these repetitions are very similar to each other in terms of their requirements of the system.

Importantly, the often repeated code may be a relatively small fraction of the total code in the application. Initialization code is only executed at the beginning. Cleanup code is only executed at the end. One may hope that error handling code is not executed at all. Thus the dynamic instruction counts are typically very skewed, and only a relatively small fraction of the instructions need to be monitored in order to identify program phases and make predictions. Thus it is possible to make branch predictions based on the PC (program counter) address of the branch instruction [637]. I/O operations have also been classified based on the PC of the I/O instruction [289].

9.2.6 Operating Systems

So far we have only considered user application behavior in desktop systems. However, the workload in desktop systems also includes significant activity by the operating system, and ignoring this component may considerably skew performance evaluation results. This is the motivation for monitoring the full system, as done in PatchWrx [112], and for full system simulation (or emulation), as done in SimOS [577].

Operating systems have two types of effects on the workload observed in desktop systems. First is the behavior (instruction mix, memory accesses) of the operating system itself. In addition, applications may behave differently when running under different operating systems.

For example, Casmira et al. show that, in a desktop setting (running Windows NT) the work of running the operating system dominates the work of the actual applications being used, and that this differs considerably from the workload as represented by benchmarks such as SPEC CPU [112].

More specifically, studies of memory reference behavior have shown that the cache performance of systems running user applications alone is significantly different from systems running the user applications with interruptions from the operating system [286]. The suggested solution was to use a dual model: an address trace for the user applications and a synthetic model for the operating system. This synthetic model, in turn, has two main components. The first component models the patterns in which the operating system interferes with the user applications. This has two parameters:

- The length of each burst.
- The interval between the beginnings of successive bursts.

The second component generates the references themselves to populate each burst of operating system activity. This is a more complex model that uses the following parameters [553]:

- The probability of accessing the code or the data.
- The probability for a read or a write (for data accesses).
- Parameters for an empirical distribution of distances between successive accesses to the same memory area (i.e., data or code of the application or the operating system).
- The probability of a backward jump (used to invert the sign of the calculated distance).

The equation used to generate the distance between the previously accessed address and the next one is

$$y = \left\lfloor \frac{S(1 - A^x)}{5A^x - 6} \right\rfloor$$

where x is a uniform random variable in the range $(0, 1)$. This leads to jumps similar to those of the fractal model of Thiébaut described in Section 6.2.6. If the memory area contains code, 1 is added to avoid zero jumps (that is, successive accesses to the same address).

To generate realistic memory behavior, the model also incorporates a simple scheduler that selects which user application to run (meaning, which application's trace will be used) after each burst of operating system activity.

9.2.7 Virtualization Workloads

Virtualization allows multiple virtual machines to share a single physical host and is especially prevalent in server consolidation scenarios. Thus the workload involved may be a combination of all those described earlier, and also quite a few of those to be described later.

Benchmarks for virtualization environments need to take this diversity into account, but at the same time limit the complexity of the evaluation. A promising approach seems to be to define a set of servers that represent several generic classes. For example, it has been suggested that a combination of a web server, an email server, and a database application can be used [111, 32].

9.3 File System and Storage Workloads

The workload on file systems has two components. One is the state of the system, embodied by the set of files that it stores. The other is the sequence of requests that applications make to access these files and possibly to modify them. At a lower level, file system operations translate into storage system operations.

To read more: An extensive survey of file system and storage benchmarks, the considerations that shaped them, and their shortcomings has been written by Traeger et al. [687].

9.3.1 The Distribution of File Sizes

The main characteristic of the state of a file system is the distribution of file sizes that it stores. This distribution has three main characteristics. First, it is comb-shaped, with many modes at powers of two, multiples of 8 KB, and (to a lesser degree) multiples of 4 KB (Figure 9.12). Second, there are many very small files of only a few bytes and also many empty files. These are files that don't really store data — they are used to flag some condition (e.g., that a certain server is running) possibly including some minimal data (such as the server's process ID). Third, it is highly skewed, with many small files and few huge files. As a result file sizes exhibit strong mass-count disparity (Figure 9.13). The preponderance of small files may be important for file system design, because it implies that small files may cause significant internal fragmentation, the more so as block sizes increase.

It has been expected that the distribution of file sizes would change over time to reflect the increasing use of large multimedia files. Current evidence does not support this expectation. The distributions from 1993 and 2005 shown in Figure 9.13 are remarkably similar, with just a slight shift to higher values. Other comparisons of the file-size distributions at the same location over a 20-year span (1984 and 2005) and a 4-year span (2000–2004) reached essentially the same conclusion [674, 18].

Of special interest is the tail of the file-size distributions, because the large files from the tail account for the vast majority of disk space needed. In fact, the distribution of file sizes is at the center of the heavy-tails controversy: is it better modeled as being heavy-tailed (that is, with a power-law tail like the Pareto distribution) or as being lognormal

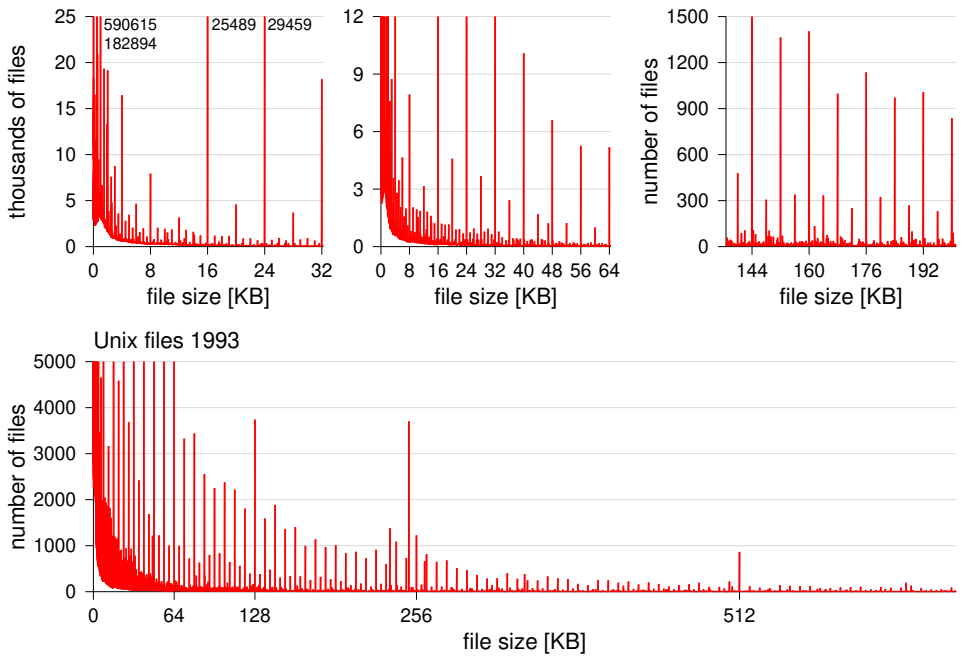


Figure 9.12: *The file-size distribution has a comb structure combined with a strong emphasis on small sizes. Data from the Unix 1993 survey.*

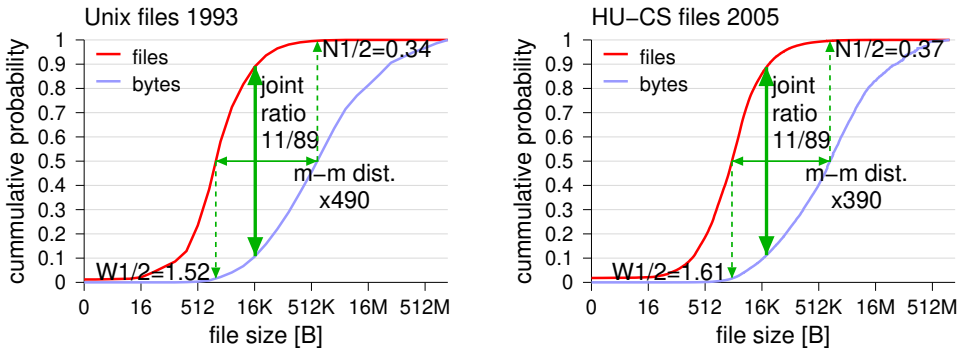


Figure 9.13: *The distribution of file sizes exhibits mass-count disparity.*

[188, 498]? Support for the claim that file sizes are heavy-tailed is provided by graphs such as those shown in Figure 9.14, which demonstrate that the distribution has a power-law tail. However, in some cases the graphs curve downwards slightly, as in the right-hand graph, and may be better modeled by a lognormal distribution. The lognormal also has several additional benefits, including the following [188]:

- It enables one to model the entire distribution, and not only its tail. However, the

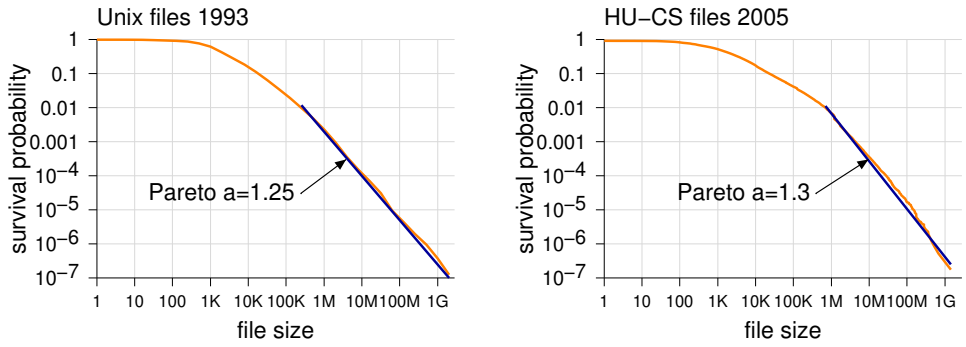


Figure 9.14: *LLCD plots for file-size data provide mixed support for the heavy-tailed model.*

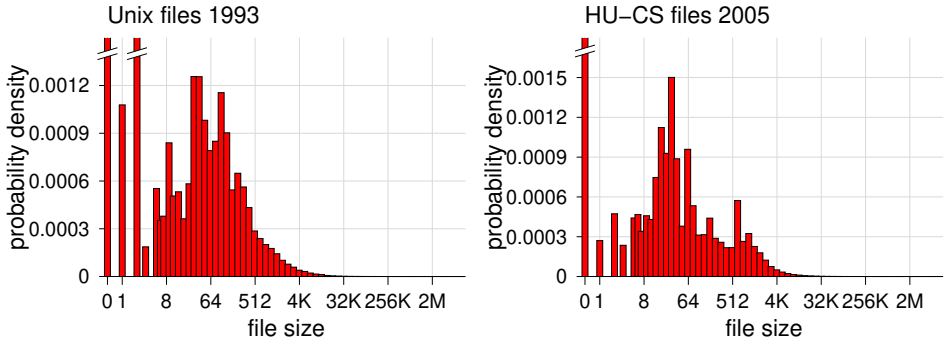


Figure 9.15: *The empirical pdf of file sizes exhibits a rough central mode and many empty files.*

shape of the distribution (in log-space) is typically not a smooth bell shape as it should be for a lognormal distribution (Figure 9.15).

- It follows from a generative model of file creation and modification.

To fit data with a heavier tail, it is possible to use a mixture of two lognormals with different means.

To read more: Using a multiplicative process as a generative model for file sizes was first suggested by Downey [188]. Mitzenmacher provides an illuminating discussion of this model, extending it to include both new file creations (independent of the multiplicative process) and file deletions [498]. Also, it should be noted that the Pareto and lognormal distributions are actually closely related [497].

Another issue that is of importance when modeling file sizes is the question of diversity: is the distribution of file sizes indeed the same in different systems? Some insight may be gained from the 1993 Unix file sizes survey, which included data from more than

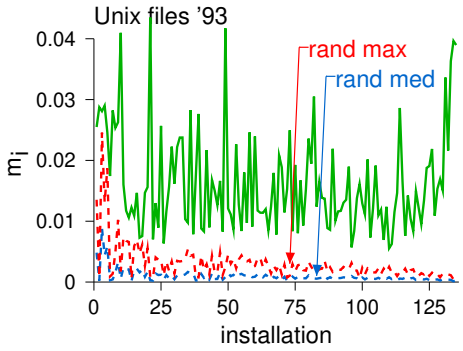


Figure 9.16: *Median deviation of the distributions of file sizes at different locations from the global distribution including all the locations. The deviations are measured in 24 ranges that are equiprobable according to the global distribution.*

one thousand different file systems located at 135 different locations. To compare the distributions, we use the same methodology as when quantifying locality of sampling, but apply them across locations [239].

Using the metrics devised in Section 6.3 to measure "spatial" rather than "temporal" locality means that the slices are not data from the same site at different times, but rather data from different sites (possibly but not necessarily at the same time). We can define a global distribution that includes the file sizes from *all* the locations, and then check the deviation between each location's distribution and this global average. The results of doing so are shown in Figure 9.16. Each location is taken as a single slice, so there are just 135 slices. The deviations between the distributions are measured by dividing the global distribution into 24 equiprobable ranges and comparing with the local distributions. For each local distribution, the median deviation of the 24 ranges is shown. To assess the significance of these results, the median and maximum deviations obtained for 100 repetitions of sampling from the global distribution are also shown. This indicates that the deviation is always significant, meaning that the distributions are different from each other. The sampling results are generally decreasing because they depend on the number of files found at each location, and locations are sorted according to an increasing number of files.

Note that this discussion relates to the distribution of file sizes in a static snapshot of a file system. This distribution can be quite different from that seen in accesses to files. In particular, temporary files that only exist for a short time will be under-represented in such a snapshot. Thus if there exists a correlation between file size and file lifetime, the distribution will be different.

9.3.2 File System Access Patterns

File system access patterns include a rich set of characteristics, such as the following:

- The relative popularity of different files.
- The existence of sets of files that tend to be referenced together or in sequence.
- Access patterns within files (e.g., sequential access vs. random access).
- The distribution of access sizes.
- The temporal pattern in which requests arrive — are they distributed uniformly in time or do they arrive in batches?
- The prevalence of reads (access to existing data) vs. writes (modifying existing data or creating new data).
- Locality in the sense of reaccessing the same data repeatedly or overwriting recently written data.
- Correlations among the above factors (e.g., are typical access sizes for sequential access different from those used for random access? Is the distribution of sizes of reads the same as for writes?)
- The lifetime of data until it is overwritten or deleted.

An early study of file system dynamics was conducted by Ousterhout et al. [530]. They found that a suitably sized buffer cache can eliminate 65–90% of the disk accesses. This result is based on the finding that 20–30% of newly written information is deleted or overwritten within 30 seconds, and 50% is deleted or overwritten within 5 minutes. In addition, about two thirds of all the data transferred was in sequential accesses of files. In files that were opened for reading or writing, but not both, 91–98% of the accesses were sequential. In files that were opened for both reading and writing, this dropped to 19–35%.

When looking at access patterns, it was found that file access size distributions tend to be modal, because they are generated by applications that access the same type of data structure over and over again in a loop [634].

An important variant is that of distributed file systems. The dynamics of file access in distributed systems are interesting because caching may cause the characteristics of the workload to look different at the client and at the server. For example, locality is reduced near the servers because repetitions are filtered out by caches, whereas the merging of different request streams causes interference [271].

Additional data about file system access patterns in different contexts is available from the following references:

- A Unix BSD 4.2 system cited above [530].
- The Sprite distributed system and NFS [46, 27].
- Personal Windows systems [711, 761].
- Multiple Windows servers operated by Microsoft [397].
- Self-similarity in access patterns [304, 293, 294, 397].

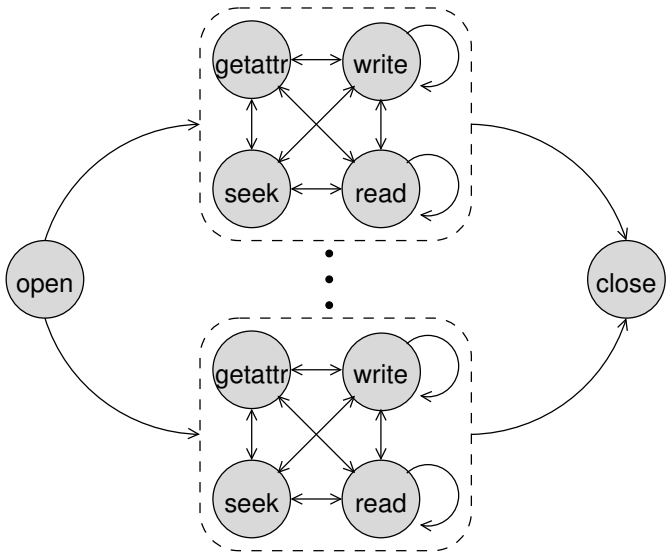


Figure 9.17: *Markovian model for sequences of file operations (following [559]).*

A possible model for file access patterns is based on a hidden Markov model (HMM) [557]. The reason is that file operations do not come in a random order, but tend to be related [559]: reads and writes tend to come in sequences rather than being mixed, seeks tend to appear alone, and open and close appear only at the beginning and the end, respectively.

An example of such a model is shown in Figure 9.17. The first state is the "open" operation, and the last one is the "close" operation. In between come several options of subchains modeling the actual operations on the file. The "read" and "write" operations have self-loops, because these operations tend to come in long sequences. The "seek" and "getattr" operations, in contrast, typically only occur as isolated instances. The reason that several such models are needed is that the probabilities on the arcs of a Markov chain dictate the probabilities of visiting the different states (see the box on page 242). Thus if we want some files to be read a lot, and others to be written a lot, we need separate submodels for these distinct patterns.

At a slightly higher granularity, I/O activity can be expressed by micro-models which encapsulate the behavior of different applications. For example a C compiler may first open and start reading a .c file, then open and read a bunch of .h files, and finally open and write a .o file. Combining such recurring micro-models is at the basis of the SynRGen workload generator [195].

9.3.3 Feedback

Performing I/O is typically not a one-time occurrence. I/O operations typically come in sequences of bursts, interleaved with other activity. From a modeling point of view, an important issue is the dependencies between the I/O operations and the other activity.

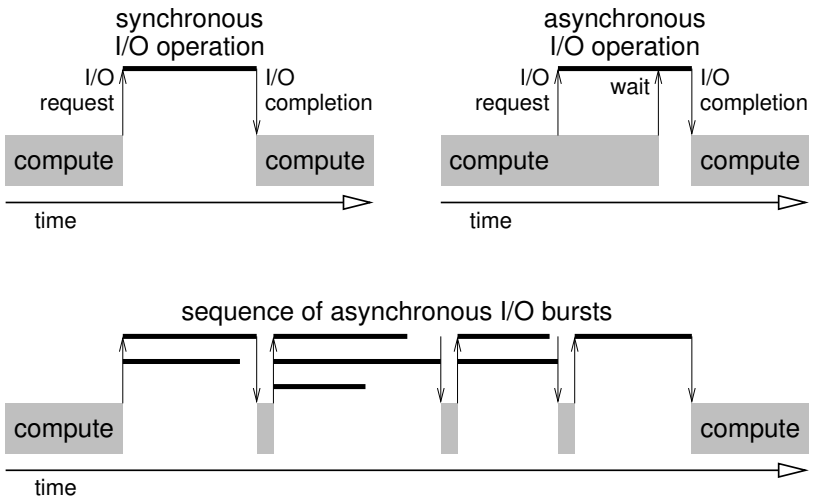


Figure 9.18: *Generic models of I/O activity.*

One obvious type of dependency is that the I/O is initiated by some computation. Therefore an I/O request cannot happen before the computation that initiates it. If the computation initiates I/O activities in a loop, it may be that the computation needed to issue each additional request is relatively unimportant; in this case we can consider a sequence of I/O requests that are dependent on each other. Alternatively, the computation may actually depend on the I/O, as when reading data from a file: if the computation needs to process the read data, it cannot proceed until the read operation is complete. But it is also possible to post an asynchronous request, which is done in parallel to the computation. In this case, the computation can later poll the I/O operation to check whether it has completed, and can wait for it if it has not. Of course, all these patterns can be combined in various ways (e.g., in a sequence of bursts, each of which is composed of several asynchronous requests). These behaviors are illustrated in Figure 9.18.

The importance of dependencies is that they induce *feedback*. If issuing one I/O operation depends on the completion of another, it also depends on the performance of the system: in a fast system the first I/O will complete sooner, and the second will be issued at an earlier time, whereas in a slow system the first I/O will take more time, and thus delay the issuing of the second one.

An important consequence of feedback is its effect on how to use workload traces correctly. In trace-driven simulation, a trace of activity recorded in a real system is replayed, and the performance of a simulated system on such a workload is measured. But if the trace is simply replayed using the original timestamps, all feedback effects are lost [350]. If the simulated system is slower than the original system in which the trace was collected, requests will arrive "before their time" — that is, before previous requests on which they depend have completed. Conversely, if the simulated system is faster, requests will appear to be artificially delayed. In either case, the results of the simulation will not be reliable.

The feedback among I/O requests may also lead to important feedback effects at the

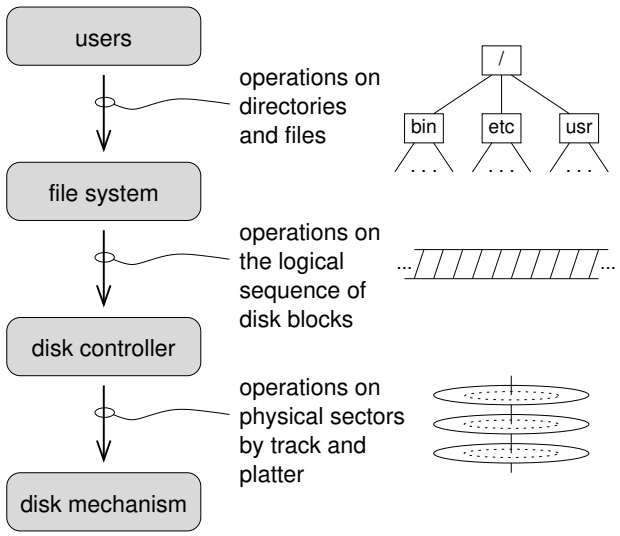


Figure 9.19: *Workloads at different levels of I/O from the file system down to the disk mechanism.*

system level. To reduce complexity it is common to model storage subsystems independently from the full systems in which they are embedded. This is based on the implicit assumption that whatever is good for the storage subsystem is also good in general. However, such an approach misses the effect of feedback between the storage subsystem and the process that generates the storage requests, as demonstrated by Ganger and Patt [277].

The demonstration of the importance of feedback effects hinges on the distinction between critical requests and non-critical requests. Critical requests are those that may cause a process to block and the CPU to idle, such as synchronous writes. These are the requests that lead to the feedback effect: if they are delayed, the whole computation is delayed, including the issuing of subsequent I/O requests. Non-critical requests are those that do not cause an immediate block, such as background writes. From a full-system point of view, it is therefore clear that critical requests should be prioritized. However, from a storage subsystem point of view, it is better to prioritize requests according to how long it takes to position the disk arm in order to serve them, irrespective of whether they are critical or not. But this is a short-sighted approach that only makes sense if we ignore the feedback caused by the critical requests. It may lead to situations where a non-critical requests is served before a critical request, thereby degrading overall performance.

9.3.4 I/O Operations and Disk Layout

The discussion so far has focused on file system operations, and indeed this is the level at which the users operate. But there are lower levels too (Figure 9.19). The file system translates the user operations on files and directories into read and write operations to the disk (after filtering out those that are served by the file system's buffer cache) [347]. This

is done at the logical block level, as modern disks use an interface in which all blocks are presented as a single linear sequence. Internally, the disk controller then maps these logical blocks to physical sectors that reside in different tracks on different platters.

Although there is little data about disk-level workloads, some patterns do emerge [570]. One is that in many systems the disks are idle the vast majority of the time — often more than 90%. Related to this is the observation that disk activity tends to be bursty. Request sizes tend to be very modal, with the most prominent mode — often at 4 KB — accounting for up to 60% of the requests. There are, however, differences in different environments. For example, in enterprise systems it is typical to have several requests queued for disk service, whereas on desktop machines requests typically arrive one at a time.

The mapping of file system structures to disk sectors is important because it might interact with the access patterns. For example, in Unix systems the parsing of a file's path requires the system to alternately read each directory's inode and data. If, say, all the inodes are stored together at one edge of the disk, this may lead to excessive seeking. Such considerations led to the design of the so-called fast file system, which distributed inodes across the disk, and attempted to co-locate each file's (or directory's) inode and data [477]. However, today intelligent disk controllers mask the actual layout, and it is becoming increasingly harder to model the latencies of different disk operations [25].

Understanding and being able to model the timing of disk operations is important not only for the study of disk systems, but also for file system workloads [586]. The reason is that the dynamics of file system operations include an element of feedback, as explained earlier. To be able to model this feedback, one has to know how long each operation will take. In principle, this requires a detailed model of the disk, including its topology in terms of tracks and sectors, and its internal features such as whether or not it caches read data [587]. But such a model cannot be applied if we don't know how the file data is mapped to the disk topology in the first place. Luckily there is evidence that disk service times are relatively consistent, averaging several milliseconds [570]. Thus, unless especially high accuracy is required, one can get away with using a simplistic model.

In addition to the hidden mapping to physical sectors, there is a problem of fragmentation at the logical level. If a file system is populated by a set of files one after the other, with no modifications or deletions, the files will probably be stored sequentially on the disk with minimal if any fragmentation. But real file system dynamics are different: files may grow incrementally with data being added at different times, and files may be removed, leaving free disk space that is subsequently used to store blocks from other files. Thus the layout of data on disk becomes less orderly with time. When simulating the use of a file system, it is therefore important to "age" it artificially [639].

9.3.5 Parallel File Systems

The workload on parallel file systems is unique in that it is common for multiple processes (that are part of the same parallel job) to share access to the same file. Therefore different accesses interleave with each other; in many cases, this is an interleav-

ing of streams of strided access used to read or write multidimensional data structures [149, 516, 416, 517, 555]. This interleaving leads to a phenomenon known as inter-process locality, because accesses by one process allow the system to predict accesses by other processes [413].

Parallel I/O is further discussed in Section 9.6.4.

9.4 Network Traffic and the Web

Networking in general and the Internet in particular have been the subject of intensive measurements and data collection for many years. Thus there exists a relative abundance of data about networking workloads. Unfortunately, this data indicates that networking workloads are very heterogeneous, and, moreover, that they change dramatically over relatively short time frames [265]. Thus there is no such thing as a "typical" Internet workload. Furthermore, dramatic changes have been occurring in network usage in recent years. Traffic used to be dominated by email, but then this changed to web traffic, and later to file sharing and video streaming. The original static web was largely replaced by dynamically generated web pages. And the desktop browsers of yesteryear are being replaced by smartphones and other mobile devices, with different capabilities and usage patterns.

In this section, we focus on some of the unique aspects of networking. However, it should be remembered that networking workloads also have general traits that are common to all workloads, such as the daily cycle of activity.

9.4.1 Internet Traffic

Internet traffic is perhaps the best studied of all computer workloads. This can be broken down along three dimensions.

The first and most commonly cited dimension is the *scale* of observation. The bottom level looks at individual packets and characterizes the traffic at the packet level. A higher level analyzes flows of related packets that together compose a logical connection (e.g., for transferring a file). A still higher level looks at sessions using a certain application (web browsing, email, telnet) that encompass many flows and even more packets. In this subsection, we focus on packets and flows. Subsequent subsections deal with the characteristics of different applications, most notably the world wide web. Of course, all the levels may be studied together, leading to a hierarchical generative model.

The second dimension is the *attribute* of the workload that is being studied. Much of the work on Internet traffic at the packet level concerns the arrival process, and, more specifically, its scaling properties. Other attributes of interest include size distributions, locality of addressing, and functionality as expressed by the use of different protocols.

The third dimension is the *class* of traffic. Most studies focus exclusively on regular traffic, which is the traffic that the Internet is intended to carry, including web access, email, and so on. But the Internet also has malicious traffic (scanning for vulnerabilities and targeted attacks on computer systems) and unproductive traffic (e.g., that due to

misconfigurations) [58, 459]. These are of interest when studying intrusion detection, for example.

A major problem with characterizing and modeling Internet traffic is its heterogeneity, both in terms of infrastructure and of usage [265]. An interesting question is therefore whether the traffic patterns seen on wide-area links (WANs) are the same or different from those that are seen within enterprises or on wireless networks. To answer this, repositories such as the LBNL/ICSI Enterprise Tracing Project and the CRAWDAD Repository of Wireless Data have been set up. Using these repositories Kotz and co-workers have conducted a preliminary analysis of wireless traffic on an academic campus [328] and Pang et al. have studied an enterprise network [532]. They show that traffic patterns within an enterprise or university can be quite different from those across a WAN, because the dominant applications are different. Within an enterprise, a large fraction of the bytes transferred tend to belong to network file systems and backups, possibly more than to web activity and email. Moreover, some of what looks like web activity is actually internal enterprise applications that use the HTTP protocol but do not really reflect web browsing. Connections are dominated by name service activity.

Packet Traffic

Internet traffic — when considered at the single-packet level — is characterized by three main attributes:

- The arrival process. In early analyses it was assumed that packet arrivals constitute a Poisson process (i.e., that these arrivals are independent and uniformly distributed). However, it is now known that packet arrivals are self-similar. This has been demonstrated at both the LAN and WAN levels [436, 540, 732]. In fact, this was the first major demonstration of self-similarity in the context of computer workloads (Figure 9.20). Self-similarity is important because it may affect the buffer sizes in switches and the probability of collisions on Ethernet networks [355] — both of which affect the ability to achieve a given quality of service and satisfy service-level agreements. More recent data shows that self-similarity is retained when the traffic scales up in gigabit networks, and even under congestion [82].

  Another important effect in the packet arrival process is feedback. This is mainly the result of congestion control in TCP [364, 84], which backs off when congestion is suspected. Importantly, this is an end-to-end effect, so traffic may flow through a link at well below this link's capacity due to congestion in some other part of the path [265]. This underscores the importance of using generative models, in which traffic is generated as part of the evaluation procedure in a way that reflects network conditions. Such models are described later.

- The distribution of packet sizes. This distribution tends to be highly modal. The commonly observed sizes correspond to the maximum transmission units of different networking technologies. The most dominant is Ethernet, with a size of only 1514 bytes (including headers).

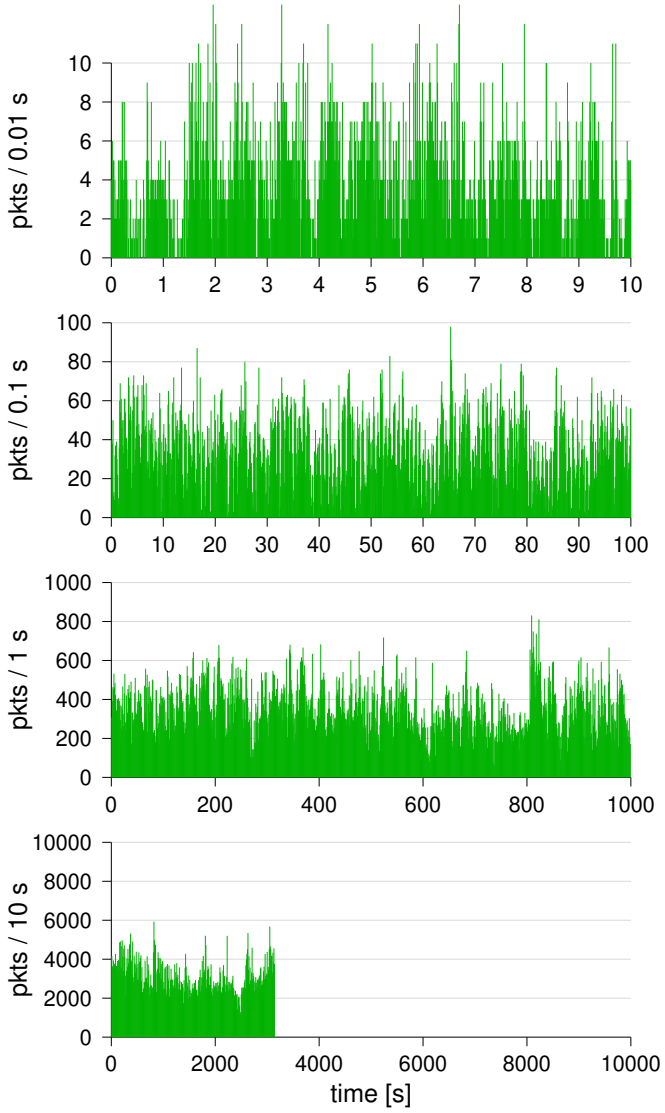


Figure 9.20: *Self-similarity in Ethernet packet arrivals, using the original dataset from Bellcore from 1989.*

- The function of the different packets, and in particular to which protocol they belong. As Internet technology evolves, the dominant protocols observed in the traffic change [252, 330]. This also affects the statistical properties of the workload [535]. Note that this effect is not limited to well-known protocols such as TCP and UDP that constitute the Internet's infrastructure, but also includes application-specific proprietary protocols layered above them.

Note that these three attributes are not necessarily independent, and therefore they cannot

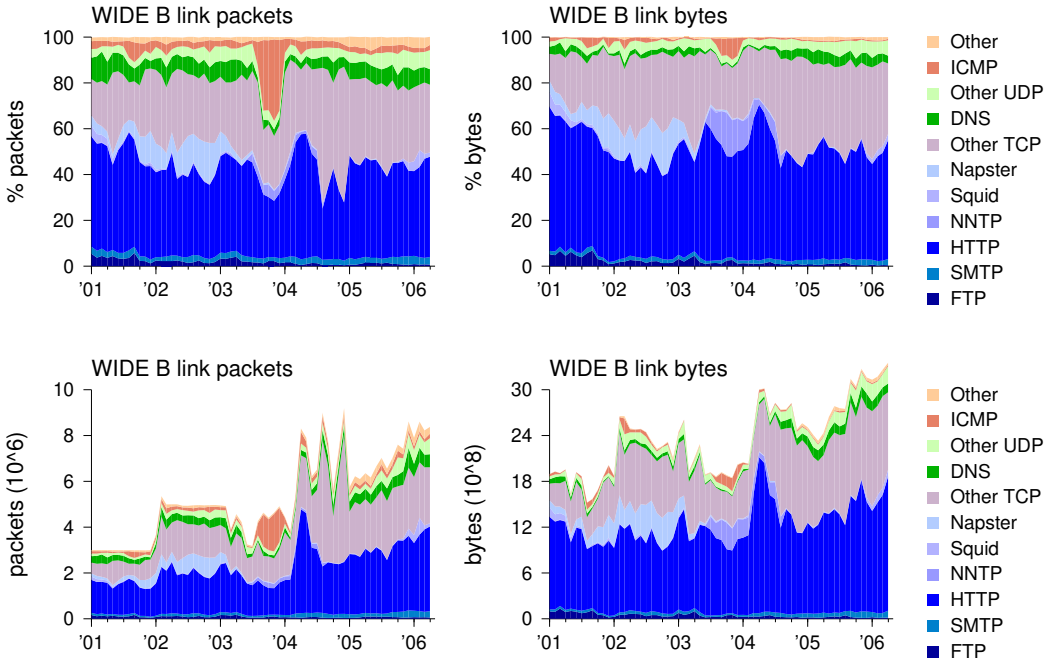


Figure 9.21: *Usage of different protocols on the Internet, in terms of packets and bytes.*

be modeled separately. In particular, certain applications and protocols may use packets of certain specific sizes. Thus if those protocols dominate the workload, they will have a strong effect on the distribution of packet sizes too. Moreover, the distribution of packets using different protocols may be quite different from the distribution of bytes delivered by the different protocols.

A relatively long-range tabulation of Internet traffic is shown in Figure 9.21. It shows monthly averages of the packets and bytes that can be attributed to different protocols, as observed on access point B of the WIDE backbone (which is a transpacific link). The top graphs show the relative fraction of packets and bytes attributed to each protocol, whereas the bottom ones show absolute counts. These graphs show weak trends of reduced FTP traffic and increased UDP traffic, but both of these are small relative to seemingly normal fluctuations in HTTP traffic. However, at least some of the relative fluctuations of HTTP are actually due to big changes in the absolute traffic using other protocols. One can also observe workload changes that only occur for a given duration, such as the massive usage of the Napster file-sharing service in 2001–2002, and the surge in ICMP traffic in late 2003 due to the spread of the Welchia worm.

The general protocol mix shown in Figure 9.21 comes from a transpacific link that carries traffic from many different sources. But networks deployed at specific locations may experience significantly different mixes depending on context. For example, in universities or large companies certain networks may be dominated by special types of activity, such as backups that use their own specialized protocols [414, 532].

An interesting observation regarding packet traffic is its symmetry, or lack thereof. Much of the traffic on the Internet is generated by client-server type applications, where the server sends much more data than the client. The assumption that this is the case is even embedded in the infrastructure of ADSL lines, which provide higher download bandwidth and lower upload bandwidth. But if one looks at packet counts, and ignores packet sizes, the communication looks much more symmetrical [484]. This is because much of the communication is based on TCP, and each packet has to be acknowledged.

Connections and Flows

An alternative to modeling individual packets is to model connections or flows. Connections are persistent virtual channels set up by protocols such as TCP (as opposed to UDP, which is connectionless). Flows are sequences of packets between the same endpoints that form one logical communication. In most cases connections correspond to flows, and the distinction is mainly one of the level being studied: connections are a feature of the underlying protocol, and flows are part of the structure of the application. In any case, both flows and connections provide a higher level of abstraction than individual packets, and correspond to application-level activities such as the transfer of a file or servicing of a request. Sequences of successive or parallel connections form a user session.

Identifying connections in a packet trace is done based on the five-tuple of fields from the packet header, consisting of the IP address and port of the sender, the address and port of the receiver, and the protocol used. For TCP connections this can be augmented using specific flags in the TCP packet headers: SYN and SYN/ACK are used to set up a connection, and FIN or RST to terminate it. The sequence of intervening packets can be identified using the size, sequence number, and acknowledgment fields, which specify how many bytes have been transmitted so far in each direction. Of course, packet reordering and retransmissions need to be handled correctly. Thus an alternative is to just consider all packets between the same endpoints with short intervals between them (e.g., less than one minute) [338]. This has the advantage of also handling lagging packets and connections that are not terminated properly.

A tool that models Internet flows in an application-neutral manner is Harpoon [641]. The model is based on four distributions:

- The distribution of flow sizes.
- The distribution of intervals between successive flows between the same endpoints.
- The distribution of the number of active sessions at any point in time.
- The distribution of source and destination IP addresses. (Interestingly, despite constant warnings that IPv4 addresses are about to run out, it appears that a large fraction of them are actually not used [107].)

The implementation of the model (i.e., the generation of synthetic network traffic) is based on running processes on the different hosts of the testbed, and having them send files to each other according to these four distributions.

Time frame	*Dominant effect*
milliseconds	shaping due to congestion control
subsecond to minutes	self-similarity due to multiplexing of on-off processes
hours	nonstationarity due to daily work cycle

Table 9.1: *Different effects dominate Internet traffic when observed at different time frames.*

The distribution of flows or connections has generally received more attention than that of other elements. Several papers have reported on the fact that it has a heavy tail (which is sometimes linked with the heavy tail of the distribution of file sizes, because large connections are typically used to transfer large files). However, the distribution as a whole is not well modeled by a Pareto distribution. It has therefore been suggested to use a mixture, in which the body of the distribution (about 99% of the mass) is lognormal and only the upper tail is Pareto [265]. Flows also figure in the on-off model of self-similarity [732]. However, their role has been debated based on the finding that bursts of activity do not result from the confluence of multiple flows, but rather from the arrival of a few high-rate flows [592].

Wireless Traffic

Most studies of Internet traffic focus on the links between backbone servers. Others look at local-area networks (LANs). A special kind of LAN that has become prevalent at the beginning of the 21st century is the wireless network, in which mobile devices connect to access points in an ad-hoc manner. Although much of its traffic characteristics are similar to conventional wired networks and depend on the context (e.g., a university campus vs. a company's enterprise network), user mobility is creating new phenomena to study.

The mobility of users was discussed earlier in Section 9.1.5. But note that a complete user mobility model may depend not only on user behavior, but also on physical characteristics of the system. Specifically, Kotz et al. [415] show how the movement among different access points may depend on radio propagation characteristics, because the nearest access point may not always provide the strongest signal due to blocking or fading effects. In particular, they point out that common assumptions used in models and simulations, such as that all access points provide full coverage in perfectly circular areas with the same radius, are way too simplistic.

Wireless devices also have special characteristics in terms of session lengths. This pertains mainly to voice over IP devices which are always on in order to be able to accept calls. They therefore have very long sessions [328]. Other mobile devices that connect to a network only when they are opened for use, such as laptops, have shorter sessions.

Generative Models

The properties of a generative model for Internet traffic depend on the time frame that is of interest. At different time frames, different effects dominate, and these have to be incorporated into the model. The main ones are listed in Table 9.1.

For networking research, the time frame of interest is often seconds to minutes. The main reason is that this time frame covers the typical round-trip time of sending a message and receiving a reply or acknowledgment. It is also of interest because of the rates involved: just a few minutes of activity can translate to millions of packets. The reason for not modeling short-term effects related to congestion control is that they are not part of input traffic but rather an outcome of network conditions. Thus such effects should be introduced as part of the evaluation procedure, and not imposed on the evaluation from the outset. As a result of these considerations, generative models of Internet traffic focus on source-level modeling of how the traffic was generated [265].

Internet traffic at the packet level is obviously the result of some higher-level activity at the application level, or, rather, the interleaving of transmissions generated by many users using diverse applications. A useful generative model can be constructed based on common activity patterns when using different applications. For example, web browsing can be characterized at the following five levels [36, 336, 446, 536, 576]:

1. The client session: all the documents downloaded by a single user.
2. The "navigation burst": the set of documents downloaded in rapid succession as the user homes in on the desired information.
3. The document, with its embedded objects.
4. The request-response HTTP protocol used to download each object.
5. The packets used to transmit a single object (requests are typically small and fit in a single packet, but responses can be large files requiring multiple packets).

Interestingly, the temporal structure of the different levels is quite distinct. The top level, that of user sessions, may be characterized as a Poisson process, in which new sessions start uniformly and at random and are independent of each other (but actually this is a nonhomogeneous Poisson process, accounting for different arrival rates at different times of the day [540]). This structure may also extend to navigational bursts [536]. The bottom level, that of individual packets, is characterized by self-similarity, which is a result of the structure of superimposed on-off processes. Intermediate levels may be harder to characterize [265].

An interesting tool to generate Internet traffic that matches real observations is Tmix [725]. This reads a traffic trace and extracts communication vectors that represent the activity of TCP-based applications. Each communication vector is a sequence of tuples $\langle \text{request\_bytes}, \text{response\_bytes}, \text{think\_time} \rangle$ representing successive back-and-forth communications by the application; fields may be given a 0 value if not used in a specific communication. During an evaluation, corresponding messages are sent at times that reflect the interaction between the system dynamics and the user think times. Importantly,

this allows all the observed TCP-based activity to be included in the evaluation, even if we do not know what the applications were or what they actually did.

A more sophisticated traffic generator is Swing [710]. It uses a given trace to extract data about users, applications, and network conditions. Then it uses a full network emulation to generate a trace of the packets that flow on a single designated target link. The full emulation is used to create realistic round-trip times for request-response communication patterns. The generation of traffic at the endpoints uses think times to incorporate feedback into the generation process, so such timing details are important.

9.4.2 Email

Internet traffic is generated by computer applications. To understand and evaluate these applications, one needs to consider application-level workloads. One of the oldest applications on the Internet is email.

Like many other applications, email workloads have changed with time. In the 1980s users worked on terminals connected to multi-user systems, and email was transmitted from one host to another using the SMTP protocol. Today most email access is done via the web. This means that the email-induced workload has another component: the connection from the user's desktop or mobile system to the mail server.

Access to a mail server using PoP (Post Office Protocol) shows that there are three main classes of users [72]: those who typically do not get mail, those who typically download everything but also leave it in the mailbox, and those who typically download and delete from the mailbox. The relative proportions of these three user populations obviously affect the load on the mail server (e.g., in terms of required disk space).

A unique aspect of email is that in addition to being interested in how it is transmitted (which induces work on servers and networks) we may also be interested in classifying it (which induces a different type of work on mailer applications). The problem with doing research on email classification is one of privacy. Probably the only publicly available dataset to use is the Enron email archive, which was published as part of the litigation following the company's collapse, and has been used to train and evaluate machine-learning classifiers [407].

Of course, any characterization of email will not be complete without a characterization of spam and other malware propagation. In fact, spam accounts for more than 90% of all email sent [666]. However, there is little data about its composition, except the obvious characteristics of bulk email: huge numbers of copies of the same messages. As for viruses that spread by email, analysis of user behavioral patterns (and especially deviations from them) has been suggested as an effective identification method, independent of the actual mail contents [665].

9.4.3 Web Server Load

The dominant Internet workload in the period 1995–2005 was world wide web (WWW) traffic. There has therefore been much effort devoted to analyzing the resulting traffic patterns and how they interact with proxy caches and web servers. Since 2005 the

Attribute	*Details*
success rate	nearly 90% successful in original study, down to 65–70% ten years later, with the difference largely made up by "not modified" indication for cached data
file types	90–100% were HTML and images in original study, down to 70–86% ten years later, but difference largely made up by CSS and directory access*; note that there was wide variability in fraction of HTML or images when considered separately; video and postscript/pdf files may be responsible for large fraction of volume transferred
typical transfer size	the median is small, around a few kilobytes; but the mean grew from the original study to the follow-up, because the maximal size of transfers grew
size distribution	both distinct files and transfers are heavy-tailed
popularity	file popularity is Zipf distributed with parameter near 1
distinct requests	well over 90% of requests and data are repetitions (in logs of several months)
one-time referencing	about a third of documents and bytes are requested only once
inter-reference times	times between repeated requests to the same file are exponential and independent
remote requests	around 70–80% of requests are from remote sites
request sources	a small number of sources generate most remote requests

* Directory access is used as shorthand to access a default file, usually index.html. CSS files specify formatting style.

Table 9.2: *Web server workload invariants according to Arlitt and Williamson [37, 38, 729], based on data from 1995 and 2004, mainly from three universities.*

dominant type of traffic has probably been peer-to-peer file sharing and streaming, but positively identifying applications is becoming more difficult.

As with file systems, characterizing the workload on the world wide web has two components: the data available on servers and how that data is accessed by clients. One of the main characteristics of web workloads is the prevalence of power laws [9]. Examples include the distribution of file sizes on web servers and the distribution of popularity of the different files [155, 57, 85, 572, 525]. Moreover, the distribution of transfer sizes (which is the product of the interaction between file sizes and their popularities) is also heavy-tailed. Such power laws figure prominently in an early study of web workloads by Arlitt and Williamson, which identified 10 workload invariants (Table 9.2) [37, 38]. Most of these indeed seem to be persistent attributes of web workloads, as demonstrated by a follow-up study 10 years later — they stayed the same despite a marked increase in the overall load [729]. It should be noted that the study focused on academic sites, hence the relatively low fraction of file types such as video. Overall similar results were also

found in a follow-up study focusing on scientific websites [225]. Other types of sites, such as e-businesses and search engines, are considered later. File sharing typically uses a distinct protocol to transfer files, so the bulk of the traffic there is not considered web traffic.

As an example of the importance and impact of such workload characteristics, the heavy-tailed request sizes (known for static pages) suggest the use of SRPT scheduling when serving them (at least if we assume that service time is strongly correlated to file size, which has been contested [453]). This leads to optimal response times, but with a danger of starving the largest requests, which are deferred in favor of smaller ones. However, studies show that this does not, in fact, happen to any large extent [318, 47].

In addition to individual servers, it is important to study the workload on server farms that host a multitude of individual websites [65]. For example, this level of aggregation may have an impact on caching. However, at present little if any data is publicly available.

URLs and Pages

The study of web server loads is based on server logs that record all the requests fielded by a given server (the format of such logs was described on page 29). These server logs include a specification of the requested page in the form of a URL (Uniform Resource Locator). Whetehr a URL is identified with a page depends on the type of request being made.

The first and simplest type involves static web pages. In this case the URL indeed identifies a specific file stored on the web server, and this file is returned to the client.

A more complicated case occurs when dynamic content is involved. In this case the request is actually a query composed of two parts: a path identifying the script that should be used to generate the content, and a sequence of one or more parameters to that script. For example, the request for

```
http://search.aol.com/aol/search?q=workload+modeling
```

specifies search.aol.com as the server, /aol/search as the path, and workload+modeling as the value of the parameter q, namely the query itself.

At first glance it may seem that requests for dynamic content do not identify the returned page of results, because this is created dynamically upon request and may change each time. This is true but irrelevant: in web servers that generate content dynamically, it is the query and not the delivered page that identifies the interaction. Therefore the full request can in fact serve as a unique identifier. For example, if the same query is made repeatedly by multiple users, it may be worthwhile to cache the results instead of regenerating them, at least for some limited time. The only qualification is that in some cases the results are personalized per user, and therefore the query does not reflect all the considerations regarding the page that will be generated. Also, it should be remembered that static web pages also change with time.

A bigger problem of identifying requests occurs when a clickmap is used. A clickmap is a large graphical image that invites users to click on various locations. These clicks

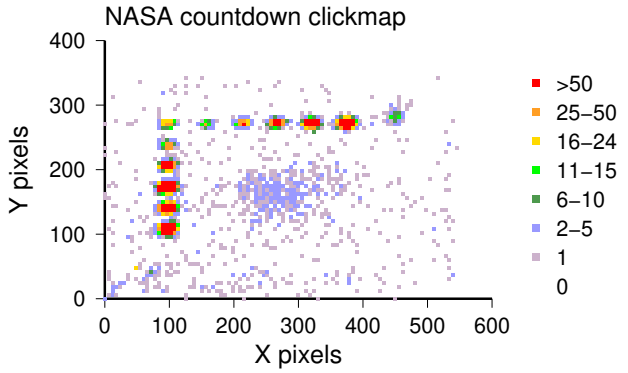


Figure 9.22: *Locations of clicks on the clickmap /cgi-bin/imagemap/countdown from the NASA web server log.*

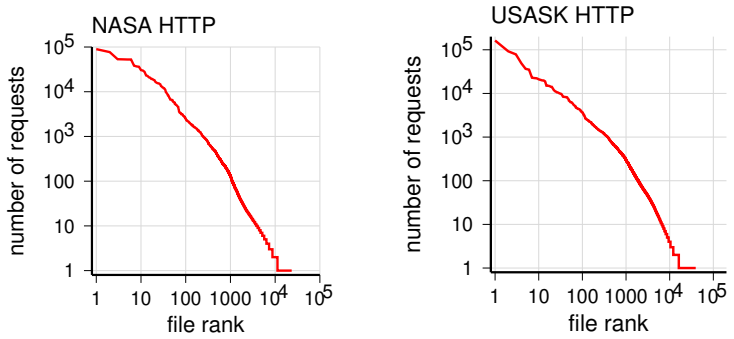


Figure 9.23: *The popularity of documents on a web server approximately follows the Zipf distribution.*

are then sent for interpretation by a script, giving the X and Y coordinates of the click location as parameters. In reality, the clickmap is divided into rectangles, and all the clicks that fall within the same rectangle lead to the same result. But the requests may be different, because they reflect different coordinates within the rectangle.

An example is given in Figure 9.22. The concentrations of click locations indicate that the clickmap most probably included a vertical row of button on the left and a horizontal row across the top — enabling the reconstruction of the rectangles that represent individual targets. Clicks around the center may or may not be meaningful, and the scattered clicks throughout the image area most probably are not.

Page Popularity

Examples of page popularity are shown in Figure 9.23. They resemble a Zipf distribution, as indicated by the arguably straight line of the count-rank graph [37]. There

are several reasons for this skewed distribution and the very high number of requests to the top-ranked pages. One is the hierarchical nature of browsing: visitors typically start from the home page, so the home page (and graphical elements embedded in it) enjoys the most requests, followed by other high-level pages. Another is that web pages are often updated or even generated dynamically, so requesting them many times actually leads to different information being obtained [309].

A third reason for increased skew is that the client may be browsing a site using the "back" button. It is pretty common for users to click on successive links in a page, each time returning to the original page using the "back" button and then continuing with the next link. Data from 1995 indicated that there were three "back" operations for every four forward link traversals on average [113]. This was attributed to the need to search a site to find the desired information. However, newer data from 2008 showed considerable reduction in the use of "back", probably due to the more interactive nature of web work and the availability of multiple tabs and windows [727].

Finally, popularity is also affected by search engines. An increasing fraction of traffic enters a website by a redirection from a search engine, rather than by browsing through the site's home page (data for blogs indicates that search engines lead to a third of all read requests, higher than any other individual source [192]). Thus pages that rank high on search engines tend to become ever more popular, whereas new pages that are not indexed have a much harder time [132].

It should be noted that the popularity distribution is affected by proxies, and the distribution of requests sent by a client may be quite different from the distribution as seen by a server. This is the result of an interplay of caching and merging [267, 266]. When a request stream passes through a cache, it becomes less skewed, because many of the requests for the popular items are served by the cache and are not propagated to the server [309]. But when streams are merged they become more skewed, because the popular items typically appear in all of them while the less popular do not. The end result is that the distribution of requests is more skewed at the server than near the clients [572, 266]. Knowing the characteristics of the skewed popularity is important for designing cache policies [657].

Locality

The skewed distribution of popularity naturally leads to locality, as the most popular pages are requested again and again. This leads to a stack-depth distribution with a very high propensity for small stack depths (Figure 9.24). In particular, the median stack distance in the relatively short SDSC trace is 40, and in the much longer NASA trace it is only 52.

However, individual users may exhibit different browsing patterns with different degrees of locality. One suggested classification differentiates between a "mostly working" reference pattern, in which the same documents are accessed repeatedly, and a "mostly surfing" pattern, in which users rarely return to previously accessed documents [525].

Although popularity and locality have an obvious importance for caching, one should nevertheless remember that other factors also have an effect. In the context of web

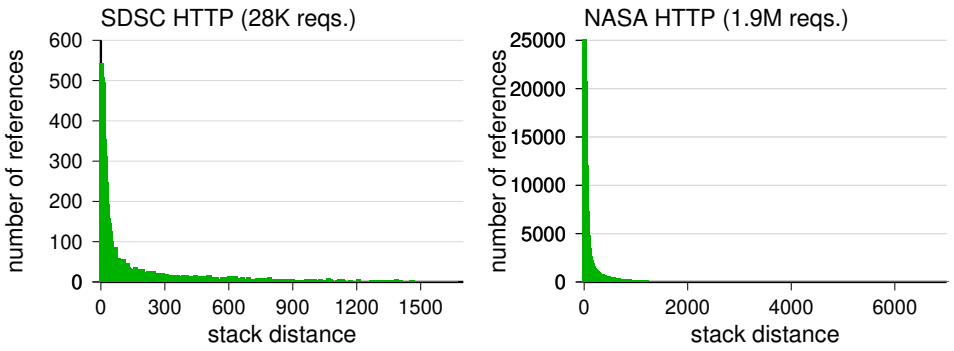


Figure 9.24: *The stack distance distribution from a stream of requests to an HTTP server.*

servers, an important factor is the use of cookies. Cookies are typically used for personalization of web pages, so requests that include cookies are typically not served from a cache [65], which may increase the load on the web server more than is strictly necessary.

Generating Representative Web Server Loads

Various tools have been created to generate representative web server loads in order to test (and stress test) web servers. One of the early models is SURGE (Scalable URL Reference Generator) [57], which implements a generative model based on user equivalents. Each user equivalent is a thread that sends requests to the server, and the number of such user equivalents defines the load. To provide self-similarity, user equivalents operate in an on-off manner, with heavy-tailed off times. Request sizes, object popularity, and the number of embedded objects in a web page are also Pareto distributed.

Another well-known workload generator is SPECweb2005 (as with other SPEC benchmarks, it is updated at irregular intervals of several years; the previous version was SPECweb99) [656]. It is a combination of three workloads, representing a banking application, an e-commerce application, and a support application. The requests sent to the server are generated by multiple simulated user sessions. Exponentially distributed think times are inserted between successive requests from the same session. The applications include dynamic content creation, and the requests include the simulation of caching effects by sending requests for images with an "if-modified-since" flag. The number of sessions, how many run each application, and so on are configurable workload parameters.

Flash Crowds

One of the remarkable phenomena of web workloads is the flash crowd, also known as a traffic spike, which is a huge surge of activity directed at a specific website. These flash crowds have been observed relating to diverse events, including Live-Aid rock concerts,

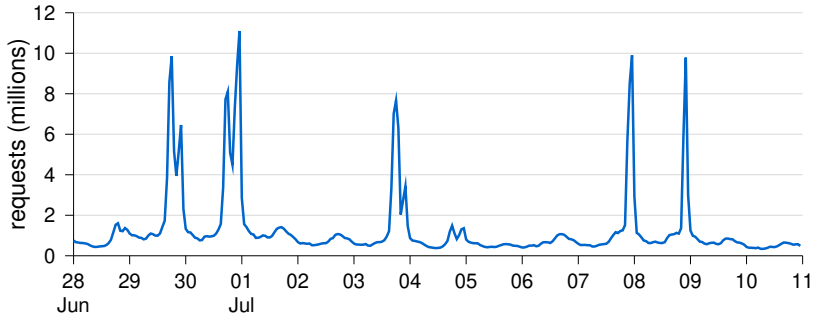


Figure 9.25: *Volume of arrivals at the 1998 World Cup site, toward the end of the tournament, shows a flash crowd at the time of each major game.*

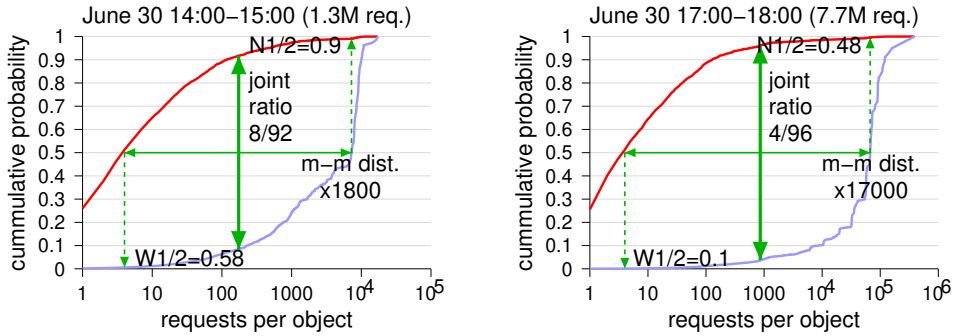


Figure 9.26: *Mass-count disparity plots of web activity before and during a flash crowd.*

Victoria's Secret fashion shows, Microsoft software releases, and singular news events such as 9/11 [434, 438]. Although each is a unique and singular event, and hence an anomaly that does not reflect normal usage, such flash crowds have similar dynamics and are of prime interest due to their effect on the stability of web servers and Internet routers.

An example of a workload trace that includes several such events is the HTTP trace from the 1998 World Cup website, collected by Arlitt [35]. The full dataset contains more than 1.3 billion requests spanning three months. But the activity was far from being uniform. The first half recorded activity from before the tournament started and displayed relatively moderate load. During the tournament itself, there were huge surges of activity relating to each game that was played. Data from the end of the tournament is shown in Figure 9.25. On June 29 and 30 two games were played. On July 3 and 4 the quarter-finals were played, again with two games on each day. The semi-finals were played on July 7 and 8, one game on each day. Most of these games caused activity levels about an order of magnitude higher than normal.

Figure 9.26 shows mass-count disparity plots for the activity on June 30, before and during the flash crowd. Although the joint ratio grows a bit, the main difference is seen

in the median-median distance metric, which grows from about 1,800 to about 17,000. This indicates that the vast majority of the additional activity is concentrated on a small subset of the pages (a "hotspot"). Similar observations have been made for other traffic spikes as well [76].

Many flash crowds may be anticipated because they relate to prescheduled events, such as sporting events [35, 438]. In such cases the infrastructure can be prepared in advance, but still the magnitude of the flash crowd may be surprising. In other cases the flash crowd is a surprise, as may happen with relation to news events, when a popular site publishes a link to another smaller site (the "slashdot effect"), or when there is an epidemic in the blogosphere [13]. An interesting question is then whether such flash crowds can be identified automatically in real time, thus enabling the server to take remedial action. Data from a news-on-demand server suggests that little time is available, because such spikes of activity form within a few minutes [382]. In other cases too it was found that spikes resulting from surprising events build up more quickly than the spikes related to anticipated events [76]. A possible approach is to measure performance indicators online, to identify the deterioration that occurs as a result of overload in real time [123].

From a workload modeling perspective, adding flash crowds to a workload is important as a means to investigate a system's resiliency to such events. Two main attributes should be modeled: the manner in which the load grows, and the concentration of the additional requests on a limited number of web pages. These attributes can be combined by selecting a set of pages that will become the hotspot, and creating successively more requests for these pages for the duration of the desired traffic spike. The number of flash crowd events is usually not modeled, because the goal is not to mimic reality but rather to stress test the system. Thus either a single event is used to study the system's behavior in detail, or many are used to derive a statistical characterization.

Currently, the main way to cope with a flash crowd is to use a hosting service that adapts to the load level, or to use tools to balance the load among multiple servers [210]. An alternative approach is to reduce the amount of content being delivered, so as to enable a higher throughput under the given bandwidth constraint [2, 88, 296, 297]. Using an admission control mechanism to prevent additional overload has also been suggested [123]. On a more global scale, adequate caching mechanisms can alleviate the problem [33].

Robots

Part of the load on a web server is typically denerated by robots, such as those used to crawl the web and collect data for search engines [180]. Interestingly, search engines themselves are also not spared such visits, and search engine logs contain unmistakable fingerprints of robots too [372].

In general, robot behavior is quite distinct from human behavior [525]. For example, a robot may repeatedly crawl a website at certain intervals. This leads to a unique reference pattern, in which all available documents are visited the same number of times, as opposed to the highly skewed distribution of popularity that characterizes human ac-

cesses. As a result robot activity may distort the results of workload studies that are aimed at characterizing human behavior [298].

In addition, robots may control and modify their behavior. For example, the robot exclusion protocol allows website owners to create a robots.txt file that contains instructions that robots are requested to honor. The basic protocol just lists areas in the site that robots are supposed not to visit. Nonstandard extensions also allow site owners to specify when robots may visit the site (e.g. only at night) and at what rate.

Characteristics of robot behavior include the following, which may be used to identify them [180, 281].

- Web robots, especially those that crawl the web in the service of search engines, are typically interested mainly in text. Thus they tend to avoid downloading embedded images, because such images consume significant bandwidth and are hard to classify.
- Robots are capable of accessing distinct pages in a website at a rate that is much faster than what humans can achieve.
- In contrast, robots may spread out their activity and access the site at, say, precise intervals of five minutes in order to reduce their impact.
- Robots tend to spread their attention more evenly, retrieving more pages from the site, and not repeatedly retrieving the most popular pages.

9.4.4 User Sessions

The flip side of web server load is the user sessions. The definition of "session" in this context is typically the sequence of requests made by a single user to a single website. Thus it is not necessarily equivalent to all the requests a user issued in one sitting. In some contexts (e.g. e-commerce) the difference may not be so important. In others, such as web search, activity spanning multiple sites (the search engine and the sites it recommends) may reflect a single continuous endeavor from the user's perspective. Thus the client-side interpretation of a session may be quite different from that of the server side [727].

Data about user sessions is typically not available. Therefore data about sessions is extracted from web server logs, usually based on defining a threshold on the intervals between successive requests, and assuming that longer intervals reflect session breaks. Obtaining session data in this way was discussed on page 399.

Few analyses of web sessions have been conducted [34, 298]. Results indicate that sessions tend to be very short (few minutes and few requests), with a significant fraction containing only a single request. However, some are much longer, and the distribution of session lengths (or number of requests) has a heavy tail.

An especially important aspect of sessions is their failure characteristics [298]. It is common to study failures at the level of individual requests. But in web contexts, e.g. with regard to e-commerce, it may be claimed that the entire session (and especially the "buy" action) is what counts. This motivates an identification of sessions and a deeper analysis of the workload structure.

9.4.5 E-Commerce

Although modeling general web server loads may be useful, in some cases the specific characteristics of the web server application have to be taken into account. For example, websites dedicated to e-commerce are a special case of websites. So although many of the findings outlined earlier regarding web server loads and flash crowds apply to them, modeling their loads does have its unique requirements.

A hierarchical model of e-commerce activity has been proposed by Menascé et al. [487]. As in many other human-centered workloads, the data exhibits strong daily and weekly cycles. The top level is user sessions. When measuring session length in requests for e-commerce business functions, the distribution is heavy-tailed: most sessions are short (up to 10 requests), but some are very long. However, the long sessions were probably the effect of robot activity. The second level is the sequence of requests fielded by the business application. The most common requests were to the home page and to browse, search, and view items on the site. Requests that reflect actual commercial activity (adding to the shopping cart, login to account, and paying) were much rarer. The popularity of search terms followed the Zipf distribution. The third level is the underlying communication infrastructure with its Internet protocols (but the model does not descend below the HTTP level). This shows that arrivals are correlated and not independent.

Another workload model for e-commerce is implied by the TPC-W benchmark [278]. This has a client-server structure, with clients being browser emulators that create the workload on the server, which is the system being evaluated. Each browser emulator mimics the behavior of a user, using a sequence of web interactions separated by exponentially distributed think times. The interactions are selected at random based on a user profile (emphasis on browsing or on buying) and a graph that indicates what interactions may follow each other, based on a typical structure of the web pages in e-commerce sites.

An important characteristic of e-commerce systems is that they are typically dynamic, meaning that much of the content delivered to clients is generated on the spot based on individual client-specific data. This process is typically based on the LAMP architecture, which consists of four layers: the Linux operating system, the Apache web server, the MySQL database, and the PHP scripting language (or commercial equivalents of these open-source products). The workload must exercise each of these layers in a representative manner. In addition, the workload must also conform with the business logic of the site.

Sequences of Operations

One important aspect of the business logic is the order of operations. For example, in a real system one cannot place an order before adding some item to the shopping cart. It is therefore natural to use Markovian models, in which the different operations are the states of a Markov chain, and the transition probabilities represent the probability that one operation follows another [486]. Conveniently, these probabilities are easy to

estimate from data by simply counting how many times each operation occurred after each other operation. (For background on Markov chains, see the box on page 242).

However, creating and using a simple Markov chain may lead to sequences that are actually illegal. For example, in a real system we may see a transition from the "delete item" state to the "purchase" state, based on the relatively common scenario in which a customer places several potential buys in the cart while browsing, and then makes the final selection among them when checking out. But a simple Markov chain cannot capture such dependencies, and might erroneously create a sequence where a single item is placed in the cart and deleted, but the client nevertheless proceeds to the checkout [417]. An alternative that does not fall into this trap is to base the model on "sessionlets" — excerpts of real sessions that contain legitimate sequences that can then be stringed together.

Another problem with using a Markov chain model is that such a model necessarily leads to a fixed limiting distribution of the different states, and thus a fixed limiting distribution of the different transactions in the workload. Real e-commerce workloads, however, do not exhibit such a fixed distribution. Rather, they are nonstationary with a different mix of transaction types at different times of the day [662]. Also, clients may have different profiles, which require representation by different transition probabilities [486].

User Community Modeling

A workload modeling approach geared to handle these issues, which has been used by performance analysts in the field, is called the user community modeling language [54]. This is a graphical language that depicts usage scenarios in a website. By defining the scenarios, illegal sequences of actions are avoided. In addition, the language allows for different user types and the specification of which actions are more common then others.

An example is shown in Figure 9.27. This is similar to but not quite the same as a user behavior graph. Sequences of actions flow from left to right. As indicated on the left, three classes of users are identified. Some of the scenarios are unique to a certain class, but others are shared. Horizontal solid lines denote actions, with a possible indication of the fraction or number of users who are expected to perform them. The dashed tabs below certain actions identify details that should be provided as part of the model. A vertical dashed line denotes a synchronization or convergence point. The sequences of actions may include divergence points, merge points, conditions, and loops. An important feature is the distinction between orderly logout when finished and a plain exit where a user just quits the system, possibly due to frustration with the results so far.

The graphical representation just depicts the possible sequences of actions. Additional data is needed to fill in the timing details. This is provided in the form of distributions that specify the time needed to perform different actions and the think times between them.

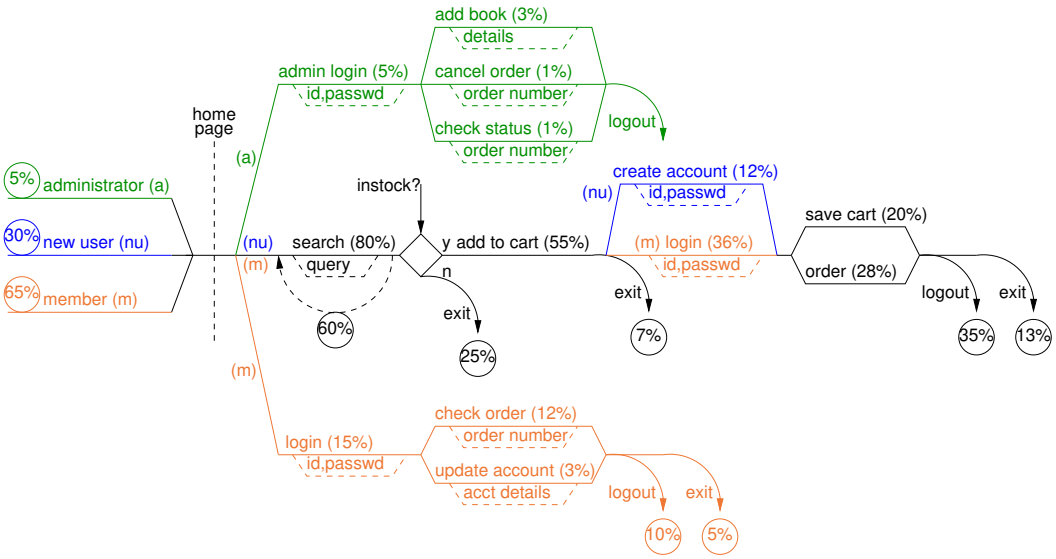


Figure 9.27: *Simplified user community model of an online bookstore (based on [54]).*

9.4.6 Search Engines

The workload on a search engine is a sequence of the queries with which it is presented. These queries may come from at least three different types of sources:

- Individual users who type in their queries using a web-based interface.
- Meta engines that accept user queries and then forward them to multiple search engines in an effort to obtain better coverage and find better results [606, 220].
- Software agents (robots) that submit multiple repeated requests to scan the search engine's contents or in an effort to manipulate its behavior.

The first two sources are genuine, but should still be distinguished from each other if one is interested in modeling the behavior of individual users. The third source is actually unrepresentative of real user behavior and should be filtered out in most cases. This filtering may be done based on the number of queries being submitted [372], their rate [194], or the fact that no results are clicked on [758].

User activity with regard to search queries has two main dimensions: how many queries are submitted and how many results are followed for each one. These are combined into a so-called clickstream, that is, the stream of clicks leading to visits to different web pages. Some clicks represent the submittal of a new query, some a request for an additional page of results, and some a drill down into one of the results to verify that it indeed satisfies the information need [390].

Data regarding the number of queries is shown in Figure 9.28, using the AOL U500k dataset (comprising a total of 20,792,287 queries from 657,425 users). This indicates a moderate mass-count disparity — 24% of the users are responsible for 76% of the queries, and vice versa; the bottom half of the users together generate only 7% of the

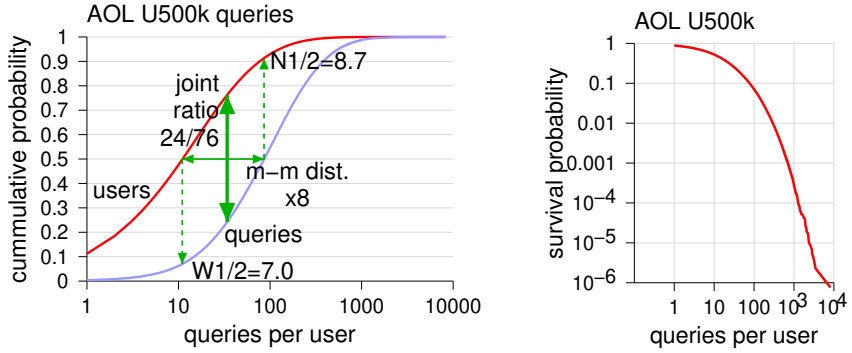


Figure 9.28: *Distribution of search engine query generation by users exhibits moderate mass-count disparity.*

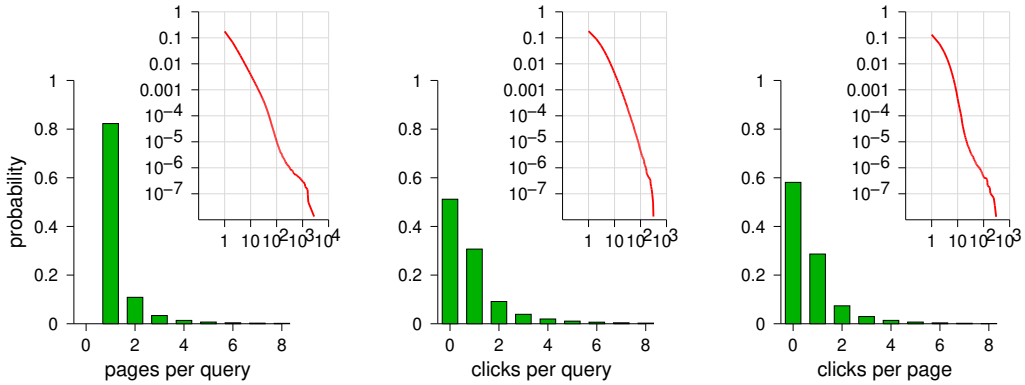


Figure 9.29: *Distributions of number of events per query. Insets show LLCDs of the tails.*

queries, while half of the queries are generated by less than 9% of the users. The distribution may have a power-law tail, but if so the tail index is around 3, as indicated by the LLCD plot on the right.

Data regarding the use of search results is shown in Figure 9.29, using the same AOL data. For each query, the sequence of consecutive events recorded was counted. There are two types of events: requests for the next page of results or a click on a link in the current page. The leftmost graph shows that in 82.3% of the queries only a single page of results is examined, and in another 10.9% two pages are viewed. However, there are rare cases when many pages are requested, with the maximum being 2775 (but note that users who request very many pages may actually be robots). Moreover, the distribution seems to have a heavy tail, albeit with a tail index around 2.

The other two graphs show clicks. The middle one shows the total clicks for a query, summing over all the pages of results: 51.2% of the queries do not lead to any clicks,

30.7% lead to a single click, and 9.1% to two. The maximal number observed in the log was 314, and indeed the tail of the distribution is seen to drop rapidly. The rightmost graph shows the distribution of clicks per page. Given that for most queries only one page is retrieved, it is not surprising that this distribution is similar to the previous one. In 58.1% of the result pages no results are clicked on. In 28.7% there is a single click, and in an additional 7.4% there are two. The maximal number of clicks is 303 — much more than the number of results in a page. This indicates that there are cases where the same result is clicked more than once. Indeed, this happens also for low numbers of clicks: it is not uncommon for a user to click on a result, back up and click on another result, and then back up again and return to the first result.

Regarding the design of search engines, it is also interesting to characterize queries in terms of what users are searching for and how they express it. One aspect of searching is the reformulation of queries to improve the quality of results. This is an interactive process in which users repeatedly modify their queries and resubmit them. Related to this are repeated searches for the same information [681]. Another aspect is the use of search terms — how popular each term is and which terms tend to come together.

At a high level of abstraction, queries may be classified into three types based on the needs of the user [91]:

Informational queries: These are queries that simply look for information regarding a certain topic. For example, the query may be "workload modeling" or "web search taxonomy".

Navigational queries: These are queries where the user is trying to reach a specific website, typically the home page of some company or organization. For example, the query "IBM" most probably indicates a search for www.ibm.com.

Transactional queries: Here the user is trying to perform some action, but probably does not care which website will be used to do it. For example, a search for "Sinai tours" can be used to organize a trip using any of several websites suggested by the search engine.

This classification is important for user modeling of web search and also for the evaluation of search engines. Broder estimates that approximately 50% of queries are informational, 30% transactional, and 20% navigational [91], but others have claimed that there are much fewer transactional and navigational queries [371].

Delving into the details, data regarding search term popularity is shown in Figure 9.30. For these graphs, repeated executions of the same query (e.g., to obtain additional pages of results) are only counted once. However, if the query was modified it was counted again, and this may of course inflate the counts for query words that remain. The distribution of search terms is pretty close to the Zipf distribution. More than 70% of the search terms only appear once in the log. The joint ratio is 9/91: the most popular 9% of the search terms account for 91% of the instances of a term in a query, whereas the less popular 91% of the terms account for only 9% of the instances. A full half of the search term instances in the data are repetitions of the top 0.05% of the search terms.

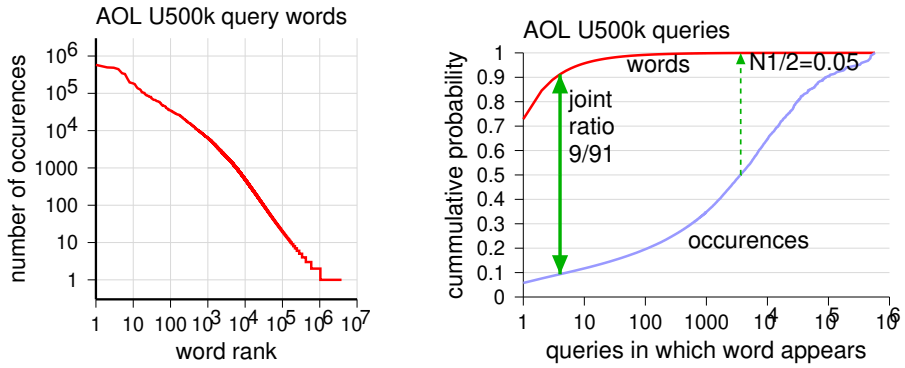


Figure 9.30: *Distribution of search terms approximates a Zipf distribution with a heavy tail.*

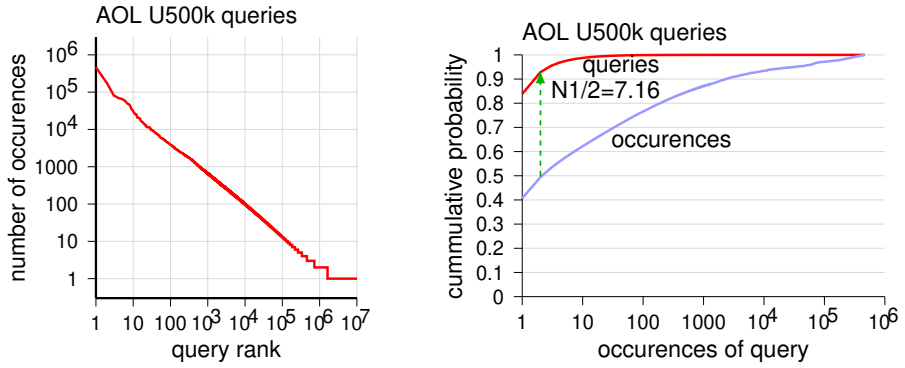


Figure 9.31: *Distribution of queries follows a Zipf distribution with a heavy tail, albeit most queries only appear once.*

The most popular terms are simply the most popular words in English, with "of", "in", "the", "and", and "for" occupying the top slots. High ranks are also achieved by some popular navigational queries: "google", "yahoo", and "ebay" occupy ranks 8, 15, and 20, respectively. Other high-ranking terms are "free" (rank 9), "lyrics" (13), "school" (16), "florida" (22), "home" (26), "pictures" (29), and "bank" (30). The notorious "sex" is only rank 44. Thus, not surprisingly, the search vocabulary is distributed differently from normal English usage. Co-occurrence is largely limited to phrases, such as "real estate", "red hat", "clip art", and "puerto rico". However, some words, mainly adjectives like "free", tend to co-occur with several other terms [623, 375, 373].

A somewhat different picture is seen if one considers complete queries rather than individual terms. Complete queries appear to follow the Zipf distribution much more closely, as shown in Figure 9.31. However, although the distribution has a heavy tail, the tail elements do not dominate the mass. On the contrary, nearly 84% of the unique queries appear only once, and together they account for more than 40% of all queries. Thus there is not much mass-count disparity, and the joint-ratio is not even defined.

Another difference is that simple English words tend not to appear in very popular queries, which typically have only one search term (exception: "bank of america" at rank 18). The most popular queries are predominantly navigational, with searches for Google being by far the most common. However, these searches may be done using different query strings that reflect the Internet naming system, including "google" (top rank), "google.com" (rank 6), "www.google.com" (10), and also the misspelled "goggle" (52).

Other top navigational queries are also Internet companies and also have several variants, including "ebay" and "ebay.com" (ranks 2 and 15); "yahoo", "yahoo.com", and "www.yahoo.com" (3, 5, and 11); "mapquest" (4); and "myspace.com" and "myspace" (7 and 8). Slightly lower down we start seeing navigational queries targeting more traditional companies and institutions, such as "walmart" (22), "southwest airlines" (31), and "target" (33). These typically have only one main variant, without Internet address trimmings.

Other popular queries are a combination between being informational and transactional: they seek a website that can provide the desired information, rather than seeking the information directly. Examples include "weather" (14), "dictionary" (21), "lyrics" (64), and "horoscope" (86). Then there are quite a few nonsense queries, many of which are just parts of Internet addresses, such as "http" (13), ".com" (20), "www." (49), but also "m" (37) and "my" (53). The usual suspects are ranked lower: "porn" is 68, "sex" is 76, and the first celebrity is "whitney houston" at 225 (but "american idol" is 17).

All these queries are omnipresent, and in the three-month AOL log they are popular in each and every week. Other queries are popular only in a certain week, typically due to a specific news story involving some celebrity or sports event. Examples include "nfl draft" (which occurred on 29–30 April 2006) and "kentuky derby" (6 May 2006).

The misspelling of Google mentioned earlier is by no means an isolated incident. Google have published a list of 593 variations on the name "Britney Spears" that were used at least twice (by two different users) within a three-month period — with all of the variants in the first name (AskJeeves had a list of 1126 variations in both names, but did not include data about how many times each one appeared). The correct spelling was the most popular, accounting for 77% of the cases. But misspellings accounted for no less than 23% of all instances. And as may be expected, the popularity distribution of the different misspellings is heavy-tailed, with a nearly straight LLCD.

Quite a few papers analyzing web search workloads have been published in recent years. They typically provide data such as that discussed earlier, from a variety of search engines, often using proprietary data. Interesting observations include the following:

- Queries are typically short, with an average of two to three terms (2.21 for Excite in 1997 [376]; 2.35 and 2.92 for AltaVista in 1998 and 2002, respectively [623, 375]; 2.4 for AlltheWeb in 2001 [373]; 2.34 for both the 2006 MSN data [758] and the 2006 AOL data; and only 1.67 for Google searches at a German university in 2006 [390]).
- The length depends on the type of the query. While 42% of regular AOL queries have only a single word and the average is 2.29, the 1% of queries that are posed as questions ("is 42 old" and others starting with why, what, where, when, who,

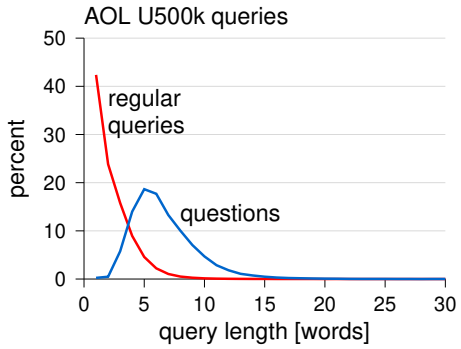


Figure 9.32: *Distribution of query lengths. Note that questions form only 1% of all queries.*

whom, whose, how, which, do, did, does, is, or are) are, on average, 6.65 words long (Figure 9.32).

- There has been a continuing decrease in search for sexual content and an increase in search for general information [646, 375, 647].
- Different topics are popular at different times of the day: music peaks at 3–4 AM, porn at 5–6 AM, and personal finance at 8–10 AM [63].
- Some queries may exhibit large fluctuations in popularity from one hour to the next. For example, this happens when a certain news story breaks and many people search for related information at the same time [63].
- There is an increasing use of search engines as a navigational tool, as was indeed reflected in the results we shown earlier [375, 758].
- Users occasionally search for the same thing again and again over extended periods of time [681, 704], possibly using search as an alternative to keeping a bookmark.
- Advanced search features such as Boolean operators are not used much, and users often make mistakes in using them [376].
- Search sessions are typically short, containing only one or a small number of queries, with few modifications. And in most cases the user does not look beyond the first page of results [623, 758].

Despite the last item, an important observation is that queries sometimes do not come alone, and a sequence of queries is performed to accomplish some search goal [385, 185]. This may be needed because the first query did not uncover the desired information, in which case it is reformulated and submitted again. Search goals, in turn, are typically elements of a larger search mission, leading to a hierarchical structure. In addition, queries belonging to different missions may be interleaved as the user shifts from one topic to another and back again.

Practice Box: Stream Algorithms

Results such as those presented in this section are based on offline analysis of a sample of the queries received by a search engine over a period of between one day and three months. But it is also desirable to be able to collect such statistics online, as things happen, and based on all the queries that arrive; for example, this is important in identifying new trends and the emergence of hot new topics (e.g. [550]). Algorithms that perform such analysis are called stream algorithms, because they operate on a continuous input stream.

The chief requirement for stream algorithms is that they operate in constant time and space, so as to be able to handle each stream element (in our case, each query) before the next one arrives. For example, the following algorithm identifies the k distinct elements that each appear in the stream more than $\frac{1}{k+1}$ of the time, if such elements exist:

1. Start with k counters initialized to zero.
2. Associate the counters with distinct stream elements, and use them to count the number of appearances of the first k unique elements in the stream.
3. Subsequently, for each arriving element of the stream, compare it with the k stored elements and do the following:
   (a) If the newly arrived item has been seen before, simply increment its associated counter.
   (b) If it is not identical to any of the k stored items, decrement all their counters.
   (c) If any counter reaches zero, replace its associated element with the newly arrived element.

Note, however, that if there are less than k items (or even none) that meet the threshold of appearing $\frac{1}{k+1}$ of the time, the items that will be stored at any given instant are not necessarily the most popular ones.

A nice survey of stream algorithms and their capabilities and limitations is provided by Hayes [323].

End Box

Evaluating the performance of search engines is a special case of evaluating information retrieval systems. To evaluate such systems test collections are needed, and these should rightly be considered part of the system's workload. Such collections are considered in Section 9.5.2.

9.4.7 Media and Streaming

Some web servers are dedicated to streaming data to their clients. Examples include media servers and news servers. Again, they have special workload characteristics.

Yu at al. present an analysis of a video-on-demand media server from China [747], and Tang et al. present a similar analysis of two media servers belonging to HP [675]. Among their findings are the following:

- The workload exhibits a daily cycle, as expected. The peak activity is in the evening hours, after work. In the Yu study this cycle was a very strong effect, whereas in the Tang study it was relatively minor. This difference may be a result of using data from a server belonging to a multinational corporation, rather than from a server with a more localized user base.

- The vast majority of sessions are aborted before the movie being watched ends. Yu also found a small fraction of sessions that were longer than the movie, due to using features such as pause and rewind. Tang et al. found that the probability of watching the movie to its end strongly depends on the movie length, with the fraction of full views declining as a power law of the total length.

  As for the aborted sessions, most of them are very short, lasting just a few minutes; specifically, the data indicates that more than 50% of the sessions aborted within 10 minutes (Yu) or 2 minutes (Tang). These results may be related to the fact that the analyzed services were free (in Yu's case, free to subscribers), and paying customers may have a lower tendency to abort. Nevertheless, the results have important consequences. First, they indicate that users frequently sample a movie before deciding whether or not to watch it. And second, caching the first few minutes of all movies in easily accessible storage can serve a large fraction of the requests.

- Yu found that popular movies tended to be aborted more often and earlier than less popular movies. Possibly their popularity caused many users to check them out, but most then decided not to watch them.

- In the Yu study, the popularity of different files was well described by a Zipf-like distribution, except for the least popular ones, which are accessed less than expected. This implies that only the tail of the distribution obeys a power law [690]. In the Tang study, a good fit to a Zipf-like distribution was obtained after applying a k-transform, which scales the axes so as to treat files in groups rather than individually.

- On different days the skewness of the popularity distribution was different, with a normally distributed parameter.

- The churn for the most popular videos is significant, with 2–3 of the top 10 replaced each day and 5–8 replaced each week. This churn is at least partly due to the introduction of new content. When looking at a larger set things are more stable: only about 15 of the top 100 are changed each day, and between 25–30 each week. A byproduct of this behavior is the generation of temporal locality. Different files are popular on different days, and the accesses to each file are concentrated in the period when it is popular, and not spread out evenly across time.

- Popularity is strongly affected by a movie's appearance in the site's top 15 list or the list of (typically new) movies recommended by the system.

Based on the collected data, Tang et al. devised the MediSyn workload generator for streaming media servers [675]. It operates in two steps. First, it generates the population of files on the server, and assigns each file its attributes according to the model distributions. Then, in the second step, it generates a sequence of accesses to the generated files.

Another workload generator is Genius by Costa et al. [151]. The emphasis here is on interactive user behavior. Media objects are classified into several classes that are

typically used in different ways. Each class then has a characteristic pattern of user sessions, and the sessions in turn are composed of sequences of interactive requests (such as pause, fast forward, etc.).

Two interesting complementary studies concern the YouTube service, in which users can upload and view short videos. Cha et al. studied YouTube's workload by crawling the site's indexes several times and collecting data about the popularity and rating of different videos [114]. Gill et al. collected data about all the YouTube traffic from one specific location, namely the University of Calgary [285]. These studies show that the popularity distribution of different videos corresponds to the Zipf distribution. This is explained by the generative model of preferential attachment — videos that are highly popular are prominently displayed by the site, leading additional users to check them out. However, when a global view is taken, the number of requests for the topmost videos seems to be truncated rather than growing according to Zipf's law. This is explained by the finite size of the user population.

Other analyses of similar workloads include that by Johnsen et al. of a news-on-demand service [382].

A special type of streaming data comes from voice over IP services such as Skype. The problem is that Skype uses proprietary protocols, encryption, and obfuscation to prevent being identified and blocked. This naturally makes it difficult to analyze its generated traffic. However, it is possible to observe and characterize the traffic generated under laboratory conditions. Bonfiglio et al. do so and find dynamically varying behavior: Skype may use different codecs and adjust to changing network conditions [79]. When analyzing Skype user behavior on a campus network, the expected characteristics are found: a daily cycle of activity, and paid calls (to regular phone numbers) tend to be much shorter than free calls between two Skype clients.

Interestingly, media-based workloads may also characterize certain client devices, and not only servers. For example, a study of hand-held devices found that their Internet activity was dominated by playing multimedia files and application downloads [460].

9.4.8 Peer-to-Peer File Sharing

Web workloads operate in a client-server setting: users use browsers to make requests for web pages, and these requests are served by the web servers. An alternative model is the peer-to-peer approach, which is completely symmetric, with every node operating as both a client and a server [437]. This approach has gained widespread popularity in the form of file sharing services, which allow users to share files with each other. These services have come to dominate Internet traffic, due to the combination of large-scale data transfers (large media files) and voluminous control activity (pinging peers in the network and using flooding to perform searches). As a result, changes in such applications may cause changes in the overall observed traffic [535, 279].

A special characteristic of peer-to-peer traffic is its daily cycle. For most applications, the peak usage (and load) occurs during the day or perhaps during the evening hours (after work). But for peer-to-peer applications the peak occurs during the small hours of the night, from 3 AM to 8 AM [279]. The reason is that these applications

often operate in the background to download large media files. In doing so they exploit available bandwidth during the night.

As mentioned earlier, the files typically shared are multimedia files, in particular music, video clips, and movies. Such files have the following important attributes [309]:

- They are immutable: once a file appears in the system, it never changes.
- There are relatively few distinct files. For example, the Internet Movie Database (http://www.imdb.com/stats) listed roughly 2.8 million titles as of February 2014, of which 1.7 million are TV series episodes and 304,000 are feature films. Although these are large numbers, they are small relative to the number of users who download such files.

In addition, the file sizes are generally very large and display significant mass-count disparity.

The fact that files are immutable has a strong effect on the popularity distribution as seen on the system. Assume the real popularity is according to the Zipf distribution. However, once users download a file, they will never download it again even if they use it many times. Therefore the number of downloads of the most popular files will be much lower than the number of times they are used, i.e., much lower than predicted by the Zipf distribution. (Note that this assumes downloaded files can be stored; thus this argument does not apply to video-on-demand systems [747]).

The fact that the number of distinct files is limited implies that users may run out of new files they want to download. This helps explain the dynamics of downloading patterns. Such patterns exhibit a strong locality of sampling: at any given time a few files are the most popular, but this group changes with time [309]. Essentially, the most popular files for download are always recent releases. When they are new they are widely downloaded, but once they have been downloaded by a significant fraction of the interested user base, the number of downloads decreases. This can be viewed as each new release leading to its own flash crowd effect.

Another type of locality characterizing peer-to-peer systems is the skewed level of service provided by different peers: a small number of peers respond to most queries, while a third to two thirds are "free riders" that do not contribute any service [14, 312]. In particular, one study showed that the top responding peer could respond to nearly half of all queries. However, the global set of all results was spread out over many more peers. The top source could only provide about 3% of all possible results to a query, and even querying 20% of all peers generated only about 60% of the results [312].

A problem with the study of P2P file sharing is that current services attempt to disguise themselves [392]. The original file sharing service, Napster, used a specific port and was thus easy to identify (see Figure 9.21). Services such as Gnutella and KaZaA may use different ports and even masquerade as using an HTTP protocol. Although these services are still reasonably easy to detect, they could be missed if the data collector is not aware of the need for special handling [455]. For example, a large part of the "other TCP" and "HTTP" traffic shown in Figure 9.21 is probably file-sharing traffic. Approaches to detecting P2P traffic include looking for specific strings in the packet

payload (which, however, faces severe legal/privacy/technical problems), or looking for tell-tale usage patterns such as using both TCP and UDP with the same IP address and port [393]. Interestingly, different file-sharing applications seem to be popular in different locations. For example, eMule is heavily used in Italy, whereas BitTorrent is much more popular in Hungary and Poland [279].

9.4.9 Online Gaming

Online gaming is a booming industry, with game servers supporting thousands of interactive users simultaneously. The interactive real-time nature of the play places stringent requirements on the infrastructure. Conversely, to evaluate the suitability of the infrastructure one needs to know about the characteristics of the gaming workload — which is subject to the high volatility of the field.

A set of observations regarding gaming workloads include the following:

- The relative popularity of different games exhibits a Zipf distribution [116].
- The distribution of player session lengths is exponential or Weibull, with many short sessions [327, 116]. It does not have a heavy tail. When looking at the player population, player interarrivals are found to be heavy-tailed.
- Client packet interarrival times are proportional to the machine's CPU speed: faster machines reduce the rendering delay associated with data received from the server, and then respond with their own data more quickly [81]. Thus different clients will have different communication rates.
- First-person shooter games exhibit faster movement of avatars relative to other virtual environments [503].
- The network bandwidth requirements of online games are quite modest at around 50–200 Kb/s per client [39].
- Game players are human (OK, not all of them... [421]). Thus gaming workloads exhibit a strong daily cycle [327, 116]. They also exhibit a weekly cycle, which is different from other workloads: there is more activity on the weekend instead of less [327].
- Those players that are not human may sometimes be identified by identical repetitions of activity patterns (e.g., repeated traversal of exactly the same path) [494].

9.4.10 Web Applications and Web 2.0

Modern web applications often exhibit behaviors that differ significantly from the traditional client-server model of the original static web, in which relatively small requests were sent from clients to servers, and most of the traffic was composed of files sent from the servers back to the clients. For example, in many web services the content is provided by users rather than by the service. Examples include blogs, services for sharing pictures and videos, and sites such as Wikipedia. Thus there is a noticeable component of uploading content, which is largely absent from conventional web servers. Another

example is the extensive use of AJAX technology, which allows for asynchronous downloads that do not subscribe to the request-response pattern. For example, in Google maps nearby tiles of the map are downloaded in the background and cached in case they will be needed, thus ensuring faster response when the user pans the map. This leads to heavier and more bursty traffic [595].

Wikipedia is perhaps the poster child of user-generated content, representing an estimated 41 million hours on work on the English version and 102 million hours on all versions combined as of 2012 [282]. An analysis of edit sessions reveals that they last on average for around 33 minutes, but the median is only 10 minutes, testifying to a highly skewed distribution. Interestingly, the distinction between short and long sessions extends to users: there is only partial overlap between the group of users who contribute the largest number of edits and the group that contribute the most editing time.

In addition to having user-generated content, blogs are also characterized by a many-to-many access pattern: bloggers' posts are read by many readers, and readers comment on many posts [192]. This is different from conventional web access patterns. Moreover, because blogs refer to each other, they may cause "epidemics" in which certain posts receive disproportionate popularity [13]. This even affects the daily cycle of activity: the peaks of daily activity become more varied than normal, because they depend on the specific popularity of items at different times [192]. Thus the peak in each day can be considered to be a combination of base activity and some number of unique flash crowds.

Other characteristics are more in line with conventional web workloads, including the heavy-tailed distribution of transfer sizes and the diurnal cycle of activity [192, 377]. User activity is heavy-tailed, but popularity is not quite a Zipf distribution. Rather, it tapers off (like the distribution of instances of words in this book), possibly indicating that the number of blog postings is the limiting factor. Distributions of session lengths, inter-post times, and inter-comment times are skewed, but not heavy-tailed.

Another type of Internet-based infrastructure for user interactions is social networks such as Facebook. As may be expected, interactions on Facebook are often well described by power laws. For example, The in-degree and out-degree distributions of connections are heavy-tailed, and a small number of core "power users" dominate interactions [492]. These core users also generate most of the traffic [511]. But interestingly, the precise patterns depend on the type of application that is being used. Social applications (such as sending virtual hugs) correspond to the underlying friendship connections on Facebook. But gaming applications transcend existing friendships and create new ad-hoc connections. Another interesting finding is that Facebook usage is often not interactive — it seems that many users are actually not connected all of the time. Thus the average response time in three Facebook applications was no less than 16.5 hours.

The recent growth in cloud computing and online web applications promises new types of workloads with new characteristics and requirements — a combination of desktop computing and Internet communications. Initial analyses of cloud workloads have been published by Mishra et al. [491], Reiss et al. [563], and Di et al. [177]. Di et al. characterized the workload on a Google data center [177], and compared it with the workload on computational grids. This analysis showed that the jobs on Google's cloud

infrastructure were quite different from those on grid systems. Specifically, their runtime distribution is much more skewed, with the vast majority of jobs being very short (80% less than 1000 seconds), while the longest jobs are several weeks long. As a result job runtimes have a joint ratio of 6/94, and the median-median distance is 23 days. Typical job interarrival times on Google are also substantially shorter than on grids. Both of these characteristics bear witness to the largely interactive nature of cloud workloads, as opposed to the computational nature of grid workloads.

Another interesting development that is related to the move of interactive user activity to the web is the correlation between different modalities. For example, one can find correlations between blogging activity and search activity on the same topic [15]. This has implications for predicting user behavior.

9.4.11 User Types

The Internet is perhaps the locus of the widest differences between different demographic groups of users. For example, in practically all families, children and even more so teenagers engage in much more Internet activity than their parents and grandparents. Thus detailed generative models may need to take different user types into account.

A characterization of user types has been performed by García-Dorado et al., as part of a study of ISP traffic patterns [279]. Based on the volume of traffic and the diversity of applications used, they partition users into three classes:

- Old-fashioned users use the Internet only for email and simple web browsing. Up to about a third of users exhibit such behavior, generating few megabits of traffic per day.
- The biggest group of users are the normal users. These users mainly use social networks (like Facebook) and streaming services (like YouTube), generating tens to hundreds of megabits of traffic per day.
- A relatively small fraction of the users are advanced users. These are the users who use many different types of applications (social networks, file sharing, file hosting, etc.) and generate large volumes of traffic, typically measured in gigabits per day.

9.4.12 Feedback

As in other types of workloads, feedback effects also exist in network traffic. But here they exist at different levels at the same time: the underlying TCP protocol used to move data, the request-response style of many Internet applications, and the think and take action cycle of human users [426]. These feedback effects are extremely important in shaping the traffic, and ignoring them may lead to erroneous evaluation results [731, 265].

At the lowest level of packet transmission, an important source of feedback is the TCP congestion control algorithm. Congestion control was introduced into TCP to prevent traffic buildup as a result of positive feedback under loaded conditions [364, 84].

The problem is that when the load is high, queues at the routers may overflow. When this happens the routers are unable to store the packets and forward them on the next lag of their path. Therefore they simply drop such packets. If the packet is part of a TCP flow, the dropped packet will naturally not be acknowledged by the receiver. The sender will then time out and retransmit the packet. Thus packet loss as a result of load will cause the load to grow even more, leading to even more packet loss and a dramatic reduction in the effective achieved bandwidth.

TCP battles this problem by introducing a stabilizing negative feedback effect, whereby congestion leads to *reduced* load. This is done by reducing the size of the transmission window when too many packets are lost. The transmission window is the set of packets that have been transmitted but not yet acknowledged by an "ack" from the receiver. To achieve the maximal bandwidth possible, the transmission window should include enough packets to account for the round-trip time until the acks are received. But when the links are congested and packets are lost, it is impossible to achieve the maximal bandwidth, and it is better to throttle the transmission of additional packets.

The implication of TCP congestion control on workload modeling and use is that it is dangerous to use packet traces to drive network simulations, because such traces are "shaped" by the conditions that existed when they were recorded [265]. Insted, traffic should be modeled at the source, that is, using a generative model. The feedback will then be realized by including an implementation of the TCP congestion control behavior in the simulation.

Interestingly, it has been argued that this behavior of TCP may also be responsible for the self-similarity observed in network traffic [622, 736]. However, this claim is in dispute, based on the observation that the indicators of self-similarity seen in a single TCP flow only apply for relatively short time spans [311, 260].

Additional forms of feedback occur at the application layer. In many cases the communication follows a client-server pattern, in which the client makes a request, the server responds, and this is repeated many times. For example, a browser may request a web page, and the server returns it. But the requested page may include embedded graphic objects, thus triggering additional requests for these objects. Naturally these additional requests are contingent on receiving and parsing the page that was requested before, leading to a feedback effect: when these requests are sent depends on network conditions.

Above all this is the feedback associated with user actions. Networking that is interactive in nature, such as browsing on the web, is typically done one page at a time. Users scan the retrieved page before clicking on a link to go to the next page. Thus the generation of additional requests is again conditioned on the completion of previous ones, leading to a feedback effect. Importantly, this feedback effect may also reduce congestion when users abort their downloads or even their sessions when performance deteriorates beyond what they are willing to tolerate [504, 688, 552, 744].

9.4.13 Malicious Traffic

In addition to legitimate traffic, the Internet also transmits malicious traffic such as worms and denial-of-service attacks. Modeling such malicious traffic is needed in order to evaluate systems that are supposed to protect against it, such as firewalls and network intrusion detection systems (NIDS).

An example of a system that generates malicious traffic is MACE (Malicious Traffic Composition Environment) [642]. It takes a modular approach, in which each attack vector is constructed by composing three elements:

1. An exploit model, which specifies the characteristics of the packets that administer the attack; for example, a SYN flood specifies that packets be sent with the SYN flag set.
2. An obfuscation model, which specifies small variations (e.g. in IP fragmentation) that are often used in an attempt to fool NIDS.
3. A propagation model, which specifies how targets are selected; for example, this can be a serial scan of IP addresses and a random selection of ports.

The generated traffic is composed of one or more such vectors, in combination with legitimate background traffic generated by some other source.

An additional parameter that may be important is the source address. Distributed denial-of-service (DDoS) attacks are typically propagated by botnets comprising thousands of computers around the world. As a result the distribution of source addresses in such attacks has two distinguishing properties, that are markedly different from what is observed in normal legitimate traffic [389]:

- The distribution of sources is very uniform. In normal traffic the addresses from which requests come can be clustered by address prefixes. This represents some locality in the distribution of sources. Botnets lack such locality.
- The distribution of sources is novel. Under normal workloads, the request sources also display temporal locality. In other words, the requests seen today tend to come from the same places (and specifically, the same address clusters) as requests seen yesterday or last week. Again, botnets employ computers from around the world, and therefore most of them will be from addresses never seen before.

Importantly, these two characteristics allow DDoS attacks to be distinguished from flash crowd events.

Botnets are used not only for attacks, but also to propagate spam. Such traffic also has its unique characteristics. To investigate that traffic, one study actually purchased bogus advertisement impressions from several botnet operators [650]. These turned out to be quite different from each other, especially in the pattern in which the generated load was distributed over the day.

9.5 Data-Centric Workloads

The data being processed by computer systems is no less important than the operations being performed on this data. This is especially true in database systems, information retrieval systems, and, more recently, systems involved with "big data". In all these cases the data is a basic component of the workload, and should be included in the workload model.

9.5.1 Database Systems

Databases can be viewed at several levels: the enterprise level with its business logic, the level of the database management system and its tables and SQL queries, and the level of the basic I/O operations [11]. Here we focus on the database level.

As in other domains, queueing models of database systems are often based on unrealistic workloads that are chosen for mathematical convenience. Thus arrivals are assumed to follow a Poisson process, or else a closed model is used, and objects within the database are often assumed to be uniformly accessed [526]. However, real workloads are more complex.

A detailed analysis of traces from 10 large-scale production systems running IBM's DB2 in the early 1990s was conducted by Hsu et al. [348]. One of their interesting results is that significant inter-transaction locality exists, indicating that in real workloads transactions are not independent of each other. Another finding is that there is significant use of sequential accesses, which allows prefetching to be employed. Sequentiality is a consequence of long-running queries that examine a large number of records, e.g. in order to perform a join operation.

Other studies of database workloads indicate that OLTP (online transaction processing) workloads are characterized by a large memory footprint, combined with a small critical working set [448], and by the reduced benefit from microarchitectural optimizations [398]. In addition, index searching in OLTP workloads requires a different cache design [60].

The evaluation of database systems is dominated by the use of TPC (Transaction Processing Performance Council) benchmarks. Although these are benchmarks in the sense that they are well defined and include a builtin metric for evaluating systems, they can also be interpreted as generative workload models, which can be scaled to enable the evaluation of databases of different scales. The benchmarks defined by TPC are the following. The changes over the years also show the progress that was made in characterizing the workloads and the interaction between them and the system.

TPC-A: a microbenchmark using a single update-intensive transaction to load the system. This benchmark was defined in 1989 and deemed obsolete in 1995.

TPC-B: a microbenchmark designed to stress test the core of a database management system, with significant disk I/O activity, to see how many transactions per second it can handle. This was defined in 1990 and is obsolete since 1995.

TPC-C: the most enduring TPC benchmark, defined in 1992 and updated several times since then. This benchmark simulates a complete OLTP environment characteristic of a wholesale supplier. The model is a set of operators who execute transactions such as order entry and monitoring, updating and checking stock levels, and recording payments. The relative frequency of the different transactions and their sizes (e.g., how many items are ordered) are modeled based on realistic scenarios. The metric for performance is the number of order-entry transactions per minute. The entire benchmark is structured around a number of warehouses, with each warehouse serving 10 sales districts (terminals) and each sales district serving 3000 customers. The scale of the benchmark is changed by adding warehouses.

TPC-D: a decision-support benchmark composed of 17 complex and long-running queries, defined in 1995. The definition proved problematic as vendors added automatic summary tables to their databases, leading to much faster processing of the queries themselves. This benchmark was therefore replaced in 1999 by TPC-H and TPC-R.

TPC-H: a derivative of the decision-support TPC-D benchmark, with complex queries, but explicitly disallowing precomputed optimizations. Defined in 1999.

TPC-R: also a derivative of the decision-support TPC-D benchmark, in which it is assumed that the domain is well known and therefore extensive precomputed optimizations are assumed to be realistically possible. Defined in 1999 and obsolete as of 2005.

TPC-W: a web e-commerce benchmark, which measures performance by completed web interactions per second, subject to a response time requirement for each interaction [278]. The scenario is an online book store, with the system comprising web servers connected to a back-end database, as well as image servers and caches. Defined in 1999 and obsolete as of 2005.

TPC-App: a benchmark for web applications, including an application server supported by a web services framework. The workload simulates business-to-business transactions using XML documents and SOAP for data exchange. Defined in 2004 and now obsolete.

TPC-E: a new OLTP workload defined in 2006. This benchmark is based on a brokerage firm model and can be scaled by changing the number of customers.

TPC-DS: a new decision-support benchmark defined in 2012. It is based on a retail scenario with sales in stores, from a catalog, and on the web. It combines both queries and data maintenance activities.

TPC-VMS: the newest benchmark, designed to test the effect of running multiple database applications in virtual machines on the same physical platform. The benchmark requires three copies of one of the other benchmarks (TPC-C, TPC-E, TPC-H, or TPC-DS) to be used, and the result is the slowest one.

To enable measurements of database performance, these benchmarks need to specify the structure and contents of the database. The TPC-DS benchmark, for example, defines a

database schema with 24 tables [689]. These tables can be populated at seven different scaling levels, leading to database sizes that range from 100 GB to 100 TB. The bulk of the data is contained in the tables that describe the sales through the different sales channels: the web sales table grows from 72 million lines at the smallest scale to 72 billion lines at the largest scale, the catalog sales table grows from 144 million to 144 billion lines, and the store sales table grows from 288 million to 288 billion lines. Each of these tables has a corresponding table of returns that is one tenth as large. Other tables are smaller and also grow at a sublinear rate. For example, the table of catalog pages grows from 20,400 lines to 50,000 lines, the table of stores grows from 402 to 1902, and the table of call centers grows from 30 to 60. A few tables (e.g., the table of shipping modes) do not change with scale. Tools are provided to generate the actual data and the queries that are needed to run the benchmark.

Given that real database traces are hard to come by, some studies use the workloads produced by the TPC benchmarks as a proxy. For example, TPC-C and TPC-B have been used to characterize OLTP workloads [661, 760, 755], and TPC-D and TPC-H have been used to characterize decision support [760, 755]. TPC-C and TPC-H have been used to provide training data for a classifier that can be used as the basis for adaptive systems that tune themselves to different workload types [209].

However, using TPC in this way should be avoided if possible. Hsu et al. performed a direct comparison between production database traces and traces obtained by running the TPC-C and TPC-D benchmarks [349, 348]. It indicated that real production data is more diverse, being a mixture of many types of both small and large queries. The benchmarks, in contradistinction, tend to be more uniform. TPC-C, in particular, is based on small transactions that access random data, leading to access patterns that are markedly different from those typically observed in production systems. There are also differences in the distributions of object sizes referenced and the degree of concurrency exhibited by the workloads and benchmarks.

This study focused on the patterns of queries. It is also important to examine the data stored in the database. How many tables are there? How many fields do they have? And what is the distribution of values in the different fields? As noted earlier, the TPC benchmarks provide detailed specifications, but they are by definition synthetic general scenarios and cannot be counted on to reflect the diversity found in real production systems.

An example where studying the data is important is given by Wolf et al. [737]. They studied the parallel implementation of join operations, in which the rows of two tables are unified based on a shared field value. The implementation is based on hashing the shared field values and distributing tuples among the available processors based on these hash values. Thus each processor receives matching tuples from the two tables and can perform part of the join independently from the others. This works well if the values are uniformly distributed. However, real data tends to have skewed distributions, which causes load imbalance between the processors.

Another example is given by Zuck et al. [766]. The context is a study of how compressing the data to save space affects the performance of OLTP workloads. Random data as generated by the TPC-C benchmark compresses badly, and its volume is reduced

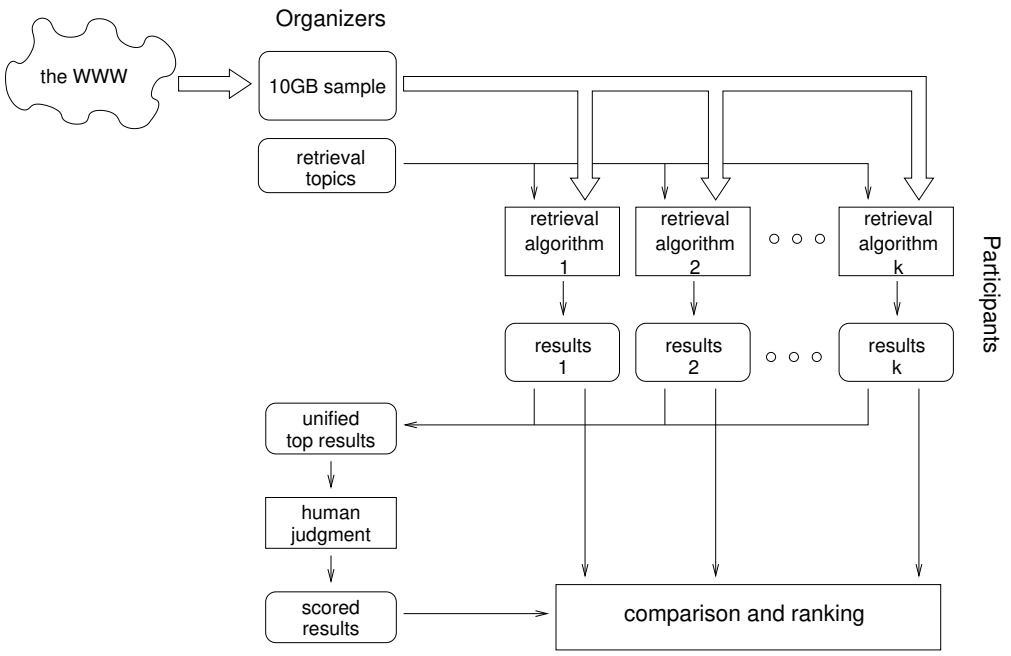


Figure 9.33: *The TREC experimental procedure (based on [714]).*

by only 13%. This does not reflect the relatively high compressibility of real databases, where compression may reduce the data volume by 76%. As a result, the TPC-C data is misleading and not usable for such a study.

9.5.2 Information Retrieval

As mentioned earlier, search engines are a special case of information retrieval systems. To evaluate such systems, test collections are needed. Such collections contain a corpus of documents (e.g., the results of a web crawl), and a list of topics that form an abstraction of user queries. Search mechanisms are then evaluated by what fraction of the relevant documents they retrieve for each topic (recall), and what fraction of what they retrieve is indeed relevant (precision) [713].

To provide a realistic and challenging setup for such evaluations, the corpus on which the search takes place must be huge. The problem is that it is then hard to know in advance what the correct results are, and therefore it is impossible to evaluate the recall.

To solve this problem TREC (Text REtrieval Conference) competitions have been set up [716, 714]. In these competitions, the organizers create a large sample of web documents and define a set of topics that reflect the desired information (Figure 9.33). Participants then apply their information retrieval algorithms and produce ranked sets of results. The organizers then take the top 100 results from each participant for each topic, and create a unified pool that is assumed to contain all the relevant documents for that topic. These documents are then evaluated for relevance by human experts, and the resulting scores are used to evaluate and rank the retrieval algorithms. Thus part of the

workload definition (the identity of the desired documents) is only known retroactively, after the experiment takes place.

Although this approach has been effective in evaluating information retrieval systems [714], it is limited to static situations in which the "correct" set of retrieved documents is well defined in advance. A more complicated situation is when the system is adaptive, and thus user behavior and expectations cannot be abstracted by static topics and their related sets of documents [715].

9.5.3 Big Data

"Big data" is a catch-all phrase for emergent systems that are characterized by the need to handle vast amounts of data at very high rates, way beyond the capabilities of conventional databases and mainframes. Thus using a distributed infrastructure is necessary. Such data storage and processing needs are becoming ubiquitous, with examples including the following:

- Using data about personal buying patterns in retail and commercial settings, often based on credit card transactions, to optimize advertising and sales.
- Using data relating to logistics and utilities, e.g. based on distributed sensing, to optimize operations.
- Making health care records and medical imaging data available to doctors where and when they need it, regardless of where and when they were collected.
- Storing and analyzing scientific data, such as data from life science genome projects, astrophysics observatory imaging, or data from the sensors of particle physics experiments.
- Storing and analyzing data from surveillance and security cameras.
- Storing, classifying, prioritizing, and serving data from social media and communications services such as Facebook, Twitter, and YouTube.

While some of the data involved is well structured as in conventional databases, other data may be semi-structured (such as HTML web pages with various tags) or even unstructured text.

Importantly, several data types and data manipulation patterns are common to a number of these domains. These include the following:

- Analyzing large image collections.
- Indexing large text collections.
- Traversing large graph structures.
- Filtering a high-rate incoming data stream.
- Maintaining large key-value stores.
- Performing MapReduce computations.

These patterns combine with four main features of the data, and together show why conventional databases cannot be used effectively [280]:

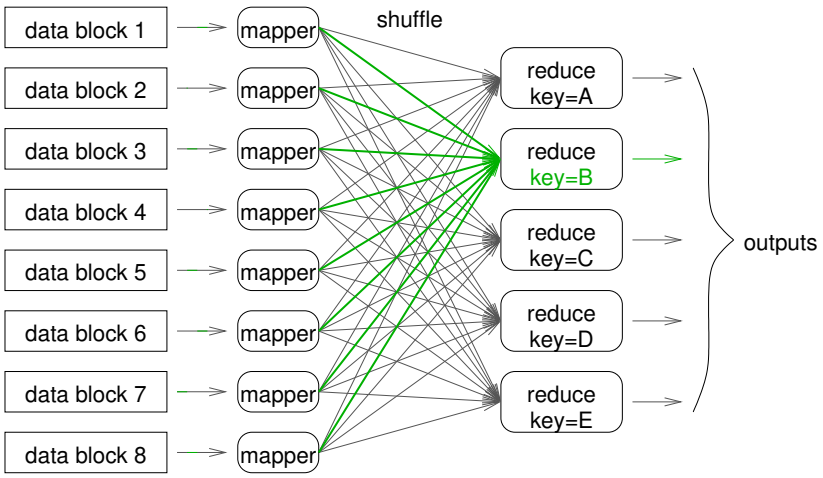
- The data has a vast volume, often on the order of petabytes. Thus, big data systems require more I/O resources than other system types to be balanced [337].
- Data is unstructured and comes from a variety of sources that use different data formats.
- Often the data has real-time characteristics, meaning that the data loses value if it is not processed rapidly.
- The data may have correctness and validity problems.

Data about actual workload characteristics in big data systems is sparse, and moreover the entire field is in constant flux. But some datasets have been analyzed, and insights have been shared. For example, it has been noted that data querying in very large datasets tends to be simpler than the queries that may be used in databases. SQL is not used, and in particular, joins are avoided. Instead, querying is based on simple word searches [159]. In fact, much of the processing is done using the MapReduce paradigm. To ease this, SQL-like interfaces to MapReduce have been designed, such as Hive [685].

Background Box: MapReduce

MapReduce is a parallel programming paradigm developed by Google [167, 168]. It is aimed at applications that need to scan vast amounts of data and extract the important and meaningful information from them. The idea is to partition the work into mapper tasks and reduce tasks. The mapper tasks are independent of each other and do the heavy lifting of going over all the data. To do so the data is partitioned into blocks, and each mapper works on one block. Importantly, this allows the mappers to run where the data is stored, thus reducing communication overhead. The output of the mappers is expressed as a set of key-value pairs that represent the information that was extracted.

The reduce tasks then combine the key-value pairs from the mappers to create the final output. In essence, this amounts to performing some merging operation on all the different values that correspond to each key. To do this efficiently, all the pairs with the same key from all the mappers are collected at the same location. This rearrangement of the data is called the shuffle stage. The reduce tasks are then again independent local operations, just as the mappers are.



The importance of MapReduce is that many real data-processing needs are expressible in this framework by using appropriate mappers and reducers. Thus by writing just a mapper (which operates locally on some data, independent of other mappers) and a reducer (which combines given data inputs) one can implement a parallel application that processes vast amounts of data — and a large number of applications can indeed be formulated in this way [168]. All the issues related to distributing the data and scheduling the computations are handled by the underlying system.

Perhaps the most widely used MapReduce platform today is Hadoop. This is an open source project run by the Apache Software Foundation, which can be downloaded from http://hadoop.apache.org/. In addition to the MapReduce platform, it includes a distributed file system that stores the data being processed.

End Box

The way that MapReduce was originally described and named implies that its purpose is to map vast amounts of data to extract the useful parts, and then to reduce them into pure distilled information. But experience shows that the actual usage of MapReduce is more varied, and in some cases the volume of the information produced increases instead of being reduced.

Chen et al. describe MapReduce workloads from 6 months on a 600-machine cluster and 1.5 months on a 3000-machine cluster at Facebook, from 3 weeks on a cluster at Yahoo!, and from several other installations [125, 124]. Some of their data is shown in Figures 9.34 and 9.35. Figure 9.34 shows the distributions of the map size, the shuffle size, and the reduce size based on 24 random 1-hour samples from two clusters in two years. It apparently shows that in 2009 the reduce sizes tended to be larger than the map sizes, so the data grew as a result of the processing. Even more surprisingly, for 75% of the jobs the shuffle size was 0, implying that no data was passed from the map stage to the reduce stage. To better understand these results, Figure 9.35 shows the relationship between the map and reduce sizes of each job. The concentration of values along the diagonal indicates that for many jobs the input and output data sizes are similar. Looking at the data indicates that the jobs with zero shuffle size are mainly those that are above the diagonal. A possible interpretation is then that these are reduce-only jobs that do not

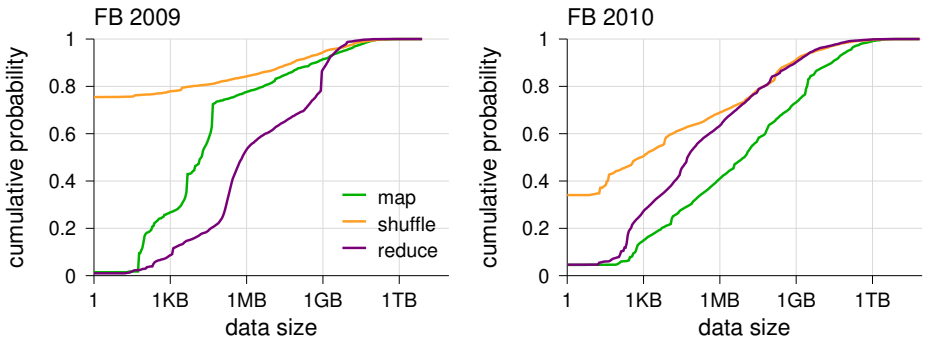


Figure 9.34: *Distributions of map, shuffle, and reduce data sizes from Facebook clusters.*

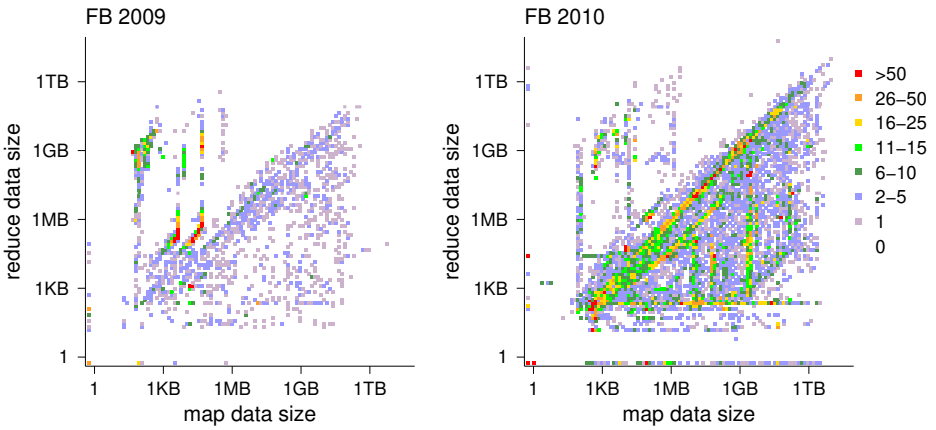


Figure 9.35: *Scatterplots showing the relationship between map size and reduce size in the Facebook clusters data.*

really have a map stage, and that the input is actually a specification of what the reduce should do. Likewise, the zero-reduce jobs in 2010 could be map-only jobs.

Regardless of the data interpretation problems, these graphs show that MapReduce workloads are varied. They may have different relationships between the map and the reduce. The datasets are not all big data — rather, some of them are as small as kilobytes. Some are also interactive [124]. In addition, the data exhibits the expected characteristics that are common to many workloads [124]:

- A daily cycle of submitting work.
- A high degree of burstiness, with peak-to-median ratios of one order of magnitude and even higher.
- Skewed distributions of data sizes.
- A Zipf-like distribution of input file popularity.

Given the growing use of MapReduce as the de facto standard for big data programming, the HiBench benchmark was developed to evaluate the performance of Hadoop [351]. It includes both microbenchmarks and full MapReduce applications. The microbenchmarks are programs that do sorting or word counting on the input. For sorting, both the map and the reduce are identity functions, and the sorting is actually done by the shuffle stage. For word counting, the mappers emit pairs where the key is a word and the value is 1, and the reducers simply sum these values. The applications are related to web search (indexing and page rank) and machine learning (Bayesian classification and k-means clustering). Inputs are based on a dump of Wikipedia with 2.4 million web pages. Finally, a benchmark of the Hadoop file system is also included.

Other benchmarks emphasize the volume and variability aspects of the data [280], thus enabling a system-level evaluation of performance that focuses on text processing workflows. The workflows used are the classification of documents into given categories, the identification of key documents within each category, and the identification of subtopics in categories. These workflows share the common task of retroactively having to find new structure in existing data, which is typical of big data applications. They are applied to a corpus of nearly 400 million web pages, totaling more than 10 TB uncompressed.

An alternative to using benchmarks is to create a statistical model. Such a model was suggested by Ganapathi et al. and includes the following components [275]:

- Successive job arrivals are modeled based on empirical interarrival time data.
- Jobs are assigned to a class based on the popularity of different job names.
- The input size is selected from a job-class–specific distribution and scaled for the desired installation size.
- The shuffle size is created based on sampling from the shuffle/input ratio distribution, and the output size based on an output/shuffle ratio distribution, both of which are specific per job class.

Distributions are modeled by five data points from the empirical workload data: the 1st, 25th, 50th, 75th, and 99th percentiles. Linear interpolation is used between them. The problem with this model is that the sampling is done randomly and independently from the different distributions, so all correlations and locality are lost. However, note that the input/shuffle/output data sizes are not modeled independently, but rather by their ratios. This retains a measure of correlation between them.

Another alternative is to use samples from real data. Chen at al. have claimed that due to the complex nature of the MapReduce workloads and their many attributes, which typically do not conform to well-known distributions, using randomly mixed samples of traced data is better than using benchmarks or models [125].

In addition to the full MapReduce applications discussed so far, it is also interesting to consider the more intricate interaction between components of the MapReduce workflow and the underlying system. This interaction has considerable impact on selecting configuration parameters for optimal performance [719, 333]. In addition, note that MapReduce is defined in terms of key-value pairs, and therefore the handling and

caching of such data is of great importance. Atikoglu et al. report on the workload handled by five large key-value stores at Facebook [42]. Their main findings are as follows.

- In four of the five cases the workload is dominated by get requests, as befits a distributed cache. In the largest and most general purpose of these four cases, there is also a relatively large (30%) component of delete requests. The fifth one, which is used to store transient data such as client window sizes, is dominated by update requests.
- Both key size and value size distributions are modal, with up to 90% of the instances being the same size. The value-size distribution is dominated by small sizes, and the number of large values is small enough that the mass-count disparity is typically low.
- Key popularity exhibits very high mass-count disparity, with most keys requested very few times, but some keys accessed again and again millions of times.
- Key requests also exhibit strong temporal locality, with most reuse occurring within an hour.
- Most stores show daily and weekly cycles of activity.

9.6 Parallel Jobs

Parallel jobs, which are composed of multiple threads (up to many thousands) that cooperate to solve a computational problem, are the mainstay of high-performance computing. So models of parallel job workloads are needed for the evaluation of parallel job schedulers, which are used to allocate the resources of large-scale parallel platforms such as supercomputers and clusters. Such models are nontrivial because parallel jobs have several distinct attributes, most notably their parallelism and runtime, which have to be modeled. Several models and benchmarks for this domain have been devised [741, 230, 370, 187, 137, 454]. They complement the benchmarks of multithreaded applications on modern multicore systems that were discussed in Section 9.2.2.

9.6.1 Arrivals

The arrival process of parallel jobs shares all of the main features of other arrival processes.

In most parallel job models arrivals are assumed to be Poisson. However, real arrivals are actually self-similar [653, 670, 234], as demonstrated in Figure 9.36. In fact, this dataset was the main example used to demonstrate various ways to quantify self-similarity and long-range dependence in Section 7.4. In particular, interarrival times are not exponentially distributed, but rather tend to have a longer tail, fitting a lognormal or Weibull distribution [405]. The arrival of individual jobs is also affected by feedback from the system's performance and the session dynamics, as discussed in Section 8.4.

In addition, arrivals display the obvious daily and weekly cycles (Figure 9.2). Such cycles are especially important in this domain because parallel jobs can be several hours

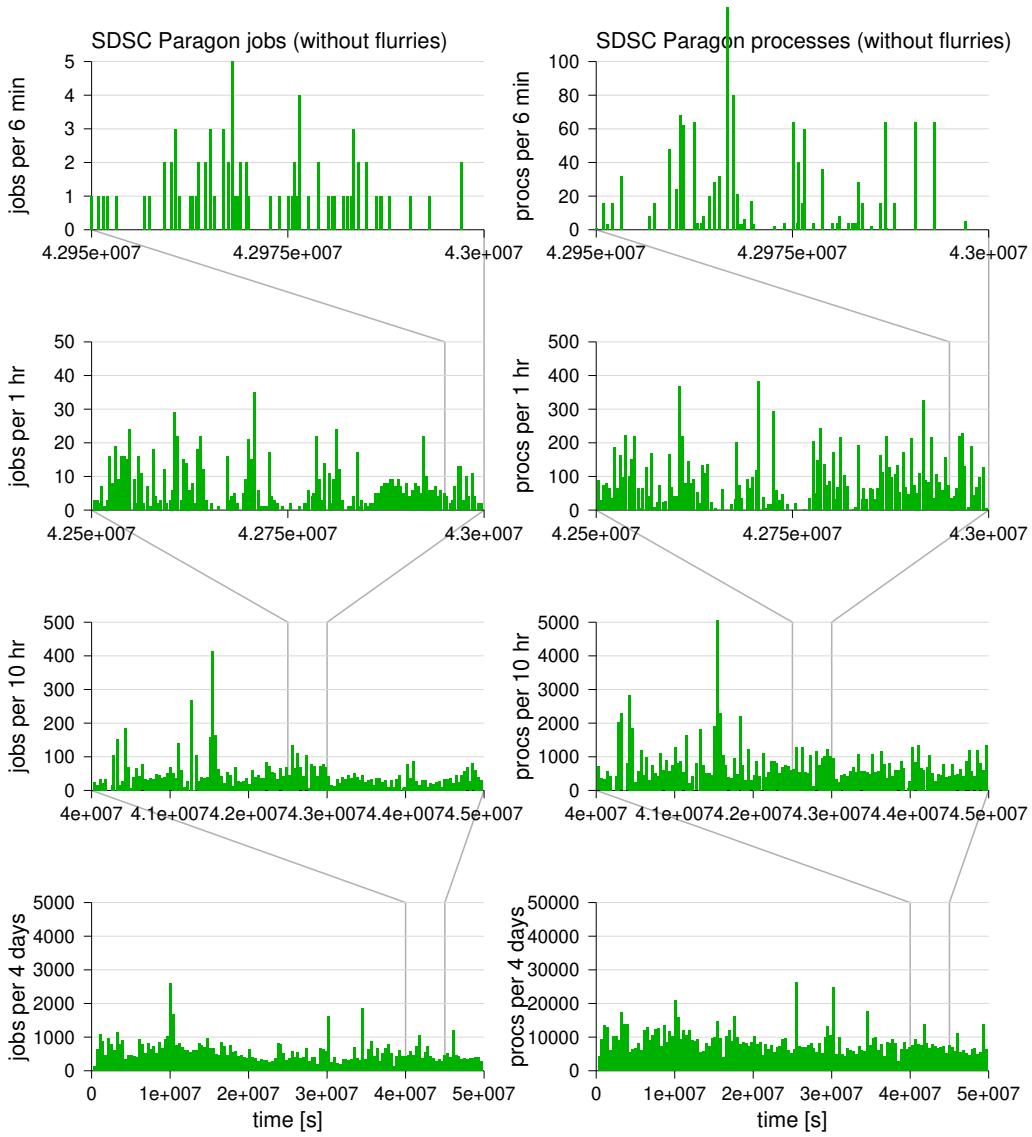


Figure 9.36: *Job and process arrivals at the SDSC Paragon parallel supercomputer shown at different levels of aggregation, showing self-similarity.*

long, so their duration can interact with the daily cycle [616, 248]. Moreover, the workload on parallel supercomputers typically includes a significant non-interactive component, which can be postponed and executed at night. Doing so increases the effective capacity of the system. But the fact that there is extra capacity at night that can be used for the postponed jobs depends on the daily cycle [248].

Another phenomenon that appears to be relatively common in large-scale parallel systems is workload flurries (Figure 9.37). These are large surges of activity, in which

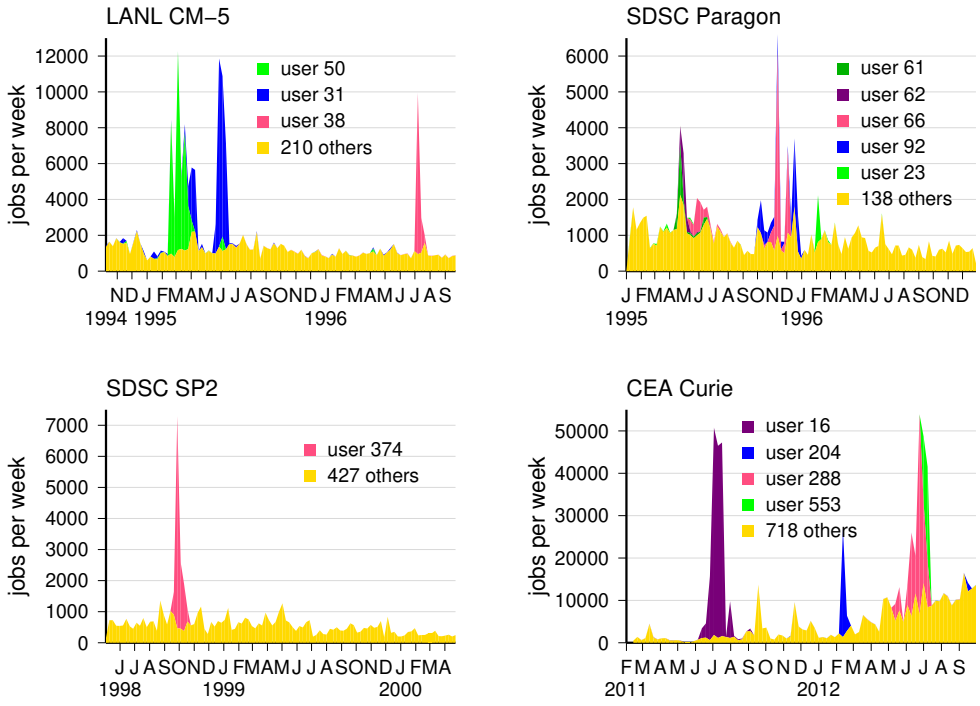


Figure 9.37: *Flurries of activity by hyper-active users on large-scale parallel systems.*

a single user creates a volume of jobs or processes that is many times larger than the normal workload, but only for a limited amount of time. These large flurries are easy to recognize, and given that they are anomalous and distort the workload characteristics, it is recommended that they be removed before using the workload for performance evaluations or modeling. The only reason to leave them in is when studying the effect of the flurries themselves on the system or on the performance experienced by other jobs.

9.6.2 Rigid Jobs

A common approach to modeling parallel workloads is to consider rigid jobs — jobs that use a fixed number of processors. Each job is then a rectangle in processors×time space: it uses a certain number of processors for a certain time. This enables the use of logs that contain just this data and the creation of models based on them [230, 370, 187, 137, 454]. Many of the examples in this book come from this domain.

Job Sizes

The distribution of job sizes is approximately log-uniform, as observed by Downey [187] (Figure 9.38). Lublin suggests that a two-stage log uniform model is better, and also

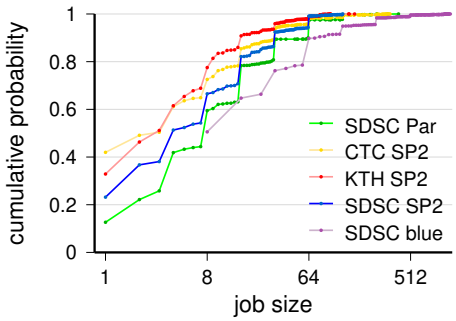


Figure 9.38: *The distribution of parallel job sizes is approximately log-uniform, but with many serial jobs and emphasis on powers of two.*

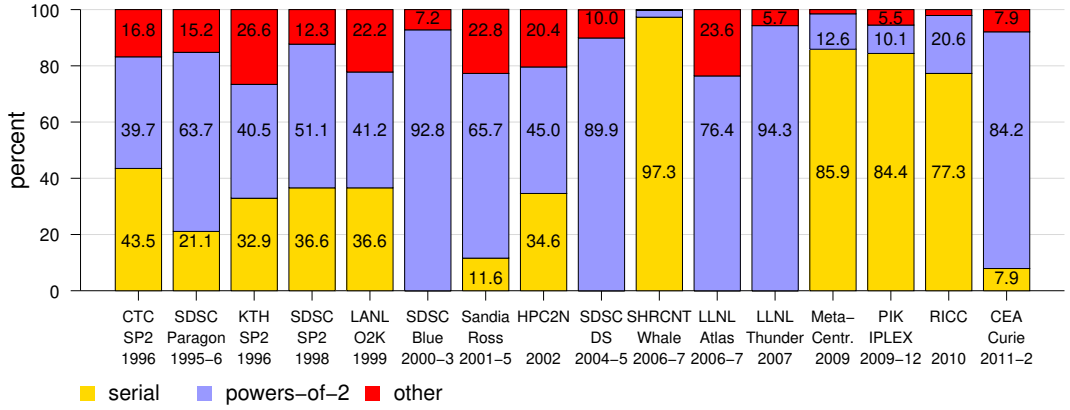
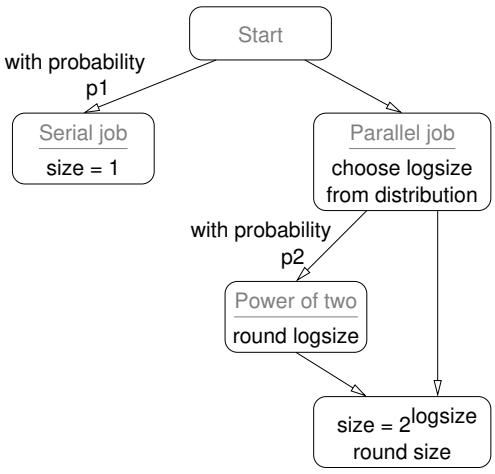


Figure 9.39: *The fraction of serial and power-of-two jobs in different systems. Data showing all jobs in each log, without cleaning.*

relates the two segments to the machine size, so as to create a parametric model that can be adjusted to different machine sizes [454].

In addition to the general trend, the distribution of job sizes also has two special characteristics (Figure 9.39). The first is that a relatively large fraction of the jobs are serial (or at least use a single node, on machines such as the SDSC Blue Horizon in which the minimal allocation is an SMP node with 8 processors). The other is that very many jobs use power-of-two nodes [735]. Lublin therefore suggests the following algorithm for selecting a job size [454] (Figure 9.40). First, decide whether a job is parallel or serial. If parallel, select the log of its size from the model distribution (log-uniform or two-stage log-uniform). Then determine whether it should be a power of two, and if so round the log size. Finally raise 2 to the selected power, and round to obtain an integral size. Note that this model does not represent all workloads very well. In particular, workloads dominated by "bag-of-tasks" jobs composed of multiple individual processes



Parameter	*Value*
p_1	0.24
p_2	0.75

Figure 9.40: *Algorithm to select a parallel job size, from Lublin [454].*

appear to have many more serial jobs. This seems to be the case for recent clusters and grids.

The emphasis on powers of two has been questioned by Downey and by Jann et al. [187, 370]. They claim that this is not a real feature of the workload, but rather an effect of queue size limits in legacy systems. Such systems usually define a large set of queues and associate each with different resource limits. In particular, the limits on the degree of parallelism are typically set to be powers of two (see, e.g., [244, 718]). This claim is supported by a user survey conducted by Cirne, which found that users do not in fact favor powers of two [136]. However, power-of-two jobs continue to be prevalent in all observed workloads.

Job Runtimes

Somewhat surprisingly, parallel job runtimes are not heavy-tailed, in contrast to process runtimes on Unix machines, which are. This may be attributed to two factors. First, the use of large-scale parallel supercomputers is tightly controlled by system administrators, who impose limits on the maximum job length. Second, parallel jobs are typically much longer than Unix processes, so the entire distribution is shifted. Thus the distribution may not be heavy-tailed, but all the values are actually rather high.

Various models have been suggested for modeling parallel job runtimes. Feitelson uses a hyper-exponential distribution with two or three stages [230]. Downey suggests the use of a log-uniform distribution [187]. Jann et al. prefer the more general hyper-Erlang distribution [370]. Lublin in turn suggests the use of a hyper-gamma distribution (i.e., a mixture of two gamma distributions) [454]. The choice is related to how one wants to handle the correlation of runtime and size, as described later.

The runtime distributions of parallel machines also provide an opportunity to consider workload diversity [239]. Figure 9.41 shows the distributions of runtimes from several logs. It indicates that they are typically indeed similar to each other. The most

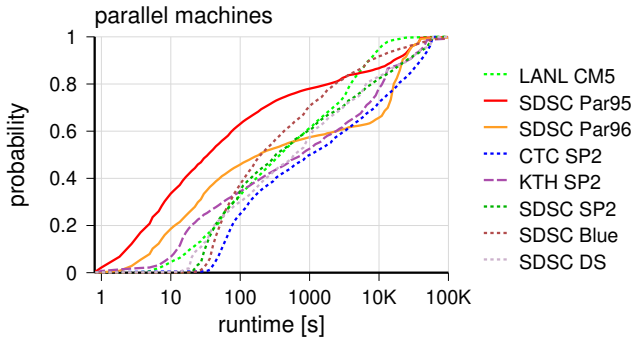


Figure 9.41: *Runtime distributions on different parallel machines.*

prominent deviations are found in the SDSC Paragon log, whose two years exhibit different behaviors.

A related issue is the question of user runtime estimates. Many parallel job schedulers require users to provide an estimate of how long each job will run, in order to plan ahead. The problem is that real user estimates are notoriously inaccurate [242], and it seems that users are generally incapable of providing more accurate estimates [431]. The effect of such inaccurate estimates was investigated using a model where the real time is multiplied by a random factor from a limited range; the larger the range, the less accurate the resulting estimates. However, this model still retained information regarding the relative length of each job, and therefore enabled schedulers to achieve better results than they would in reality [697]. Thus, Tsafrir et al. developed an intricate model to mimic the real relationship between job runtimes and user estimates [694]. A central feature of this model is the discreteness of estimates: users tend to use round values in their estimates, such as 10 minutes or 1 hour, and doing so causes a loss of information. All this is described at some length in Section 9.1.6.

Correlation of Runtime and Size

Parallel job sizes and runtimes generally have a weak correlation coefficient, but a strong distributional correlation, meaning that the distribution of runtimes for small jobs emphasizes shorter runtimes than does the distribution of runtimes for large jobs. This is used as a running example in Section 6.4. But in fact there are some exceptions.

The data from 12 systems is shown in Figure 9.42. In the first eight there is a positive distributional correlation, meaning that big jobs tend to run longer. In four systems this is not the case. Specifically, data from the SDSC DataStar is unique in showing that the distributions of runtimes for small and large jobs are essentially identical. Data from the LANL Origin 2000, the HPC2N cluster, and the PIK IPLEX system indicate that there is an inverse correlation, with small jobs generally running longer. However, in the first two cases, this is a result of unusual modal distributions.

Focusing on the consistent results from the first eight systems, this type of data moti-

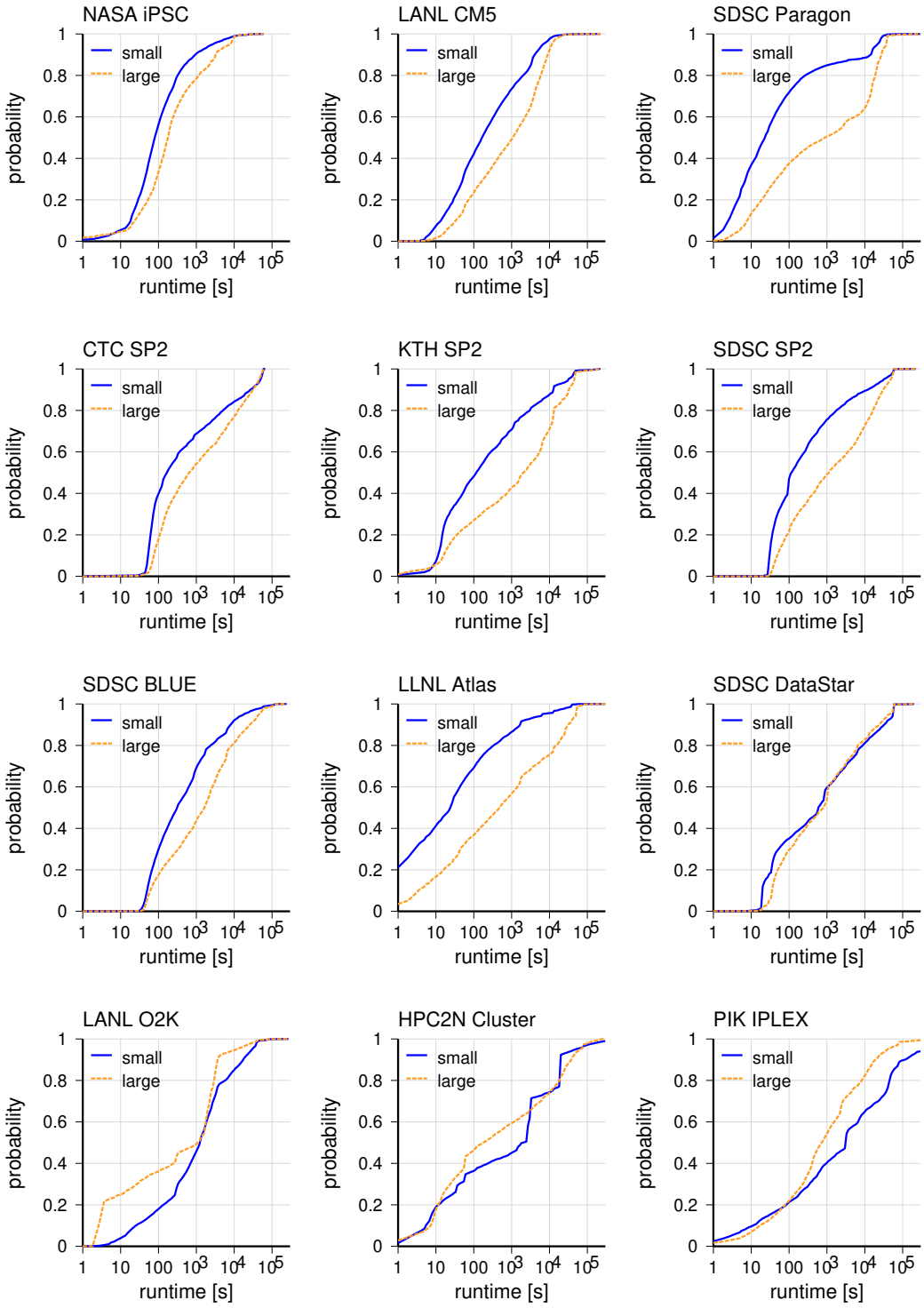


Figure 9.42: *Data from multiple systems mostly shows a positive distributional correlation of runtime and size in parallel jobs.*

vates the modeling of this correlation by distributions of runtimes that are conditioned on job size. Jann et al. partition jobs into bands of sizes based on powers of two or multiples of 5 and 10. They then create a separate model of the runtimes for each band, using a hyper-Erlang distribution [370]. Feitelson and Lublin prefer to use parameterized hyper-exponential and hyper-gamma distributions, respectively [230, 454]. This approach uses the same distributions for all job sizes. However, the probability of using one branch or the other depends on the job size. For example, Feitelson uses a simple two-stage hyper-exponential distribution, in which the probability for using the exponential with the higher mean is linearly dependent on the size:

$$p(n) = 0.95 - 0.2(n/N)$$

Thus, for small jobs (the job size n is small relative to the machine size N) the probability of using the exponential with the smaller mean is 0.95, and for large jobs this drops to 0.75.

9.6.3 Speedup

Some system evaluations cannot be done with rigid jobs, because part of the system's function is to decide on the number of processors to use. This is the case for moldable or malleable jobs. (As defined in [246], moldable jobs allow the system to dictate the number of processors to use when they start up; malleable jobs also allow the number to change at runtime.) This requires a model of the job's speedup — how long it will run on different numbers of processors.

One such model was proposed by Sevcik [608], based on an assumed structure of parallel applications. The speedup function is modeled using two inputs and three parameters. The inputs are

W = the total work done by the job
n = the number of processors allocated

The parameters are

ϕ = inflation in the runtime due to load imbalance among threads
α = the amount of sequential and per-processor work
β = communication and congestion delays that increase with n

The formula for the runtime of the job is then

$$T(n) = \phi \frac{W}{n} + \alpha + \beta\, n$$

Chiang and Vernon have noted that, as written, load imbalance and communication overheads occur even when $n = 1$. They therefore suggest that the formula be changed by writing $n - 1$ in place of n [130]. Measurements by Nguyen et al. on a KSR-2 machine indicate that load imbalance and communication overhead are indeed the two dominant sources of runtime inefficiency and reduced speedup [515].

Suggested values for the parameters are given in Table 9.3, where the values for W are mean values, and the distribution is selected so as to achieve the desired variance. In

	W	ϕ	α	β
workload1	1000	1.02	0.05	0
workload2	1000	1.3	25	25

Table 9.3: *Suggested parameter values for Sevcik's speedup model.*

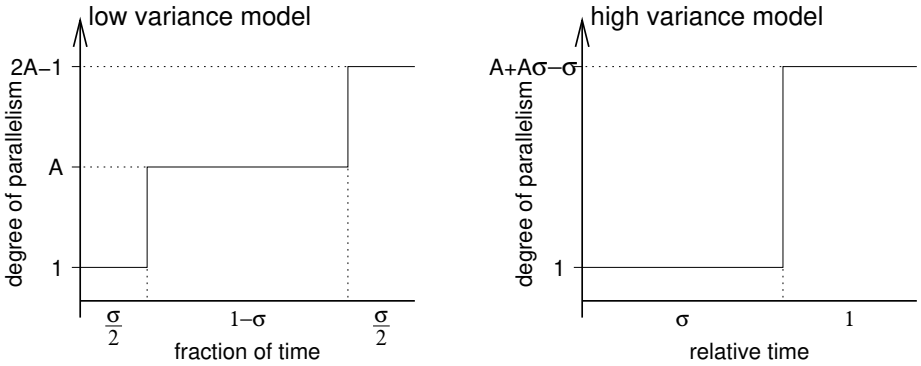


Figure 9.43: *Parallelism profiles assumed in the Downey speedup model.*

the first workload, all jobs have nearly ideal speedup. In the second, small jobs experience poor speedup whereas large jobs experience reasonable speedup.

Another model was proposed by Downey [187] as part of a comprehensive model based on observations of the CTC SP2 and SDSC Paragon workloads. This model explicitly disregards the dominance of power-of-two sizes in the logs, under the assumption that it is a result of the interface to commonly used queueing systems such as NQS. It therefore proposes to "smooth" the distribution. The suggested model is log-uniform: the probability that a job uses less than n processors is proportional to $\log n$. This is interpreted as the average parallelism in the job and is used in the definition of the speedup function.

Rather than model runtime independently from the degree of parallelism, the cumulative runtime (that is the sum over all processors) is modeled. This is taken to represent the total work done by the job, independent of how many processors are used. The actual runtime is then obtained by applying the speedup function, given the number of processors chosen by the scheduler. The proposed model is again log-uniform: the probability that a job uses less than t node-seconds is proportional to $\log t$.

The speedup function is modeled using three parameters: the average parallelism of the job A, the variance in parallelism V, and the number of processors n. Given these parameters, a hypothetical parallelism profile is constructed and the speedup calculated. The parallelism profile (or "shape") indicates the distribution of degrees of parallelism that are exhibited throughout the execution [607].

Two types of profiles are suggested, as illustrated in Figure 9.43. They are characterized by a parameter σ. In both profiles the average parallelism is A, and the variance is $V = \sigma(A-1)^2$. In the low-variance profile, the degree of parallelism is A for all but a fraction σ of the time (naturally $\sigma \leq 1$ in this case). The rest of the time is divided

equally between being serial and highly parallel. The speedup of programs with this profile, as a function of the number of allocated processors n, is then given by

$$S(n) = \begin{cases} \frac{An}{A + \frac{\sigma}{2}(n-1)} & 1 \le n \le A \\ \frac{An}{\sigma(A - \frac{1}{2}) + n(1 - \frac{\sigma}{2})} & A \le n \le 2A - 1 \\ A & n \ge 2A - 1 \end{cases}$$

In the high-variance profile $\sigma \ge 1$. The program is either serial or highly parallel. The speedup of programs with this profile is

$$S(n) = \begin{cases} \frac{An(\sigma+1)}{\sigma(n+A-1)+A} & 1 \le n \le A + A\sigma - \sigma \\ A & n \ge A + A\sigma - \sigma \end{cases}$$

9.6.4 Parallel Program Behavior

Computer jobs are often classified as being compute-bound or I/O-bound, depending on whether they spend most of their time computing or waiting for I/O operations. In parallel jobs there is a third class of communication-bound jobs, in which a large fraction of the time is spent communicating partial results among the different processes. A classification of jobs into these three classes can be done by noting the number of communication operations executed in a time slice and whether the job was blocked on I/O [161].

Communication and Synchronization

As noted earlier, the unique characteristic of parallel jobs is that they are composed of multiple communicating processes. Such communication serves two functions: to move data, and to delay processes that are too far ahead so that others can catch up. In message-passing systems these two functions are performed simultaneously when one process waits for a message from another. In shared memory systems they are separated: a synchronization mechanism is used to delay the receiving process until the data is ready in the shared memory.

The standard for communication (at least on distributed memory machines) is the Message Passing Interface (MPI) [505, 506, 640]. This extensive standard defines multiple types of point-to-point and collective communication primitives, thus enabling the characterization of communication patterns at a relatively high level of abstraction. Results of studies of MPI activity reveal a wide distribution of message sizes, rates, and blocks of computation between messages [708], as do studies based on earlier proprietary interfaces [160]. However, each individual application typically has a rather modal distribution of sizes (Figure 9.44). In addition, collective communications typically have

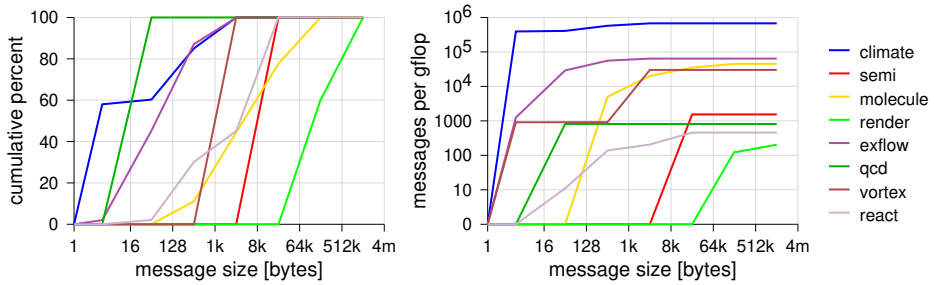


Figure 9.44: *Distributions of message sizes and rates for several parallel applications. Note the logarithmic Y scale of rates, which are normalized per gflop of computation on the entire machine. Data from [160].*

small payloads, indicating that they are used more for synchronization than to move data. The different granularities of computation between communications imply that they have different synchronization requirements: rapid communications require processes that really run in parallel and respond promptly, whereas communications that are separated by large blocks of computation are more tolerant to delays and allow more flexibility in scheduling [269].

An especially important characteristic of message passing is the traffic pattern and how it maps onto the communication network, because different patterns may lead to rather different performance results [546]. In particular, different routing algorithms may interact with the traffic pattern in either good or bad ways [17]. Another issue is network hotspots, in which a large fraction of the traffic is directed at the same destination, thereby causing congestion. Congestion is especially problematic in shared memory architectures, in which shared variables that are used for synchronization or coordination may become hotspots [547, 433, 80, 163]. An interesting effect in this respect is that synchronous communications (that is, a protocol where a processor must wait for communications to complete before going on) lead to feedback from the network, which, in the case of congestion, throttles additional load [605, 17].

Despite the obvious importance of communication patterns to the understanding of parallel applications, there has been little work on modeling them. Instead, the tendency is to use various applications as benchmarks and resort to actual measurements. This reflects the complexity of the problem and the difficulty of devising models that will be good enough for reliable performance predictions. The only model that has seen some use is the simple distinction between computation phases and communication phases, as in the BSP model [706].

Parallel I/O

Parallel I/O is in effect a combination of message passing and regular I/O: one can view it as a set of parallel processes exchanging messages with I/O nodes and through them with disks. It has been modeled in a sense by support for special I/O modes in early parallel

Mode	*Description*
broadcast/reduce	all processes collectively access the same data
scatter/gather	all processes collectively access a sequence of data blocks, in rank order
shared offset	processes operate independently but share a common file pointer
independent	allows programmer complete freedom

Table 9.4: *File modes used in various parallel I/O systems (see Figure 9.45).*

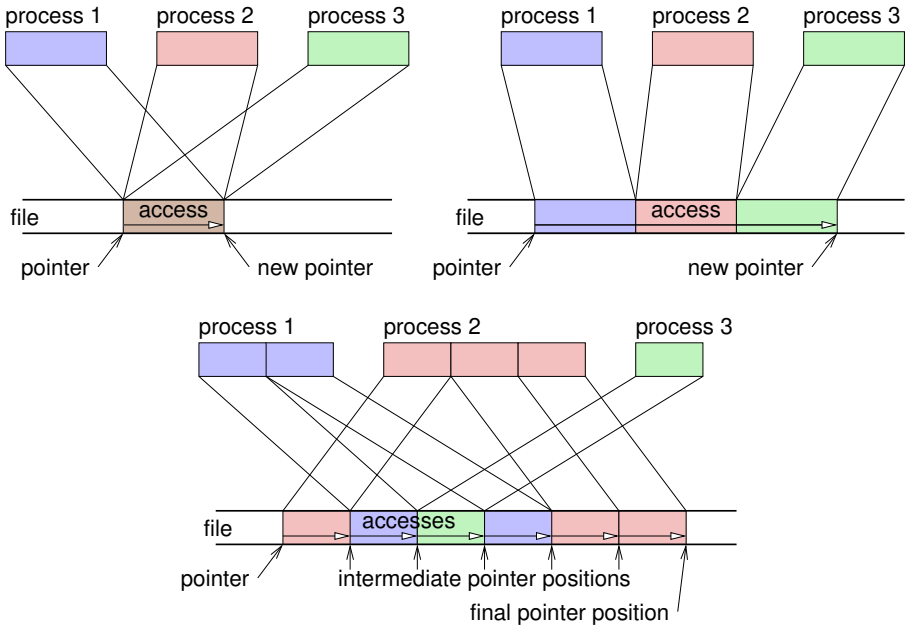


Figure 9.45: *Illustration of the parallel broadcast/reduce, scatter/gather, and shared pointer I/O modes from Table 9.4.*

file systems, as described in Table 9.4 and Figure 9.45 [549, 73]. Many of these modes reflect collective operations, in which I/O from all the nodes is combined in some way. For example, in the broadcast/reduce mode, a *read* operation broadcasts the same data to all the participating processes, whereas a *write* operation performs some reduction on the data provided by the processes and writes the result (e.g., the maximal value). As in message passing, I/O operations tend to come in distinct phases, interspersed by computing [583].

The distribution of I/O operations is similar to the distribution of message sizes. In a study from the mid-1990s, most I/O operations were in the range of tens to hundreds of bytes long, but most of the data being read or written were part of the few very large operations typically involving (many) megabytes [517]. This pattern leads to significant mass-count disparity. The distribution of access sizes is very modal [634]. In files that

were opened for only reading or only writing, most accesses were sequential. However, some of these sequential accesses were not consecutive: they started at a higher file offset than where the previous request ended, but not immediately after it. In files opened for both reading and writing, most accesses were not sequential.

Perhaps the most important characteristic of parallel I/O is that accesses from different processes can be finely interleaved in the file [416, 517]. This is reflected in the file access modes of Table 9.4 and Figure 9.45, which typically reflect a partitioning of some data among the processors. For example, the scatter/gather mode allows the processes to access a matrix together in one collective operation such that each one of them gets a distinct set of rows. This leads to significant differences between the logical access pattern as seen by individual processes and the physical access patterns at the disks. In addition, there may be metadata accesses by the file system [624].

Benchmark Suites

A major problem with modeling the internals of parallel jobs is that there are so many parameters and options: the number of threads, the granularity of the interactions between them, the communication patterns, the amount of data that is transferred, and so on. An alternative is therefore to use benchmarks, namely representative sets of real applications.

One benchmark suite for parallel computing is SPLASH (Stanford parallel applications for shared memory). As the name implies the applications are designed for shared memory, rather than message passing. In fact two versions were defined [632, 741]. The first contained six applications, mostly scientific simulations written in C. The second contains eight applications and four kernels, with better algorithms and implementations that also took better advantage of available hardware facilities.

Another widely used benchmark suite was the NAS parallel benchmark [44], which included five kernels and three applications in computational fluid dynamics from the NASA aerodynamics simulation program. Several problem sizes were defined to enable scaling to larger installations. Later versions of these benchmarks used MPI (message-passing interface) for communication.

Benchmark suites such as SPLASH and NAS were useful for evaluating parallel architectures, in terms of how well they support the different applications. But they were not useful for evaluating parallel job schedulers, where a sequence of jobs that arrive and need to be handled is needed. Such a scenario is provided in the ESP (effective system performance) benchmark defined at Lawrence Berkeley National Lab [740, 232]. This benchmark is composed of 82 jobs that together run for several hours. These jobs are instances of nine applications, where the number of instances of each application is between 1 and 22. In some cases, different instances of the same application use different numbers of processors. Two of the jobs are supposed to use the full machine; this checks how well the system manages to collect nodes without causing undue idleness. Using real applications led to an interesting result when the benchmark was run on two different machines: due to architectural differences, job runtimes were quite diverse and did not

scale in the same way [740]. This implies that workloads assuming rigid jobs may need to use different runtime distributions for different architectures.

9.6.5 Load Manipulation and System Size

It is often desirable to evaluate the performance of a system under different load conditions, e.g. to check its stability or the maximal load it can handle before saturating. Thus the workload model should not model only a single load condition, but should contain a tunable parameter that allows for the generation of different load conditions.

In the context of parallel workloads, the most common approach to changing the load is to systematically modify the interarrival times (it is also possible to change the runtime, but this has the drawback of creating a builtin correlation between response time and load). For example, if a model generates a load of 70% of system capacity by default, multiplying all interarrival times by a factor of $7/8 = 0.875$ will increase the load to 80%, while retaining all other features of the workload model. This is a workable solution for a stationary workload. However, if daily cycles are included in the model, such modifications influence the relative lengths of the jobs and the daily cycle, which is obviously undesirable; for example, it reduces the extra capacity available for scheduling delayed batch jobs at night [248]. Moreover, reducing interarrival times may violate dependencies between jobs and any feedback from system performance to job submittal [614].

Parallel jobs are characterized by two dimensions: their runtime and the number of processors they use. Thus another alternative to modifying the load is by changing the relative size of the jobs and the full machine. For example, if the model generates a load of 70% on a 100-node system, we can reduce the system size to $7/8 \cdot 100 \approx 88$ to create a load of 80% (naturally this has to be rounded to an integer value, so the actual load produced may deviate a bit from the target value). The problem with this approach is that it may change the workload packing characteristics and fragmentation, especially in the common case where both the system size and common job sizes are powers of two, and these changes may have a significant effect on performance [450].

The alternative suggested in Chapter 8 is to use a user-based model. This can be done in either of two ways. The first is to modify the load by changing the size of the user population. As shown in Figure 9.46, the justification for this approach is dubious. The other is to change the characteristics of the users. Thus to increase the load we may cause the users to have longer sessions in which they submit more jobs. Both these approaches have the advantage that they do not create other problems, except possibly some modification of the degree of locality of sampling.

An underlying consideration in all the previous load manipulation schemes is the desire that — except for changing the load — all other aspects of the workload "remain the same". This is required in order to perform the system evaluations under equivalent conditions. However, it is not necessarily the right thing to do. In principle, it may be that the characteristics of the workload on lightly loaded systems are actually different from those on highly loaded systems. If this is the case, we may well want to employ distinct system designs that take the differences in the workloads into account.

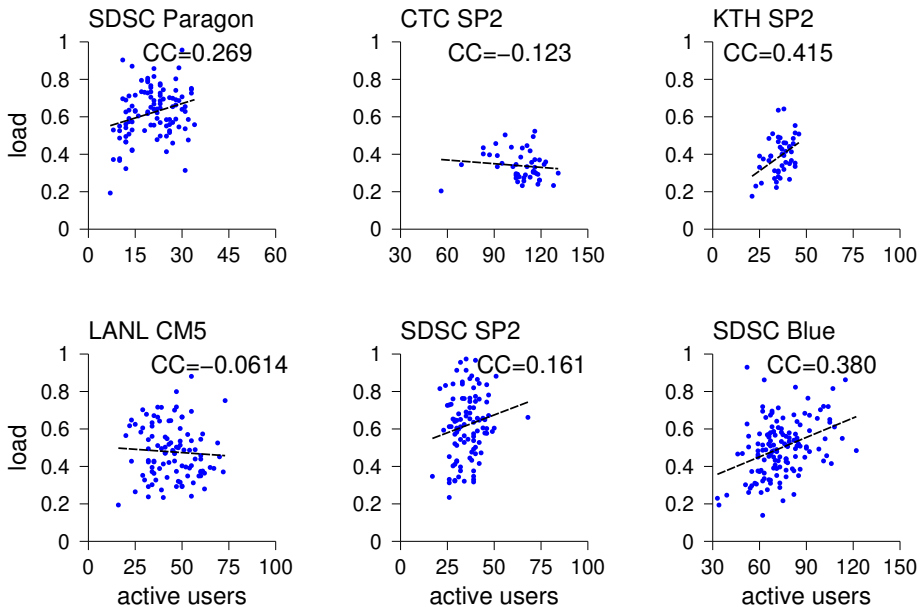


Figure 9.46: *Scatterplots showing number of active users and resulting load, using a resolution of one week.*

Some evidence in this direction is provided by the workload analysis of Talby et al. [671]. Using the co-plot method to compare several parallel system workloads, they show that systems with higher loads tend to have a lower job arrival rate. At the same time, the distribution of job runtimes seems to be uncorrelated with the system load, although there is some weak correlation between job sizes (that is, degree of parallelism) and system load. Thus a model in which increased load is induced by fewer larger jobs may be worth checking. To avoid a strong impact on fragmentation, shifting the distribution of job sizes upward should be done by increasing the probability for larger powers of two, rather than by multiplying job sizes by a constant factor. In effect, this moves the CDF of the job sizes downward rather than to the right (Figure 9.47).

A special case of load manipulation involves adjusting a workload to a different machine size than the original one. For example, we may have a workload log that was captured on a machine with 1024 nodes, but we need to simulate scheduling on a machine with 2048 nodes. Using the given workload as is would lead to low load, and moreover, none of the jobs would be larger than half the machine, which makes packing them together much easier [144, 145, 703]. Some adjustment is therefore required, taking into account the target machine size.

The parallel workload model by Lublin accepts the machine size as a parameter and creates job sizes according to the process described in Figure 9.40 [454]. This is based on a two-phase log-uniform distribution. Assuming we want job sizes in the range from $P_\perp$ (typically 2, because serial jobs are handled separately, but could be larger to

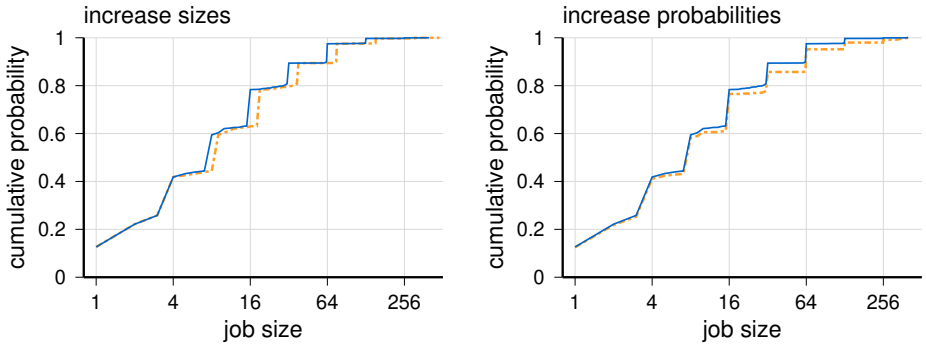


Figure 9.47: *Emphasizing larger jobs by multiplying their sizes by a constant (effectively shifting the CDF to the right, as shown at left) destroys the structure of powers of two; increasing their probabilities (i.e., shifting the CDF downward, right) maintains the structure. Data from SDSC Paragon.*

reflect a minimal allocation) to $P^\top$ (typically the target machine size), we select the jobs size as follows. With probability $p = 0.86$, use the first phase of the distribution, and otherwise the second phase. For the first phase, select a number e uniformly from the range $\log P_\perp - 0.2$ to $\log P^\top - 2.5$. For the second phase, select e from the range $\log P^\top - 2.5$ to $\log P^\top$. The job size will then be 2^e, rounded to the nearest integer or power of two. The parameters in this procedure were set based on typical machine sizes in the hundreds; as sizes have grown considerably since then, the parameters may have to be revised if this algorithm is to be used today.

Ernemann et al. have devised an alternative approach based on manipulating an existing log [213]. They start with a scaling factor f that reflects the required increase in size. Thus if the logged machine had only 300 nodes and the simulation target is 1000, the factor is $f = 3\frac{1}{3}$ (however, in their experimental results they found that it is beneficial to use a slightly higher factor than the precise increase in scale). For each job in the log a random decision is made between two courses of action: either increase the job size by f, or create f copies of the job. Naturally, fractional values need to be handled: if the size is increased the resulting size is rounded to the nearest integer, and if we need $f = x.y$ copies then we create x copies for sure, and an additional one with probability y. The fraction of jobs that are enlarged or replicated can be chosen at will, based on preference for having larger jobs or more small jobs. A possible default is to apply each approach to half of the jobs.

9.6.6 Grid Workloads

The previous sections are largely based on workloads from large-scale parallel supercomputers and clusters. Another platform for parallel computation is grids — loosely federated collections of clusters belonging to different organizations, that are shared to enable better use of resources and the solution of larger problems than possible on

any single cluster. This type of platform may require specialized workload models [358, 357].

One early finding regarding grid workloads concerns the typical degrees of parallelism. It turns out that the vast majority of grid jobs appear to be serial [440, 360, 357] (see Figure 9.39). One reason for this is that grids tend to be used for "bag-of-tasks" type jobs, in which numerous instances of the same application are executed with different parameter values [359, 490]. This usage pattern is even supported by special facilities that ease the submittal of such bags of similar tasks. As a result more than 90% of the jobs may belong to these bags. At the same time, most grids still lack the facilities needed to run truly parallel applications, such as the ability to co-allocate resources at different locations at the same time.

Another interesting deviation of grid systems concerns the arrival process. Specifically, at least in one case it has been observed that arrivals do not exhibit marked daily or weekly cycles, in contrast with practically all other types of workloads [524]. This was attributed to the fact that users come from multiple time zones and may schedule jobs for later execution. However, many other grids do in fact exhibit strong daily and weekly cycles of activity.

10

Summary and Outlook

Developing a model of a nontrivial system is itself nontrivial. There is no simple recipe that can be applied that promises good results. Instead, model building is usually an iterative and interactive process, involving three recurring steps: model formulation, model estimation, and model validation [121, sect. 4.8]. Most books, including this one, devote most of their attention to model estimation. This is the activity of matching a specific piece of a model to a given feature of the data. But one must not forget the big picture.

From Workload Data to Workload Model

In previous chapters we have described and compared many workload models in various domains. Here we want to summarize recurring principles and draw them together.

To recap, there are three main approaches to using workload data:

1. Find the simplest abstract mathematical model that captures a desired feature.
2. Use raw data as when driving simulations directly from traces, or using empirical distributions.
3. Create a generative model that could plausibly give rise to the observed data.

Perhaps the most entrenched and commonly used approach in workload modeling is to use a mathematical abstraction in the form of a statistical model. For example, the method of moments can be used to fit a marginal distribution, and an autocorrelation function is used to characterize the dependence structure and fit a long-range dependent fARIMA model. When a new workload feature is recognized as being important, mathematical modeling is often the first approach used to evaluate its effect. And doing so often leads to great advances in understanding the effect of the new feature.

But such abstractions can also miss out on important issues. Distributions with the correct moments can still have the wrong shape and taint detailed analysis. Moreover, descriptive mathematical models may actually lead to conclusions that do not really reflect the workload. For example, consider a study of a communication network that finds a negative correlation between packet sizes and the subsequent interval to the next

packet. Such a phenomenon may be due to the fact that full packets are usually part of a long flow, so the intervals between them are minimal, whereas smaller packets tend to be singletons (e.g., used for acknowledgments). Thus a negative correlation indeed exists, but it would be wrong to model it as a general feature applicable to all packet sizes.

Naturally, the other approaches to using workload data are also not without problems. Using raw data, for example, may avoid the mistakes that could be made in modeling, but it usually does not lead to as much understanding of the structure of the workload. In particular, raw data by itself provides no means for extrapolation or adjustment to different conditions.

Approaches to Modeling

Focusing on workload modeling, and leaving the option of using raw data to the side, three recurring themes are of importance.

Descriptive vs. generative models. The first and most basic distinction is between descriptive and generative models. Descriptive models are superficial and try to mimic an observed phenomenon, often using a mathematical abstraction. Generative models, in contradistinction, attempt to understand the mechanism that led to the creation of the observed workload. The idea is that by understanding and emulating the mechanism we can better adapt the workload to different conditions.

Parsimonious vs. conservative modeling. Paraphrasing a well-known quote attributed to Einstein, models should be made as simple as possible but not simpler. But how do we know when we cross the line? Emphasizing the quest for simplicity, a parsimonious model includes only what is positively known to be important and nothing else. The opposite approach, taken by conservative models, excludes only what is positively known to be unimportant. As model building is often done in a state of uncertainty, the choice is largely up to the disposition of the analyst. The tradeoff is clear: a conservative model is safer because it will have a higher probability of leading to a correct result, even if you do not fully understand how and why. But this comes at the cost of more work and possibly reduced explanation power.

Open vs. closed models and feedback. A common assumption in many performance evaluations is that the workload is external to and independent of the system. This essentially means we use an open system model. But in many situations it is more reasonable to assume that some feedback occurs and that system performance has some effect on the generation of additional work. A rather extreme model that can then be used is a closed system model, where the user population is assumed to be fixed and every finished job immediately (or after a certain "think time") leads to the submittal of a new job. But mixed models with both open and closed characteristics are probably more realistic.

Realistic modeling of feedback may incur considerable difficulties, because it may involve the modeling of user behavior, and direct data about user behavior is seldom if ever available. However, ignoring the stabilizing effect of feedback might

lead to completely different system dynamics. This leads to the risk of a model that does not reflect reality and consequently to results that cannot be relied on.

The connection between these three themes is the following. Generative models do not settle for a superficial description of the salient features of the workload. Instead, they attempt to model the workload in the context of the system's entire environment. This leads to conservative modeling, which includes more features than the minimum that is necessary to reproduce known phenomena. And a prominent feature that is possible to model in a generative conservative framework, but is typically missing in more conventional and parsimonious settings, is feedback and its effect on stability.

Model Validation

Modeling is always subject to the questions of validation and verification. Validation asserts that the model reflects reality. Verification asserts that the implementation of the model indeed does what it is supposed to. Therefore validation is part of the actual modeling, whereas verification is part of using the model in an evaluation. In this book we are subsequently interested only in validation.

Model validation can be done at three levels.

Statistical validation. Statisticians have developed various techniques to verify that models are valid. A basic approach is to use cross-validation. To do so partition the available data into several subsets, and create a separate model for each part. Then compare the models. If they are dissimilar, they cannot be trusted, because the model obviously depends on an arbitrary decision of what data to use.

When using cross-validation it is important to partition the data in a way that does not disrupt the modeling procedure. For example, given a workload log representing the activity of many users, it may be better to partition at the user level rather than at the job level. Thus all the jobs of certain users will either be included or excluded. In some cases simple deterministic options (such as partitioning into odd and even records) may also be OK.

If sufficient data is not available, bootstrapping can be used. This creates multiple datasets via resampling, which can be used again to create multiple models.

Bootstrapping can also be used to assert that workload features are present in the first place, and deserve to be considered in a model. The idea is to resample the workload data many times subject to some specific modification. For example, the modification can be a random reordering of items in order to test for the importance of dependencies. Then the resampled workloads are compared with the original workload to see whether the original one stands out.

Phenomenological validation The fact that a model is consistent as verified by statistical validation does not necessarily mean that it captures all important workload features. Thus one needs to validate the model relative to the data. The simplest

way to do this is to use goodness-of-fit tests to verify that model distributions indeed match the data. Similarly one needs to verify that the correlation structure (including long-range dependence) is modeled correctly.

A common problem for phenomenological validation is one of workload diversity. We can match a given sample of the workload, but this is but one sample, and other samples may differ. Thus it may be useful to first compare several independent samples from different sources, and to characterize their diversity (e.g., by using the spatial locality of sampling measure). The goal of a good model is then to be able to create different workloads with such diversity. In particular, a worthy objective is to create a parameterized workload model, such that different parameter values lead to the creation of workloads at different locations in the workloads space.

Functional validation The highest level of validation is functional validation, in which we verify that the model workload affects the system under study in the same way as the real workload: it places the same load on servers, causes the same CPU utilization, the same queue lengths, the same packet arrival process, and so on. This should at least hold when the model is parameterized to behave like the original data. However, even if the model behaves correctly at the origin, there is still no way to know that it correctly performs adjustments to other conditions. A good example is adjusting the load, and specifically driving the system toward saturation. How should the workload change under such conditions?

Indeed, a central problem is workload modeling is extrapolating beyond the given data. By definition such extrapolations cannot be verified to be correct. In such cases guidance can be obtained from experience with other workload types. But evaluations that use such extended models should be done subject to an acknowledgment that the model may be suspect.

Finally, one should always remember that a model is only an approximation of the truth. Moreover, except under fortuitous circumstances, it is true for the time of its creation and not necessarily later. Therefore models may need to be revised when new data becomes available.

Data-Driven Generative Models

Significant work has been done on computer systems' workload characterization and modeling over the years, but still the field is in its infancy. A lot is not known, and things change all the time. There is a need to balance theory and assumptions with real data, as well as a need to understand the mechanisms that drive system usage.

Workload modeling necessarily starts with workload data. It is imperative to collect and share workload data. Data is needed not only for individual performance evaluations, but also for meta-studies based on accumulated knowledge. We need to be able to see trends over long periods of time to be able to predict how things may change in the future.

Once you have workload data in hand, the first thing to do is to look at it. Identify outliers and special circumstances that might require handling, and clean the data as appropriate. Then look at marginal distributions, using CDFs and LLCDs to characterize the tails. Check whether mass-count disparity effects are evident. Check for correlations and self-similarity.

Then build your model based on what you have learned from the data. As noted earlier, there are various options for how to go about this: you can opt for a descriptive model or a generative one, a parsimonious model or a conservative one, and an open system or a closed system with feedback. My opinion is that a conservative generative model with feedback should be the goal in most cases, especially if you are in the dark regarding what is really important and how the workload interacts with the system design. But this suggestion reflects more than just my opinion. Generative models with feedback have been proposed and used in many domains, sometimes under different names. For example, in networking, essentially the same idea is called "modeling at the source".

With a workload model in hand you can now evaluate the performance of the system. But at the same time, the evaluations can also be used to learn more about the workload. And new ideas for workload modeling are sure to come up.

A

Data Sources

The data used in this book comes from the following sources.

NASA Ames iPSC/860

Data about 42,264 jobs submitted in the fourth quarter of 1993, collected by Bill Nitzberg. More than half were invocations of pwd by system personnel to verify that the system was responsive. The system is a 128-node hypercube manufactured by Intel.

Available from the Parallel Workloads Archive:
URL http://www.cs.huji.ac.il/labs/parallel/workload/l_nasa_ipsc/.

LANL CM-5

Data about 201,387 jobs submitted from October 1994 to September 1996, collected by Curt Canada, and including three large flurries. The system is a 1024-node Connection Machine 5 from Thinking Machines Corp., the biggest of its kind at the time, which reached second rank in the Top500 list of 1993.

Available from the Parallel Workloads Archive:
URL http://www.cs.huji.ac.il/labs/parallel/workload/l_lanl_cm5/.

SDSC Paragon

Data about 115,591 jobs submitted from January 1995 to December 1996, collected by Reagan Moore and Allen Downey, and including several flurries. The system is a 416-node Intel Paragon machine. Originally this data was provided as two logs for separate years. Due to anonymization, user IDs may be inconsistent in the two years. Therefore we only use the 1995 data when user data is important.

Available from the Parallel Workloads Archive:
URL http://www.cs.huji.ac.il/labs/parallel/workload/l_sdsc_par/.

CTC SP2

Data about 79,302 jobs submitted from June 1996 to May 1997, collected by Dan Dwyer and Steve Hotovy, with one small flurry. The system is a 512-node IBM SP2 machine, the biggest of its kind at the time, and ranked 6 in the 1995 Top500 list.

Available from the Parallel Workloads Archive:
URL http://www.cs.huji.ac.il/labs/parallel/workload/l_ctc_sp2/.

KTH SP2

Data about 28,490 jobs submitted from September 1996 to August 1997, collected by Lars Malinowsky. The system is a 100-node IBM SP2.

Available from the Parallel Workloads Archive:
URL http://www.cs.huji.ac.il/labs/parallel/workload/l_kth_sp2/.

SDSC SP2

Data about 73,496 jobs submitted from April 1998 to April 2000, collected by Victor Hazlewood, and including one large job flurry and one large process flurry. The system is a 128-node IBM SP2.

Available from the Parallel Workloads Archive:
URL http://www.cs.huji.ac.il/labs/parallel/workload/l_sdsc_sp2/.

LANL Origin 2000 Cluster

Data about 122,233 jobs submitted to a cluster of 16 Origin 2000 machines with 128 processors each (the open partition of the ASCI Blue Mountain system), from December 1999 to April 2000, collected by Fabrizio Petrini.

Available from the Parallel Workloads Archive:
URL http://www.cs.huji.ac.il/labs/parallel/workload/l_lanl_o2k/.

SDSC Blue Horizon

Data about 250,440 jobs submitted from April 2000 to January 2003, collected by Travis Earheart and Nancy Wilkins-Diehr, and including several flurries. The system is an 144-node IBM SP, with eight processors per node, which achieved rank 8 in the Top500 list in 2000.

Available from the Parallel Workloads Archive:
URL http://www.cs.huji.ac.il/labs/parallel/workload/l_sdsc_blue/.

HPC2N Cluster

Data about 527,371 jobs submitted from July 2002 to January 2006, collected by Ake Sandgren and Michael Jack. About 57% of the data comes from a single user, in the shape of multiple flurries throughout the log. In addition there are a couple of smaller

flurries. The machine is a 120-node Linux cluster with two processors per node.

Available from the Parallel Workloads Archive:
URL http://www.cs.huji.ac.il/labs/parallel/workload/l_hpc2n/.

SDSC DataStar

Data about 96,089 jobs submitted from March 2004 to March 2005, collected by Victor Hazlewood. The system is an 184-node IBM eServer pSeries, with two types of nodes: 176 p655 nodes which are 8-way SMPs, and 8 p690 nodes which are 32-way SMPs.

Available from the Parallel Workloads Archive:
URL http://www.cs.huji.ac.il/labs/parallel/workload/l_sdsc_ds/.

PIK IPLEX

Data about 742,964 jobs submitted over more than three years, from April 2009 to July 2012, to a 320-node IBM iDataPlex Cluster, and collected by Ciaron Linstead. Each node has two processors with 4 cores each, for a total of 2560 cores in the whole system.

Available from the Parallel Workloads Archive:
URL http://www.cs.huji.ac.il/labs/parallel/workload/l_pik_iplex/.

CEA Curie

Data about 773,138 jobs submitted from February 2011 to October 2012, collected by Joseph Emeras, and containing four large flurries. The system is a combination of three partitions with different properties, designed by Bull: a partition with 360 fat nodes with four 8-core processors, another partition with 144 nodes, each with two Intel processors and two Nvidia GPUs, and a third with 5040 nodes that each have two 8-core processors. The latter two partitions were added after the beginning of the logging period, so the full configuration was in effect only during the last 11 months.

Available from the Parallel Workloads Archive:
URL http://www.cs.huji.ac.il/labs/parallel/workload/l_cea_curie/.

Unix File Sizes Survey

A Usenet survey by Gordon Irlam in October 1993, with data about 12,128,881 files from 1050 file systems [361].

Available from Gordon Irlam's website:
URL http://www.gordoni.com/ufs93.html. Also cached at this book's website.

HU-CS File Sizes

Sizes of user files (including both staff and students) at the School of Computer Science and Engineering at Hebrew University. This was scanned in June 2005 and contains data about 2,860,921 files.

Available from this book's website.

/cs/par File Sizes

File sizes from the file system of the Parallel Systems Laboratory at the School of Computer Science and Engineering at Hebrew University. This was scanned in 2003 and was found to include 1,025,039 files.

Available from this book's website.

Unix Process Runtimes

Dataset used by Mor Harchol-Balter and Allen Downey in simulations [320], that includes data about 184,612 processes that were submitted to a server at CMU over an 8-hour period, from 9AM to 5 PM, in November 1994.

Available from Allen Downey's website:
URL http://www.allendowney.com/research/sigmetrics96/traces.tar.gz.

HU-CS Process Runtimes

Data on 448,597 Unix processes submitted to a departmental server at the School of Computer Science and Engineering at Hebrew University, from 20 October to 24 November 2005.

Available from this book's website.

Pita Process Runtimes

Data on 816,689 Unix processes submitted to a departmental server, from a two-week period in December 1998, that includes robot activity by root and user 13 (user IDs were sanitized).

Available from this book's website.

Inferno Process Runtimes

Data on 661,452 Unix processes submitted to a departmental server, from a 30-hour period in March 2003. Of these, 539,253 were invocations of the Unix ps command by a single user, the result of a bug in implementing an operating systems course exercise.

Available from this book's website.

Inferno Session Lengths

Data about Unix login sessions to a departmental server from several multiday periods in 2004.

Available from this book's website.

BYU Memory Access Traces

Traces of memory accesses of SPEC 2000 benchmarks as executed on a Pentium III system.

Available from the Brigham Young University Trace Distribution Center:
URL http://tds.cs.byu.edu/tds/.

SDSC HTTP

Log from a web server at SDSC, including 28,338 requests made on 22 August 1995, collected by Joshua Polterock, Hans-Werner Braun, and K Claffy.

Available from the Internet Traffic Archive:
URL http://ita.ee.lbl.gov/html/contrib/SDSC-HTTP.html.

NASA HTTP

Log from a web server at NASA Kennedy Space Center, including 1,891,714 requests made during July 1995, and used by Martin Arlitt and Catey Williamson [37, 38]. Of these, we typically use the 1,887,646 GET requests.

Available from the Internet Traffic Archive:
URL http://ita.ee.lbl.gov/html/contrib/NASA-HTTP.html.

Saskatchewan HTTP

Log from a web server at the University of Saskatchewan, including 2,408,625 requests made from June through December 1995, also used by Arlitt and Williamson [37, 38]. Of these, we typically use the 2,398,080 GET requests.

Available from the Internet Traffic Archive:
URL http://ita.ee.lbl.gov/html/contrib/Sask-HTTP.html.

WC'98 HTTP

Complete activity log of the 1998 Soccer World Cup website collected and analyzed by Martin Arlitt [35], which includes 1,352,804,108 requests made from May to July.

Available from the Internet Traffic Archive:
URL http://ita.ee.lbl.gov/html/contrib/WorldCup.html.

AOL search

10 logs containing data about 20,792,287 queries submitted by 657,425 users over three months (March through May 2006). When using this data, we remove user 71845, who is responsible for 267,933 queries performed round the clock during five days in early April; this seems to be the combined stream of many different users coming from some other search engine, and would therefore distort any study of individual user search behavior. For statistics regarding queries we also remove the query "-", which is the

most popular at 464,724 appearances. It is assumed this actually means "no data".

This dataset was available for a short time from AOL Research at http://research.aol.com [539], but the uproar regarding privacy compromization caused the data to be withdrawn. However, copies may still be available on the Internet. Note that due to the lack of a strong effort to prevent privacy leaks, this dataset is more reliable then others in terms of reflecting real user activity.

Wikipedia page views

Extensive data at hourly resolution regarding page views of all Wikipedia-related projects since December 2007.

Available from Wikimedia:
URL http://dumps.wikimedia.org/other/pagecounts-raw/.

Facebook MapReduce

The SWIM project (Statistical Workload Injector for MapReduce) released several samples of MapReduce workload data from clusters at Facebook. Each file is the concatenation of 24 random one-hour samples from the original log file [125, 124].

Available from GitHub:
URL https://github.com/SWIMProjectUCB/SWIM/wiki.

BC packets

Several traces of Bellcore Ethernet traffic from 1989, used by Wilson et al. in the first characterizations of self-similarity in LAN traffic [436, 732].

Available from the Internet Traffic Archive:
URL ftp://ita.ee.lbl.gov/html/contrib/BC.html.

LBL packets

Several traces of Lawrence Berkeley Lab Internet traffic from 1994, used by Vern Paxson and Sally Floyd to show self-similarity in WAN traffic [540].

Available from the Internet Traffic Archive:
URL http://ita.ee.lbl.gov/html/contrib/LBL-TCP-3.html.

WIDE B

Daily sample of Internet packet data from 2001 to 2006, from access point B of the WIDE backbone. This is a transpacific link between Japan and the United States. Data from other access points is also available, including point F, which replaced point B in July 2006.

Available from the MAWI working group:
URL http://mawi.wide.ad.jp/mawi/.

Bibliography

[1] I. B. Aban, M. M. Meerschaert, and A. K. Panorska, "*Parameter estimation for the truncated Pareto distribution*". *J. Am. Stat. Assoc.* **101(473)**, pp. 270–277, Mar 2006, DOI: 10.1198/016214505000000411.

[2] T. F. Abdelzaher and N. Bhatti, "*Web content adaptation to improve server overload behavior*". *Comput. Networks* **31(11-16)**, pp. 1563–1577, May 1999, DOI: 10.1016/S1389-1286(99)00031-6.

[3] P. Abry, R. Baraniuk, P. Flandrin, R. Riedi, and D. Veitch, "*Multiscale nature of network traffic*". *IEEE Signal Processing Mag.* **19(3)**, pp. 28–46, May 2002, DOI: 10.1109/79.998080.

[4] P. Abry, P. Borgnat, F. Ricciato, A. Scherrer, and D. Veitch, "*Revisiting an old friend: On the observability of the relation between long range dependence and heavy tail*". *Telecomm. Syst.* **43(3-4)**, pp. 147–165, Apr 2010, DOI: 10.1007/s11235-009-9205-6.

[5] P. Abry, P. Flandrin, M. S. Taqqu, and D. Veitch, "*Self-similarity and long-range dependence through the wavelet lens*". In *Theory and Applications of Long-Range Dependence*, P. Doukhan, G. Oppenheim, and M. S. Taqqu (eds.), pp. 527–556, Birkhäuser, 2003.

[6] P. Abry and D. Veitch, "*Wavelet analysis of long-range-dependent traffic*". *IEEE Trans. Information Theory* **44(1)**, pp. 2–15, Jan 1998, DOI: 10.1109/18.650984.

[7] P. Abry, D. Veitch, and P. Flandrin, "*Long-range dependence: Revisiting aggregation with wavelets*". *J. Time Series Analysis* **19(3)**, pp. 253–266, May 1998, DOI: 10.1111/1467-9892.00090.

[8] L. A. Adamic, "*Zipf, power-laws, and Pareto – a ranking tutorial*". URL http://www.hpl.hp.com/research/idl/papers/ranking/, 2000.

[9] L. A. Adamic and B. A. Huberman, "*The web's hidden order*". *Comm. ACM* **44(9)**, pp. 55–59, Sep 2001, DOI: 10.1145/383694.383707.

[10] L. A. Adamic and B. A. Huberman, "*Zipf's law and the Internet*". *Glottometrics* **3**, pp. 143–150, 2002.

[11] E. J. Adams, "*Workload models for DBMS performance evaluation*". In 13th *ACM Comput. Sci. Conf.*, pp. 185–195, Mar 1985, DOI: 10.1145/320599.320676.

[12] E. Adar, "*User 4XXXXX9: Anonymizing query logs*". In *WWW'07 Workshop on Query Log Analysis*, May 2007.

[13] E. Adar and L. A. Adamic, "*Tracking information epidemics in blogspace*". In *Intl. Conf. Web Intelligence*, pp. 207–214, Sep 2005, DOI: 10.1109/WI.2005.151.

[14] E. Adar and B. A. Huberman, "*Free riding on Gnutella*". *First Monday* **5(10)**, Oct 2000.

[15] E. Adar, D. S. Weld, B. N. Bershad, and S. D. Gribble, "*Why we search: Visualizing and predicting user behavior*". In 16th *Intl. World Wide Web Conf.*, pp. 161–170, May 2007, DOI: 10.1145/1242572.1242595.

[16] R. J. Adler, R. E. Feldman, and M. S. Taqqu (eds.), *A Practical Guide to Heavy Tails: Statistical Techniques and Applications*. Birkhäuser, 1998.

[17] V. S. Adve and M. K. Vernon, "*Performance analysis of mesh interconnection networks with deterministic routing*". *IEEE Trans. Parallel & Distributed Syst.* **5(3)**, pp. 225–246, Mar 1994, DOI: 10.1109/71.277793.

[18] N. Agrawal, W. J. Bolosky, J. R. Douceur, and J. R. Lorch, "*A five-year study of file-system metadata*". *ACM Trans. Storage* **3(3)**, art. 9, Oct 2007, DOI: 10.1145/1288783.1288788.

[19] A. K. Agrawala, J. M. Mohr, and R. M. Bryant, "*An approach to the workload characterization problem*". *Computer* **9(6)**, pp. 18–32, Jun 1976, DOI: 10.1109/C-M.1976.218610.

[20] H. Akaike, "*A new look at the statistical model identification*". *IEEE Trans. Automatic Control* **AC-19(6)**, pp. 716–723, Dec 1974, DOI: 10.1109/TAC.1974.1100705.

[21] A. O. Allen, *Probability, Statistics, and Queueing Theory with Computer Science Applications*. Academic Press, 1978.

[22] G. Almási, C. Caşcaval, and D. A. Padua, "*Calculating stack distances efficiently*". In *Workshop Memory Syst. Perf.*, pp. 37–43, Jun 2002, DOI: 10.1145/773146.773043.

[23] V. Almeida, A. Bestavros, M. Crovella, and A. de Oliveira, "*Characterizing reference locality in the WWW*". In *Parallel & Distributed Inf. Syst.*, pp. 92–103, Dec 1996, DOI: 10.1109/PDIS.1996.568672.

[24] C. Anderson, *The Long Tail: Why the Future of Business is Selling Less of More*. Hyperion, 2006.

[25] D. Anderson, "*You don't know Jack about disks*". *Queue* **1(4)**, pp. 20–30, Jun 2003, DOI: 10.1145/864056.864058.

[26] D. P. Anderson, J. Cobb, E. Korpela, M. Lebofsky, and D. Werthimer, "*SETI@home: An experiment in public-resource computing*". *Comm. ACM* **45(11)**, pp. 56–61, Nov 2002, DOI: 10.1145/581571.581573.

[27] E. Anderson, "*Capture, conversion, and analysis of an intense NFS workload*". In 7th *USENIX Conf. File & Storage Technologies*, pp. 139–152, Feb 2009.

[28] J. M. Anderson and M. S. Lam, "*Global optimizations for parallelism and locality on scalable parallel machines*". In *Proc. SIGPLAN Conf. Prog. Lang. Design & Implementation*, pp. 112–125, Jun 1993, DOI: 10.1145/155090.155101.

[29] T. W. Anderson and D. A. Darling, "*A test of goodness of fit*". *J. Am. Statist. Assoc.* **49(268)**, pp. 765–769, Dec 1954, DOI: 10.1080/01621459.1954.10501232.

[30] N. Andrade, W. Cirne, F. Brasileiro, and P. Roisenberg, "*OurGrid: An approach to easily assemble grids with equitable resource sharing*". In *Job Scheduling Strategies for Parallel Processing*, pp. 61–86, Springer-Verlag, 2003, DOI: 10.1007/10968987_4. Lect. Notes Comput. Sci. vol. 2862.

[31] F. J. Anscombe, "*Graphs in statistical analysis*". *The American Statistician* **27(1)**, pp. 17–21, Feb 1973, DOI: 10.1080/00031305.1973.10478966.

[32] P. Apparao, R. Iyer, X. Zhang, D. Newell, and T. Adelmeyer, "*Characterization & analysis of a server consolidation benchmark*". In 4th *Intl. Conf. Virtual Execution Environments*, pp. 21–29, Mar 2008, DOI: 10.1145/1346256.1346260.

[33] I. Ari, B. Hong, E. L. Miller, S. A. Brandt, and D. D. E. Long, "*Managing flash crowds on the Internet*". In 11th *Modeling, Anal. & Simulation of Comput. & Telecomm. Syst.*, pp. 246–249, Oct 2003, DOI: 10.1109/MASCOT.2003.1240667.

[34] M. Arlitt, "*Characterizing web user sessions*". *Performance Evaluation Rev.* **28(2)**, pp. 50–56, Sep 2000, DOI: 10.1145/362883.362920.

[35] M. Arlitt and T. Jin, "*A workload characterization study of the 1998 world cup web site*". *IEEE Network* **14(3)**, pp. 30–37, May/Jun 2000, DOI: 10.1109/65.844498.

[36] M. F. Arlitt and C. L. Williamson, "*A synthetic workload model for Internet Mosaic traffic*". In *Summer Computer Simulation Conf.*, pp. 852–857, Jul 1995.

[37] M. F. Arlitt and C. L. Williamson, "*Web server workload characterization: The search for invariants*". In *SIGMETRICS Conf. Measurement & Modeling of Comput. Syst.*, pp. 126–137, May 1996, DOI: 10.1145/233008.233034.

[38] M. F. Arlitt and C. L. Williamson, "*Internet web servers: Workload characterization and performance implications*". *IEEE/ACM Trans. Networking* **5(5)**, pp. 631–645, Oct 1997, DOI: 10.1109/90.649565.

[39] E. Asensio, J. Orduña, and P. Morillo, "*Analyzing the network traffic requirements of multiplayer online games*". In 2nd *Intl. Conf. Advanced Eng. Comput. & Apps. in Sci.*, pp. 229–234, Sep 2008, DOI: 10.1109/ADVCOMP.2008.15.

[40] C. Ash, *The Probability Tutoring Book*. IEEE Press, 1993.

[41] S. Asmussen, O. Nerman, and M. Olsson, "*Fitting phase-type distributions via the EM algorithm*". *Scand. J. Stat.* **23(4)**, pp. 419–441, Dec 1996.

[42] B. Atikoglu, Y. Xu, E. Frachtenberg, S. Jiang, and M. Paleczny, "*Workload analysis of a large-scale key-value store*". In *SIGMETRICS Conf. Measurement & Modeling of Comput. Syst.*, pp. 53–64, Jun 2012, DOI: 10.1145/2254756.2254766.

[43] S. Bachthaler, F. Belli, and A. Federova, "*Desktop workload characterization for CMP/SMT and implications for operating system design*". In *Workshop on Interaction between Operating Syst. & Comput. Arch.*, pp. 2–9, Jun 2007.

[44] D. H. Bailey, E. Barszcz, L. Dagum, and H. D. Simon, "*NAS parallel benchmark results*". *IEEE Trans. Parallel & Distributed Syst.* **1(1)**, pp. 43–51, Feb 1993, DOI: 10.1109/88.219861.

[45] H. G. Baker, "*'Infant mortality' and generational garbage collection*". *SIGPLAN Notices* **28(4)**, pp. 55–57, Apr 1993, DOI: 10.1145/152739.152747.

[46] M. G. Baker, J. H. Hartman, M. D. Kupfer, K. W. Shirriff, and J. K. Ousterhout, "*Measurements of a distributed file system*". In 13th *Symp. Operating Systems Principles*, pp. 198–212, Oct 1991, DOI: 10.1145/121132.121164. Correction in *Operating Systems Rev.* **27(1)**, pp. 7–10, Jan 1993.

[47] N. Bansal and M. Harchol-Balter, "*Analysis of SRPT scheduling: Investigating unfairness*". In *SIGMETRICS Conf. Measurement & Modeling of Comput. Syst.*, pp. 279–290, Jun 2001, DOI: 10.1145/384268.378792.

[48] A.-L. Barabási, "*The origin of bursts and heavy tails in human dynamics*". *Nature* **435(7039)**, pp. 207–211, May 2005, DOI: 10.1038/nature03459.

[49] A.-L. Barabási and R. Albert, "*Emergence of scaling in random networks*". *Science* **286(5439)**, pp. 509–512, 15 Oct 1999, DOI: 10.1126/science.286.5439.509.

[50] A.-L. Barabási, R. Albert, and H. Jeong, "*Mean-field theory for scale-free random networks*". *Physica A* **272(1-2)**, pp. 173–187, Oct 1999, DOI: 10.1016/S0378-4371(99)00291-5.

[51] A. Barak and A. Litman, "*MOS: A multiprocessor distributed operating system*". *Software — Pract. & Exp.* **15(8)**, pp. 725–737, Aug 1985, DOI: 10.1002/spe.4380150802.

[52] A. Barak and A. Shiloh, "*A distributed load-balancing policy for a multicomputer*". *Software — Pract. & Exp.* **15(9)**, pp. 901–913, Sep 1985, DOI: 10.1002/spe.4380150905.

[53] M. Barbaro and T. Zeller, Jr., "*A face is exposed for AOL searcher no. 4417749*". *The New York Times*, 9 Aug 2006. (Technology Section).

[54] S. Barber, "*User community modeling language (UCML™) v1.1 for performance test workloads*". URL http://www.perftestplus.com/resources/ucml_1.1.pdf, Mar 2004.

[55] S. Barber, "*Creating effective load models for performance testing with incomplete empirical data*". In 6th *IEEE Intl. Workshop Web Site Evolution*, pp. 51–59, Sep 2004, DOI: 10.1109/WSE.2004.10002.

[56] J.-M. Bardet, G. Lang, G. Oppenheim, A. Philippe, S. Stoev, and M. S. Taqqu, "*Semi-parametric estimation of the long-range dependence parameter: A survey*". In *Theory and Applications of Long-Range Dependence*, P. Doukhan, G. Oppenheim, and M. S. Taqqu (eds.), pp. 557–578, Birkhäuser, 2003.

[57] P. Barford and M. Crovella, "*Generating representative web workloads for network and server performance evaluation*". In *SIGMETRICS Conf. Measurement & Modeling of Comput. Syst.*, pp. 151–160, Jun 1998, DOI: 10.1145/277851.277897.

[58] P. Barford, R. Nowak, R. Willett, and V. Yegneswaran, "*Toward a model of source addresses of Internet background radiation*". In 7th *Passive & Active Measurement Conf.*, Mar 2006.

[59] M. Baroni, "*Distributions in text*". In *Corpus Linguistics: An International Handbook*, A. Lüdeling and M. Kytö (eds.), chap. 39, pp. 803–821, Mouton de Gruyter, 2009.

[60] L. A. Barroso, K. Gharachorloo, and E. Bugnion, "*Memory system characterization of commercial workloads*". In 25th *Ann. Intl. Symp. Computer Architecture Conf. Proc.*, pp. 3–14, Jun 1998, DOI: 10.1145/279358.279363.

[61] A. P. Batson and A. W. Madison, "*Measurement of major locality phases in symbolic reference streams*". In *SIGMETRICS Conf. Measurement & Modeling of Comput. Syst.*, pp. 75–84, 1976, DOI: 10.1145/800200.806184.

[62] M. A. Beg, "*Estimation of the tail probability of the truncated Pareto distribution*". *J. Inf. & Optimization Sci.* **2(2)**, pp. 192–198, 1981.

[63] S. M. Beitzel, E. C. Jensen, A. Chowdhury, D. Grossman, and O. Frieder, "*Hourly analysis of a very large topically categorized web query log*". In 27th *ACM SIGIR Conf.*, pp. 321–328, Jul 2004, DOI: 10.1145/1008992.1009048.

[64] R. H. Bell, Jr. and L. K. John, "*Improved automatic testcase synthesis for performance model validation*". In 19th *Intl. Conf. Supercomputing*, pp. 111–120, Jun 2005, DOI: 10.1145/1088149.1088164.

[65] L. Bent, M. Rabinovich, G. M. Voelker, and Z. Xiao, "*Characterization of a large web site population with implications for content delivery*". In 13th *Intl. World Wide Web Conf.*, pp. 522–533, May 2004, DOI: 10.1145/988672.988743.

[66] J. L. Bentley, D. D. Sleator, R. E. Tarjan, and V. K. Wei, "*A locally adaptive data compression scheme*". *Comm. ACM* **29(4)**, pp. 320–330, Apr 1986, DOI: 10.1145/5684.5688.

[67] J. Beran, *Statistics for Long-Memory Processes*. Chapman & Hall / CRC, 1994.

[68] J. Beran, "*Music - chaos, fractals, and information*". *Chance* **17(4)**, pp. 7–16, Fall 2004, DOI: 10.1080/09332480.2004.10554920.

[69] J. Beran, R. Sherman, M. S. Taqqu, and W. Willinger, "*Long-range dependence in variable-bit-rate video traffic*". *IEEE Trans. Commun.* **43(2/3/4)**, pp. 1566–1579, Feb/Mar/Apr 1995, DOI: 10.1109/26.380206.

[70] G. D. Bergland, "*A guided tour of the fast Fourier transform*". *IEEE Spectrum* **6(7)**, pp. 41–52, Jul 1969, DOI: 10.1109/MSPEC.1969.5213896.

[71] M. R. Berthold, C. Borgelt, F. Höppner, and F. Klawonn, *Guide to Intelligent Data Analysis*. Springer-Verlag, 2010.

[72] L. Bertolotti and M. C. Calzarossa, "*Models of mail server workloads*". *Performance Evaluation* **46(2-3)**, pp. 65–76, Oct 2001, DOI: 10.1016/S0166-5316(01)00047-5.

[73] M. L. Best, A. Greenberg, C. Stanfill, and L. W. Tucker, "*CMMD I/O: A parallel Unix I/O*". In 7th *Intl. Parallel Processing Symp.*, pp. 489–495, Apr 1993, DOI: 10.1109/IPPS.1993.262828.

[74] A. Bianco, G. Mardente, M. Mellia, M. Munafò, and L. Muscariello, "*Web user session characterization via clustering techniques*". In *IEEE Globecom*, pp. 1102–1107, Nov 2005, DOI: 10.1109/GLOCOM.2005.1577807.

[75] C. Bienia, S. Kumar, J. P. Singh, and K. Li, "*The PARSEC benchmark suite: Characterization and architectural implications*". In 17th *Intl. Conf. Parallel Arch. & Compilation Tech.*, pp. 72–81, Oct 2008, DOI: 10.1145/1454115.1454128.

[76] P. Bodík, A. Fox, M. J. Franklin, M. I. Jordan, and D. A. Patterson, "*Characterizing, modeling, and generating workload spikes for stateful services*". In 1st *Symp. Cloud Comput.*, pp. 241–252, Jun 2010, DOI: 10.1145/1807128.1807166.

[77] S. H. Bokhari, "*On the mapping problem*". *IEEE Trans. Comput.* **C-30(3)**, pp. 207–214, Mar 1981, DOI: 10.1109/TC.1981.1675756.

[78] B. Bollobás, O. Riordan, J. Spencer, and G. Tusnády, "*The degree sequence of a scale-free random graph process*". *Random Structures & Algorithms* **18(3)**, pp. 279–290, May 2001, DOI: 10.1002/rsa.1009.

[79] D. Bonfiglio, M. Mellia, M. Meo, and D. Rossi, "*Detailed analysis of Skype traffic*". *IEEE Trans. Multimedia* **11(1)**, pp. 117–127, Jan 2009, DOI: 10.1109/TMM.2008.2008927.

[80] B. Boothe and A. Ranade, "*Performance on a bandwidth constrained network: How much bandwidth do we need?*" In *Supercomputing '93*, pp. 906–915, Nov 1993, DOI: 10.1109/SUPERC.1993.1263549.

[81] M. S. Borella, "*Source models of network game traffic*". *Comput. Commun.* **23(4)**, pp. 403–410, Feb 2000, DOI: 10.1016/S0140-3664(99)00197-8.

[82] P. Borgnat, G. Dewaele, K. Fukuda, P. Abry, and K. Cho, "*Seven years and one day: Sketching the evolution of Internet traffic*". In *IEEE INFOCOM*, pp. 711–719, Apr 2009, DOI: 10.1109/INFCOM.2009.5061979.

[83] G. E. P. Box, G. M. Jenkins, and G. C. Reinsel, *Time Series Analysis: Forecasting and Control*. Prentice Hall, 3rd ed., 1994.

[84] L. S. Brakmo and L. L. Peterson, "*Experience with network simulation*". In *SIGMETRICS Conf. Measurement & Modeling of Comput. Syst.*, pp. 80–90, May 1996, DOI: 10.1145/233008.233027.

[85] L. Breslau, P. Cao, L. Fan, G. Phillips, and S. Shenker, "*Web caching and Zipf-like distributions: Evidence and implications*". In 18th *IEEE INFOCOM*, vol. 1, pp. 126–134, Mar 1999, DOI: 10.1109/INFCOM.1999.749260.

[86] M. B. Breughe and L. Eeckhout, "*Selecting representative benchmark inputs for exploring microprocessor design spaces*". *ACM Trans. Arch. & Code Optimization* **10(4)**, art. 37, Dec 2013, DOI: 10.1145/2541228.2555294.

[87] J. Brevik, D. Nurmi, and R. Wolski, "*Predicting bounds on queuing delay for batch-scheduled parallel machines*". In 11th *Symp. Principles & Practice of Parallel Programming*, pp. 110–118, Mar 2006, DOI: 10.1145/1122971.1122989.

[88] E. A. Brewer, "*Lessons from giant-scale services*". *IEEE Internet Comput.* **5(4)**, pp. 46–55, Jul/Aug 2001, DOI: 10.1109/4236.939450.

[89] M. Brinkmeier and T. Schank, "*Network statistics*". In *Network Analysis: Methodological Foundations*, U. Brandes and T. Erlebach (eds.), pp. 293–317, Springer-Verlag, 2005, DOI: 10.1007/978-3-540-31955-9_11. Lect. Notes Comput. Sci. vol. 3418.

[90] B. Briscoe, A. Odlyzko, and B. Tilly, "*Metcalfe's law is wrong*". *IEEE Spectrum* **43(7)**, pp. 26–31, Jul 2006, DOI: 10.1109/MSPEC.2006.1653003.

[91] A. Broder, "*A taxonomy of web search*". *SIGIR Forum* **36(2)**, Fall 2002, DOI: 10.1145/792550.792552.

[92] A. Broido, Y. Hyun, R. Gao, and k. claffy, "*Their share: Diversity and disparity in IP traffic*". In 5th *Intl. Workshop Passive & Active Network Measurement*, pp. 113–125, Springer-Verlag, Apr 2004, DOI: 10.1007/978-3-540-24668-8_12. Lect. Notes Comput. Sci. vol. 3015.

[93] W. Buchholz, "*A synthetic job for measuring system performance*". *IBM Syst. J.* **8(4)**, pp. 309–318, 1969, DOI: 10.1147/sj.84.0309.

[94] B. Buck and J. K. Hollingsworth, "*An API for runtime code patching*". *Intl. J. High Performance Comput. Appl.* **14(4)**, pp. 317–329, Winter 2000, DOI: 10.1177/109434200001400404.

[95] R. B. Bunt, J. M. Murphy, and S. Majumdar, "*A measure of program locality and its application*". In *SIGMETRICS Conf. Measurement & Modeling of Comput. Syst.*, pp. 28–40, Aug 1984, DOI: 10.1145/800264.809311.

[96] M. Burgess, H. Haugerud, S. Straumsnes, and T. Reitan, "*Measuring system normality*". *ACM Trans. Comput. Syst.* **20(2)**, pp. 125–160, May 2002, DOI: 10.1145/507052.507054.

[97] W. Bux and U. Herzog, "*The phase concept: Approximation of measured data and perfrmance analysis*". In *Computer Performance*, K. M. Chandy and M. Reiser (eds.), pp. 23–38, North Holland, 1977.

[98] N. N. Buzikashvili and B. J. Jansen, "*Limits of the web log analysis artifacts*". In *Workshop on Logging Traces of Web Activity: The Mechanics of Data Collection*, May 2006.

[99] M. Calzarossa and D. Ferrari, "*A sensitivity study of the clustering approach to workload modeling*". *Performance Evaluation* **6(1)**, pp. 25–33, Mar 1986, DOI: 10.1016/0166-5316(86)90006-4.

[100] M. Calzarossa, G. Haring, G. Kotsis, A. Merlo, and D. Tessera, "*A hierarchical approach to workload characterization for parallel systems*". In *High-Performance Computing and Networking*, pp. 102–109, Springer-Verlag, May 1995, DOI: 10.1007/BFb0046616. Lect. Notes Comput. Sci. vol. 919.

[101] M. Calzarossa, L. Massari, and D. Tessera, "*Workload characterization issues and methodologies*". In *Performance Evaluation: Origins and Directions*, G. Haring, C. Lindemann, and M. Reiser (eds.), pp. 459–482, Springer-Verlag, 2000, DOI: 10.1007/3-540-46506-5_20. Lect. Notes Comput. Sci. vol. 1769.

[102] M. Calzarossa and G. Serazzi, "*A characterization of the variation in time of workload arrival patterns*". *IEEE Trans. Comput.* **C-34(2)**, pp. 156–162, Feb 1985, DOI: 10.1109/TC.1985.1676552.

[103] M. Calzarossa and G. Serazzi, "*Workload characterization: A survey*". *Proc. IEEE* **81(8)**, pp. 1136–1150, Aug 1993, DOI: 10.1109/5.236191.

[104] M. Calzarossa and G. Serazzi, "*Construction and use of multiclass workload models*". *Performance Evaluation* **19(4)**, pp. 341–352, 1994, DOI: 10.1016/0166-5316(94)90046-9.

[105] J. Cao, W. S. Cleveland, D. Lin, and D. X. Sun, "*On the nonstationarity of Internet traffic*". In *SIGMETRICS Conf. Measurement & Modeling of Comput. Syst.*, pp. 102–112, Jun 2001, DOI: 10.1145/378420.378440.

[106] O. Cappé, E. Moulines, J.-C. Pesquet, A. Pertopulu, and X. Yang, "*Long-range dependence and heavy-tail modeling for teletraffic data*". *IEEE Signal Processing Mag.* **19(3)**, pp. 14–27, May 2002, DOI: 10.1109/79.998079.

[107] Carna Botnet, "*Internet census 2012: Port scanning /0 using insecure embedded devices*". URL http://internetcensus2012.bitbucket.org/paper.html, 2013.

[108] M. Carvalho and F. Brasileiro, "*A user-based model of grid computing workloads*". In 13th *Intl. Conf. Grid Computing*, pp. 40–48, Sep 2012, DOI: 10.1109/Grid.2012.13.

[109] G. Casale, N. Mi, L. Cherkasova, and E. Smirni, "*Dealing with burstiness in multi-tier applications: Models and their parameterization*". *IEEE Trans. Softw. Eng.* **38(5)**, pp. 1040–1053, Sep/Oct 2012, DOI: 10.1109/TSE.2011.87.

[110] G. Casale, N. Mi, and E. Smirni, "*Model-driven system capacity planning under workload burstiness*". *IEEE Trans. Comput.* **59(1)**, pp. 66–80, Jan 2010, DOI: 10.1109/TC.2009.135.

[111] J. P. Casazza, M. Greenfield, and K. Shi, "*Redefining server performance characterization for virtualization benchmarking*". *Intel Tech. J.* **10(3)**, pp. 243–251, Aug 2006.

[112] J. P. Casmira, D. P. Hunter, and D. R. Kaeli, "*Tracing and characterization of Windows NT-based system workloads*". *Digital Tech. J.* **10(1)**, pp. 6–21, Dec 1998.

[113] L. D. Catledge and J. E. Pitkow, "*Characterizing browsing strategies in the world-wide web*". *Comput. Netw. & ISDN Syst.* **27(6)**, pp. 1065–1073, Apr 1995, DOI: 10.1016/0169-7552(95)00043-7.

[114] M. Cha, H. Kwak, P. Rodriguez, Y.-Y. Ahn, and S. Moon, "*I tube, you tube, everybody tubes: Analyzing the world's largest user generated content video system*". In 7th *Internet Measurement Conf.*, pp. 1–13, Oct 2007, DOI: 10.1145/1298306.1298309.

[115] A. Chakrabarty and G. Samorodnitsky, "*Understanding heavy tails in a bounded world or, is a truncated heavy tail heavy or not?*" *Stochastic Models* **28(1)**, pp. 109–143, 2012, DOI: 10.1080/15326349.2012.646551.

[116] C. Chambers, W.-c. Feng, S. Sahu, and D. Saha, "*Measurement-based characterization of a collection of on-line games*". In 5th *Internet Measurement Conf.*, pp. 1–14, Oct 2005.

[117] V. Chandola, A. Banerjee, and V. Kumar, "*Anomaly detection: A survey*". *ACM Comput. Surv.* **41(3)**, art. 15, Jul 2009, DOI: 10.1145/1541880.1541882.

[118] S. J. Chapin, W. Cirne, D. G. Feitelson, J. P. Jones, S. T. Leutenegger, U. Schwiegelshohn, W. Smith, and D. Talby, "*Benchmarks and standards for the evaluation of parallel job schedulers*". In *Job Scheduling Strategies for Parallel Processing*, pp. 67–90, Springer-Verlag, 1999, DOI: 10.1007/3-540-47954-6_4. Lect. Notes Comput. Sci. vol. 1659.

[119] A. D. Chapman, "*Quality control and validation of point-sourced environmental resource data*". In *Spatial Accuracy Assessment: Land Information Uncertainty in Natural Resources*, K. Lowell and A. Jaton (eds.), pp. 409–418, CRC Press, 1999.

[120] C. Chatfield, "*Confessions of a pragmatic statistician*". *The Statistician* **51(1)**, pp. 1–20, Mar 2002, DOI: 10.1111/1467-9884.00294.

[121] C. Chatfield, *The Analysis of Time Series: An Introduction*. Chapman & Hall/CRC, 6th ed., 2003.

[122] J. B. Chen, Y. Endo, K. Chan, D. Mazières, A. Dias, M. Seltzer, and M. D. Smith, "*The measured performance of personal computer operating systems*". *ACM Trans. Comput. Syst.* **14(1)**, pp. 3–40, Feb 1996, DOI: 10.1145/225535.225536.

[123] X. Chen and J. Heidemann, "*Flash crowd mitigation via adaptive admission control based on application-level observations*". *ACM Trans. Internet Technology* **5(3)**, pp. 532–569, Aug 2005, DOI: 10.1145/1084772.1084776.

[124] Y. Chen, S. Alspaugh, and R. Katz, "*Interactive analytical processing in big data systems: A cross-industry study of MapReduce workloads*". *Proc. VLDB Endowment* **5(12)**, pp. 1802–1813, Aug 2012.

[125] Y. Chen, A. Ganapathi, R. Griffith, and R. Katz, "*The case for evaluating MapReduce performance using workload suites*". In 19th *Modeling, Anal. & Simulation of Comput. & Telecomm. Syst.*, pp. 390–399, Jul 2011, DOI: 10.1109/MASCOTS.2011.12.

[126] Y. Chen, Y. Huang, L. Eeckhout, G. Fursin, L. Peng, O. Temam, and C. Wu, "*Evaluating iterative optimization across 1000 data sets*". In *Proc. SIGPLAN Conf. Prog. Lang. Design & Implementation*, pp. 448–459, Jun 2010, DOI: 10.1145/1809028.1806647.

[127] P. S. Cheng, "*Trace-driven system modeling*". *IBM Syst. J.* **8(4)**, pp. 280–289, 1969, DOI: 10.1147/sj.84.0280.

[128] L. Cherkasova, K. Ozonat, N. Mi, J. Symons, and E. Smirni, "*Anomaly? application change? or workload change?*" In *Intl. Conf. Dependable Syst. & Networks*, pp. 452–461, Jun 2008, DOI: 10.1109/DSN.2008.4630116.

[129] L. Cherkasova, K. Ozonat, N. Mi, J. Symons, and E. Smirni, "*Automated anomaly detection and performance modeling of enterprise applications*". *ACM Trans. Comput. Syst.* **27(3)**, art. 6, Nov 2009, DOI: 10.1145/1629087.1629089.

[130] S.-H. Chiang and M. K. Vernon, "*Dynamic vs. static quantum-based parallel processor allocation*". In *Job Scheduling Strategies for Parallel Processing*, pp. 200–223, Springer-Verlag, 1996, DOI: 10.1007/BFb0022295. Lect. Notes Comput. Sci. vol. 1162.

[131] S.-H. Chiang and M. K. Vernon, "*Characteristics of a large shared memory production workload*". In *Job Scheduling Strategies for Parallel Processing*, pp. 159–187, Springer-Verlag, 2001, DOI: 10.1007/3-540-45540-X_10. Lect. Notes Comput. Sci. vol. 2221.

[132] J. Cho and S. Roy, "*Impact of search engines on page popularity*". In 13th *Intl. World Wide Web Conf.*, pp. 20–29, May 2004, DOI: 10.1145/988672.988676.

[133] J. C.-Y. Chou, T.-Y. Huang, K.-L. Huang, and T.-Y. Chen, "*SCALLOP: A scalable and load-balanced peer-to-peer lookup protocol*". *IEEE Trans. Parallel & Distributed Syst.* **17(5)**, pp. 419–433, May 2006, DOI: 10.1109/TPDS.2006.66.

[134] D. Christensen, "*Fast algorithms for the calculation of Kendall's τ*". *Computational Statistics* **20(1)**, pp. 51–62, Mar 2005, DOI: 10.1007/BF02736122.

[135] A. Christopoulos and M. J. Lew, "*Beyond eyeballing: Fitting models to experimental data*". *Critical Rev. Biochemistry & Molecular Biology* **35(5)**, pp. 359–391, 2000, DOI: 10.1080/10409230091169212.

[136] W. Cirne and F. Berman, "*A model for moldable supercomputer jobs*". In 15th *Intl. Parallel & Distributed Processing Symp.*, Apr 2001, DOI: 10.1109/IPDPS.2001.925004.

[137] W. Cirne and F. Berman, "*A comprehensive model of the supercomputer workload*". In 4th *Workshop on Workload Characterization*, pp. 140–148, Dec 2001, DOI: 10.1109/WWC.2001.990753.

[138] D. Citron, "*MisSPECulation: Partial and misleading use of SPEC CPU2000 in computer architecture conferences*". In 30th *Ann. Intl. Symp. Computer Architecture Conf. Proc.*, pp. 52–61, Jun 2003, DOI: 10.1145/871656.859625.

[139] D. Citron, D. G. Feitelson, and L. Rudolph, "*Accelerating multi-media processing by implementing memoing in multiplication and division units*". In *Intl. Conf. Architect. Support for Prog. Lang. & Operating Syst.*, pp. 252–261, Oct 1998, DOI: 10.1145/291069.291056.

[140] D. Citron and L. Rudolph, "*Creating a wider bus using caching techniques*". In 1st *Intl. Symp. High Performance Comput. Arch.*, pp. 90–99, Jan 1995, DOI: 10.1109/HPCA.1995.386552.

[141] A. Clauset, C. R. Shalizi, and M. E. J. Newman, "*Power-law distributions in empirical data*". *SIAM Rev.* **51(4)**, pp. 661–703, Nov 2009, DOI: 10.1137/070710111.

[142] A. Clauset, M. Young, and K. S. Gleditsch, "*Scale invariance in the severity of terrorism*". *J. Conflict Resolution* **51**, Feb 2007.

[143] R. G. Clegg, "*A practical guide to measuring the Hurst parameter*". *Intl. J. Simulation: Syst., Sci. & Tech.* **7(2)**, pp. 3–14, Mar 2006. URL http://ijssst.info/Vol-07/No-2/one.pdf.

[144] E. G. Coffman, Jr., M. R. Garey, and D. S. Johnson, "*Approximation algorithms for bin-packing — an updated survey*". In *Algorithm Design for Computer Systems Design*, G. Ausiello, M. Lucertini, and P. Serafini (eds.), pp. 49–106, Springer-Verlag, 1984.

[145] E. G. Coffman, Jr., M. R. Garey, and D. S. Johnson, "*Bin packing with divisible item sizes*". *J. Complex.* **3(4)**, pp. 406–428, Dec 1987, DOI: 10.1016/0885-064X(87)90009-4.

[146] E. G. Coffman, Jr. and R. C. Wood, "*Interarrival statistics for time sharing systems*". *Comm. ACM* **9(7)**, pp. 500–503, Jul 1966, DOI: 10.1145/365719.365961.

[147] A. Cooper, "*A survey of query log privacy-enhancing techniques from a policy perspective*". *ACM Trans. Web* **2(4)**, art. 19, Oct 2008, DOI: 10.1145/1409220.1409222.

[148] C. Cooper and A. Frieze, "*A general model of web graphs*". *Random Structures & Algorithms* **22(3)**, pp. 311–335, May 2003, DOI: 10.1002/rsa.10084.

[149] P. F. Corbett, D. G. Feitelson, J.-P. Prost, G. S. Almasi, S. J. Baylor, A. S. Bolmarcich, Y. Hsu, J. Satran, M. Snir, R. Colao, B. D. Herr, J. Kavaky, T. R. Morgan, and A. Zlotek, "*Parallel file systems for the IBM SP computers*". *IBM Syst. J.* **34(2)**, pp. 222–248, 1995, DOI: 10.1147/sj.342.0222.

[150] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein, *Introduction to Algorithms*. MIT Press and McGraw-Hill, 2nd ed., 2001.

[151] C. Costa, C. Ramos, Ítalo Cunha, and J. M. Almeida, "*GENIUS: A generator of interactive user media sessions*". In 7th *Workshop on Workload Characterization*, pp. 29–36, Oct 2004, DOI: 10.1109/WWC.2004.1437393.

[152] P.-J. Courtois and H. Vantilborgh, "*A decomposable model of program paging behavior*". *Acta Informatica* **6(3)**, pp. 251–275, Sep 1976, DOI: 10.1007/BF00288657.

[153] P. Cremonesi and G. Serazzi, "*End-to-end performance of web services*". In *Performance Evaluation of Complex Systems: Techniques and Tools*, M. C. Calzarossa and S. Tucci (eds.), pp. 158–178, Springer-Verlag, 2002, DOI: 10.1007/3-540-45798-4_8. Lect. Notes Comput. Sci. vol. 2459.

[154] M. E. Crovella, "*Performance evaluation with heavy tailed distributions*". In *Job Scheduling Strategies for Parallel Processing*, pp. 1–10, Springer-Verlag, 2001, DOI: 10.1007/3-540-45540-X_1. Lect. Notes Comput. Sci. vol. 2221.

[155] M. E. Crovella and A. Bestavros, "*Self-similarity in world wide web traffic: Evidence and possible causes*". *IEEE/ACM Trans. Networking* **5(6)**, pp. 835–846, Dec 1997, DOI: 10.1109/90.650143.

[156] M. E. Crovella and L. Lipsky, "*Long-lasting transient conditions in simulations with heavy-tailed workloads*". In *Winter Simulation conf.*, Dec 1997.

[157] M. E. Crovella and L. Lipsky, "*Simulations with heavy-tailed workloads*". In *Self-Similar Network Traffic and Performance Evaluation*, K. Park and W. Willinger (eds.), pp. 89–100, John Wiley & Sons, 2000, DOI: 10.1002/047120644X.ch3.

[158] M. E. Crovella and M. S. Taqqu, "*Estimating the heavy tail index from scaling properties*". *Methodology & Comput. in Applied Probability* **1(1)**, pp. 55–79, Jul 1999, DOI: 10.1023/A:1010012224103.

[159] A. Cuzzocrea, D. Saccà, and J. D. Ullman, "*Big data: A research agenda*". In 17th *Intl. Databases Eng. & Apps. Symp.*, pp. 198–203, Oct 2013, DOI: 10.1145/2513591.2527071.

[160] R. Cypher, A. Ho, S. Konstantinidou, and P. Messina, "*A quantitative study of parallel scientific applications with explicit communication*". *J. Supercomput.* **10(1)**, pp. 5–24, 1996, DOI: 10.1007/BF00128097.

[161] F. A. B. da Silva and I. D. Scherson, "*Improving parallel job scheduling using runtime measurements*". In *Job Scheduling Strategies for Parallel Processing*, pp. 18–38, Springer-Verlag, 2000, DOI: 10.1007/3-540-39997-6_2. Lect. Notes Comput. Sci. vol. 1911.

[162] R. B. D'Agostino and M. A. Stephens (eds.), *Goodness-of-Fit Techniques*. Marcel Dekker, Inc., 1986.

[163] S. P. Dandamudi, "*Reducing hot-spot contention in shared-memory multiprocessor systems*". *IEEE Concurrency* **7(1)**, pp. 48–59, Jan-Mar 1999, DOI: 10.1109/4434.749135.

[164] P. B. Danzig, S. Jamin, R. Cáceres, D. J. Mitzel, and D. Estrin, "*An empirical workload model for driving wide-area TCP/IP network simulations*". *Internetworking: Research and Experience* **3**, pp. 1–26, 1992.

[165] R. B. Davies and D. S. Harte, "*Tests for Hurst effect*". *Biometrika* **74(1)**, pp. 95–101, Mar 1987, DOI: 10.1093/biomet/74.1.95.

[166] D. J. Davis, "*An analysis of some failure data*". *J. Am. Stat. Assoc.* **47(258)**, pp. 113–150, Jun 1952, DOI: 10.1080/01621459.1952.10501160.

[167] J. Dean and S. Ghemawat, "*MapReduce: Simplified data processing on large clusters*". In 6th *Symp. Operating Systems Design & Implementation*, pp. 137–150, Dec 2004.

[168] J. Dean and S. Ghemawat, "*MapReduce: A flexible data processing tool*". *Comm. ACM* **53(1)**, pp. 72–77, Jan 2010, DOI: 10.1145/1629175.1629198.

[169] M. H. DeGroot, *Probability and Statistics*. Addison-Wesley, 1975.

[170] A. P. Dempster, N. M. Laird, and D. B. Rubin, "*Maximum likelihood from incomplete data via the EM algorithm*". *J. Royal Statist. Soc. Series B (Methodological)* **39(1)**, pp. 1–38, 1977, DOI: 10.2307/2984875.

[171] P. J. Denning, "*The working set model for program behavior*". *Comm. ACM* **11(5)**, pp. 323–333, May 1968, DOI: 10.1145/363095.363141.

[172] P. J. Denning, "*Working sets past and present*". *IEEE Trans. Softw. Eng.* **SE-6(1)**, pp. 64–84, Jan 1980, DOI: 10.1109/TSE.1980.230464.

[173] P. J. Denning, "*The locality principle*". *Comm. ACM* **48(7)**, pp. 19–24, Jul 2005, DOI: 10.1145/1070838.1070856.

[174] P. J. Denning and S. C. Schwartz, "*Properties of the working-set model*". *Comm. ACM* **15(3)**, pp. 191–198, Mar 1972, DOI: 10.1145/361268.361281.

[175] M. V. Devarakonda and R. K. Iyer, "*Predictability of process resource usage: A measurement-based study on UNIX*". *IEEE Trans. Softw. Eng.* **15(12)**, pp. 1579–1586, Dec 1989, DOI: 10.1109/32.58769.

[176] G. Dewaele, K. Fukuda, P. Borgnat, P. Abry, and K. Cho, "*Extracting hidden anomalies using sketch and non Gaussian multiresolution statistical detection procedures*". In *Workshop Large Scale Attack Defense*, pp. 145–152, Aug 2007, DOI: 10.1145/1352664.1352675.

[177] S. Di, D. Kondo, and W. Cirne, "*Characterization and comparison of cloud versus grid workloads*". In *Intl. Conf. Cluster Comput.*, pp. 230–238, Sep 2012, DOI: 10.1109/CLUSTER.2012.35.

[178] P. Diaconis and B. Efron, "*Computer-intensive methods in statistics*". *Sci. Am.* **248(5)**, pp. 116–130, May 1983.

[179] K. Diefendorff and P. K. Dubey, "*How multimedia workloads will change processor design*". *Computer* **30(9)**, pp. 43–45, Sep 1997, DOI: 10.1109/2.612247.

[180] M. D. Dikaiakos, A. Stassopoulou, and L. Papageorgiou, "*An investigation of web crawler behavior: Characterization and metrics*". *Comput. Commun.* **28(8)**, pp. 880–897, May 2005, DOI: 10.1016/j.comcom.2005.01.003.

[181] P. A. Dinda, G. Memik, R. P. Dick, B. Lin, A. Mallik, A. Gupta, and S. Rossoff, "*The user in experimental computer systems research*". In *Workshop Experimental Comput. Sci.*, art. 10, Jun 2007, DOI: 10.1145/1281700.1281710.

[182] C. Ding and Y. Zhong, "*Predicting whole-program locality through reuse distance analysis*". In *Proc. SIGPLAN Conf. Prog. Lang. Design & Implementation*, pp. 247–257, Jun 2003, DOI: 10.1145/781131.781159.

[183] R. Dixon and T. Sherwood, "*Whiteboards that compute: A workload analysis*". In *IEEE Intl. Symp. Workload Characterization*, pp. 69–78, Sep 2008, DOI: 10.1109/IISWC.2008.4636092.

[184] W. J. Doherty and A. J. Thadani, *The economic value of rapid response time*. Tech. Rep. GE20-1752-0, IBM, Nov 1982.

[185] D. Donato, F. Bonchi, T. Chi, and Y. Maarek, "*Do you want to take notes? Identifying research missions in Yahoo! search pad*". In 19th *Intl. World Wide Web Conf.*, pp. 321–330, Apr 2010, DOI: 10.1145/1772690.1772724.

[186] A. B. Downey, "*Predicting queue times on space-sharing parallel computers*". In 11th *Intl. Parallel Processing Symp.*, pp. 209–218, Apr 1997, DOI: 10.1109/IPPS.1997.580894.

[187] A. B. Downey, "*A parallel workload model and its implications for processor allocation*". *Cluster Comput.* **1(1)**, pp. 133–145, 1998, DOI: 10.1023/A:1019077214124.

[188] A. B. Downey, "*The structural cause of file size distributions*". In 9th *Modeling, Anal. & Simulation of Comput. & Telecomm. Syst.*, pp. 361–370, Aug 2001, DOI: 10.1109/MASCOT.2001.948888.

[189] A. B. Downey, "*Lognormal and Pareto distributions in the Internet*". *Comput. Commun.* **28(7)**, pp. 790–801, May 2005, DOI: 10.1016/j.comcom.2004.11.001.

[190] A. B. Downey and D. G. Feitelson, "*The elusive goal of workload characterization*". *Performance Evaluation Rev.* **26(4)**, pp. 14–29, Mar 1999, DOI: 10.1145/309746.309750.

[191] D. Downey, S. Dumais, and E. Horvitz, "*Models of searching and browsing: Languages, studies, and applications*". In 20th *Intl. Joint Conf. Artificial Intelligence*, pp. 1465–1472, Jan 2007.

[192] F. Duarte, B. Mattos, A. Bestavros, V. Almeida, and J. Almeida, "*Traffic characteristics and communication patterns in blogosphere*". In 1st *Intl. Conf. Weblogs & Social Media*, Mar 2007.

[193] J. J. Dujmovic, "*Universal benchmark suites*". In 7th *Modeling, Anal. & Simulation of Comput. & Telecomm. Syst.*, pp. 197–205, Oct 1999, DOI: 10.1109/MASCOT.1999.805056.

[194] O. Duskin and D. G. Feitelson, "*Distinguishing humans from robots in web search logs: Preliminary results using query rates and intervals*". In 2nd *Workshop on Web Search Click Data*, pp. 15–19, Feb 2009, DOI: 10.1145/1507509.1507512.

[195] M. R. Ebling and M. Satyanarayanan, "*SynRGen: An extensible file reference generator*". In *SIGMETRICS Conf. Measurement & Modeling of Comput. Syst.*, pp. 108–117, May 1994, DOI: 10.1145/183018.183030.

[196] J. Eeckhout, "*Gibrat's law for (all) cities*". *Am. Economic Rev.* **94(5)**, pp. 1429–1451, Dec 2004, DOI: 10.1257/0002828043052303.

[197] L. Eeckhout, R. H. Bell Jr., B. Stougie, K. De Bosschere, and L. K. John, "*Control flow modeling in statistical simulation for accurate and efficient processor design studies*". In 31st *Ann. Intl. Symp. Computer Architecture Conf. Proc.*, pp. 350–361, Jun 2004, DOI: 10.1109/ISCA.2004.1310787.

[198] L. Eeckhout and K. De Bosschere, "*Hybrid analytical-statistical modeling for efficiently exploring architecture and workload design spaces*". In 10th *Intl. Conf. Parallel Arch. & Compilation Tech.*, pp. 25–34, Sep 2001, DOI: 10.1109/PACT.2001.953285.

[199] L. Eeckhout and K. De Bosschere, "*Quantifying behavioral differences between multimedia and general-purpose workloads*". *J. Syst. Arch.* **48(6-7)**, pp. 199–220, Jan 2003, DOI: 10.1016/S1383-7621(03)00003-1.

[200] L. Eeckhout, K. De Bosschere, and H. Neefs, "*Performance analysis through synthetic trace generation*". In *IEEE Intl. Symp. Performance Analysis Syst. & Software*, pp. 1–6, Apr 2000, DOI: 10.1109/ISPASS.2000.842273.

[201] L. Eeckhout, S. Nussbaum, J. E. Smith, and K. De Bosschere, "*Statistical simulation: Adding efficiency to the computer designer's toolbox*". *IEEE Micro* **23(5)**, pp. 26–38, Sep-Oct 2003, DOI: 10.1109/MM.2003.1240210.

[202] L. Eeckhout, H. Vandierendonck, and K. De Bosschere, "*Workload design: Selecting representative program-input pairs*". In 11th *Intl. Conf. Parallel Arch. & Compilation Tech.*, pp. 83–94, Sep 2002, DOI: 10.1109/PACT.2002.1106006.

[203] L. Eeckhout, H. Vandierendonck, and K. De Bosschere, "*Designing computer architecture research workloads*". *Computer* **36(2)**, pp. 65–71, Feb 2003, DOI: 10.1109/MC.2003.1178050.

[204] L. Eeckhout, H. Vandierendonck, and K. De Bosschere, "*Quantifying the impact of input data sets on program behavior and its applications*". *J. Instruction Level Parallelism* **5**, pp. 1–33, Apr 2003.

[205] B. Efron, "*Computers and the theory of statistics: Thinking the unthinkable*". *SIAM Rev.* **21(4)**, pp. 460–480, Oct 1979, DOI: 10.1137/1021092.

[206] B. Efron and G. Gong, "*A leisurely look at the bootstrap, the jackknife, and cross-validation*". *The American Statistician* **37(1)**, pp. 36–48, Feb 1983, DOI: 10.2307/2685844.

[207] L. Egghe, "*The power of power laws and an interpretation of Lotkaian informetric systems as self-similar fractals*". *J. Am. Soc. Inf. Sci. & Tech.* **56(7)**, pp. 669–675, May 2005, DOI: 10.1002/asi.20158.

[208] R. El Abdouni Khayari, R. Sadre, and B. R. Haverkort, "*Fitting world-wide web request traces with the EM-algorithm*". *Performance Evaluation* **52**, pp. 175–191, 2003, DOI: 10.1016/S0166-5316(02)00179-7.

[209] S. Elnaffar, P. Martin, B. Schiefer, and S. Lightstone, "*Is it DSS or OLTP: Automatically identifying DBMS workloads*". *J. Intelligent Inf. Syst.* **30(3)**, pp. 249–271, Jun 2008, DOI: 10.1007/s10844-006-0036-6.

[210] J. Elson and J. Howell, "*Handling flash crowds from your garage*". In *USENIX Ann. Tech. Conf.*, pp. 171–184, Jun 2008.

[211] D. W. Embley and G. Nagy, "*Behavioral aspects of text editors*". *ACM Comput. Surv.* **13(1)**, pp. 33–70, Mar 1981, DOI: 10.1145/356835.356838.

[212] D. M. Erceg-Hurn and V. M. Mirosevich, "*Modern robust statistical methods: An easy way to maximize the accuracy and power of your research*". *Am. Psych.* **63(7)**, pp. 591–601, Oct 2008, DOI: 10.1037/0003-066X.63.7.591.

[213] C. Ernemann, B. Song, and R. Yahyapour, "*Scaling of workload traces*". In *Job Scheduling Strategies for Parallel Processing*, pp. 166–182, Springer-Verlag, 2003, DOI: 10.1007/10968987_9. Lect. Notes Comput. Sci. vol. 2862.

[214] A. Erramilli, O. Narayan, and W. Willinger, "*Experimental queueing analysis with long-range dependent packet traffic*". *IEEE/ACM Trans. Networking* **4(2)**, pp. 209–223, Apr 1996, DOI: 10.1109/90.491008.

[215] C. Estan and G. Varghese, "*New directions in traffic measurements and accounting: Focusing on the elephants, ignoring the mice*". *ACM Trans. Comput. Syst.* **21(3)**, pp. 270–313, Aug 2003, DOI: 10.1145/859716.859719.

[216] Y. Etsion and D. G. Feitelson, "*Probabilistic prediction of temporal locality*". *IEEE Comput. Arch. Let.* **6(1)**, pp. 17–20, Jan-Jun 2007, DOI: 10.1109/L-CA.2007.5.

[217] Y. Etsion and D. G. Feitelson, "*L1 cache filtering through random selection of memory references*". In 16th *Intl. Conf. Parallel Arch. & Compilation Tech.*, pp. 235–244, Sep 2007, DOI: 10.1109/PACT.2007.44.

[218] Y. Etsion and D. G. Feitelson, "*Exploiting core working sets to filter the L1 cache with random sampling*". *IEEE Trans. Comput.* **61(11)**, pp. 1535–1550, Nov 2012, DOI: 10.1109/TC.2011.197.

[219] Y. Etsion, D. Tsafrir, S. Kirkpatrick, and D. G. Feitelson, "*Fine grained kernel logging with KLogger: Experience and insights*". In 2nd *EuroSys*, pp. 259–272, Mar 2007, DOI: 10.1145/1272996.1273023.

[220] O. Etzioni, "*Moving up the information food chain: Deploying softbots on the world wide web*". *AI Magazine* **18(2)**, pp. 11–18, Summer 1997.

[221] M. Evans, N. Hastings, and B. Peacock, *Statistical Distributions*. John Wiley & Sons, 3rd ed., 2000.

[222] B. S. Everitt, S. Landau, M. Leese, and D. Stahl, *Cluster Analysis*. John Wiley & Sons, 5th ed., 2011.

[223] M. Evers, S. J. Patel, R. S. Chappell, and Y. N. Patt, "*An analysis of correlation and predictability: What makes two-level branch predictors work*". In 25th *Ann. Intl. Symp. Computer Architecture Conf. Proc.*, pp. 52–61, Jun 1998, DOI: 10.1145/279358.279368.

[224] S. Eyerman and L. Eeckhout, "*Probabilistic job symbiosis modeling for SMT processor scheduling*". In 15th *Intl. Conf. Architect. Support for Prog. Lang. & Operating Syst.*, pp. 91–102, Mar 2010, DOI: 10.1145/1736020.1736033.

[225] A. M. Faber, M. Gupta, and C. H. Viecco, "*Revisiting web server workload invariants in the context of scientific web sites*". In *ACM/IEEE Conf. Supercomputing*, art. 110, 2006, DOI: 10.1145/1188455.1188570.

[226] M. Faloutsos, P. Faloutsos, and C. Faloutsos, "*On power-law relationships of the Internet topology*". In *ACM SIGCOMM Conf.*, pp. 251–262, Aug 1999, DOI: 10.1145/316188.316229.

[227] J. Fan, J. Xu, M. H. Ammar, and S. B. Moon, "*Prefix-preserving IP address anonymization: Measurement-based security evaluation and a new cryptography-based scheme*". *Comput. Networks* **46(2)**, pp. 253–272, Oct 2004, DOI: 10.1016/j.comnet.2004.03.033.

[228] A. Fedorova, S. Blagodurov, and S. Zhuravlev, "*Managing contention for shared resources on multicore processors*". *Comm. ACM* **53(2)**, pp. 49–57, Feb 2010, DOI: 10.1145/1646353.1646371.

[229] D. G. Feitelson, *A Survey of Scheduling in Multiprogrammed Parallel Systems*. Research Report RC 19790 (87657), IBM T. J. Watson Research Center, Oct 1994.

[230] D. G. Feitelson, "*Packing schemes for gang scheduling*". In *Job Scheduling Strategies for Parallel Processing*, pp. 89–110, Springer-Verlag, 1996, DOI: 10.1007/BFb0022289. Lect. Notes Comput. Sci. vol. 1162.

[231] D. G. Feitelson, "*Memory usage in the LANL CM-5 workload*". In *Job Scheduling Strategies for Parallel Processing*, pp. 78–94, Springer-Verlag, 1997, DOI: 10.1007/3-540-63574-2_17. Lect. Notes Comput. Sci. vol. 1291.

[232] D. G. Feitelson, "*A critique of ESP*". In *Job Scheduling Strategies for Parallel Processing*, pp. 68–73, Springer-Verlag, 2000, DOI: 10.1007/3-540-39997-6_5. Lect. Notes Comput. Sci. vol. 1911.

[233] D. G. Feitelson, "*The forgotten factor: Facts; on performance evaluation and its dependence on workloads*". In *Euro-Par 2002 Parallel Processing*, B. Monien and R. Feldmann (eds.), pp. 49–60, Springer-Verlag, Aug 2002, DOI: 10.1007/3-540-45706-2_4. Lect. Notes Comput. Sci. vol. 2400.

[234] D. G. Feitelson, "*Workload modeling for performance evaluation*". In *Performance Evaluation of Complex Systems: Techniques and Tools*, M. C. Calzarossa and S. Tucci (eds.), pp. 114–141, Springer-Verlag, Sep 2002, DOI: 10.1007/3-540-45798-4_6. Lect. Notes Comput. Sci. vol. 2459.

[235] D. G. Feitelson, "*Metric and workload effects on computer systems evaluation*". *Computer* **36(9)**, pp. 18–25, Sep 2003, DOI: 10.1109/MC.2003.1231190.

[236] D. G. Feitelson, "*A distributional measure of correlation*". *InterStat* Dec 2004. URL http://interstat.statjournals.net/YEAR/2004/abstracts/0412001.php.

[237] D. G. Feitelson, "*Experimental analysis of the root causes of performance evaluation results: A backfilling case study*". *IEEE Trans. Parallel & Distributed Syst.* **16(2)**, pp. 175–182, Feb 2005, DOI: 10.1109/TPDS.2005.18.

[238] D. G. Feitelson, "*Metrics for mass-count disparity*". In 14th *Modeling, Anal. & Simulation of Comput. & Telecomm. Syst.*, pp. 61–68, Sep 2006, DOI: 10.1109/MASCOTS.2006.30.

[239] D. G. Feitelson, "*Locality of sampling and diversity in parallel system workloads*". In 21st *Intl. Conf. Supercomputing*, pp. 53–63, Jun 2007, DOI: 10.1145/1274971.1274982.

[240] D. G. Feitelson and M. A. Jette, "*Improved utilization and responsiveness with gang scheduling*". In *Job Scheduling Strategies for Parallel Processing*, pp. 238–261, Springer-Verlag, 1997, DOI: 10.1007/3-540-63574-2_24. Lect. Notes Comput. Sci. vol. 1291.

[241] D. G. Feitelson and A. W. Mu'alem, "*On the definition of "on-line" in job scheduling problems*". *SIGACT News* **36(1)**, pp. 122–131, Mar 2005, DOI: 10.1145/1052796.1052797.

[242] D. G. Feitelson and A. Mu'alem Weil, "*Utilization and predictability in scheduling the IBM SP2 with backfilling*". In 12th *Intl. Parallel Processing Symp.*, pp. 542–546, Apr 1998, DOI: 10.1109/IPPS.1998.669970.

[243] D. G. Feitelson and M. Naaman, "*Self-tuning systems*". *IEEE Softw.* **16(2)**, pp. 52–60, Mar/Apr 1999, DOI: 10.1109/52.754053.

[244] D. G. Feitelson and B. Nitzberg, "*Job characteristics of a production parallel scientific workload on the NASA Ames iPSC/860*". In *Job Scheduling Strategies for*

Parallel Processing, pp. 337–360, Springer-Verlag, 1995, DOI: 10.1007/3-540-60153-8_38. Lect. Notes Comput. Sci. vol. 949.

[245] D. G. Feitelson and L. Rudolph, "*Evaluation of design choices for gang scheduling using distributed hierarchical control*". *J. Parallel & Distributed Comput.* **35(1)**, pp. 18–34, May 1996, DOI: 10.1006/jpdc.1996.0064.

[246] D. G. Feitelson and L. Rudolph, "*Toward convergence in job schedulers for parallel supercomputers*". In *Job Scheduling Strategies for Parallel Processing*, pp. 1–26, Springer-Verlag, 1996, DOI: 10.1007/BFb0022284. Lect. Notes Comput. Sci. vol. 1162.

[247] D. G. Feitelson and L. Rudolph, "*Metrics and benchmarking for parallel job scheduling*". In *Job Scheduling Strategies for Parallel Processing*, pp. 1–24, Springer-Verlag, 1998, DOI: 10.1007/BFb0053978. Lect. Notes Comput. Sci. vol. 1459.

[248] D. G. Feitelson and E. Shmueli, "*A case for conservative workload modeling: Parallel job scheduling with daily cycles of activity*". In 17th *Modeling, Anal. & Simulation of Comput. & Telecomm. Syst.*, Sep 2009, DOI: 10.1109/MASCOT.2009.5366139.

[249] D. G. Feitelson and D. Tsafrir, "*Workload sanitation for performance evaluation*". In *IEEE Intl. Symp. Performance Analysis Syst. & Software*, pp. 221–230, Mar 2006, DOI: 10.1109/ISPASS.2006.1620806.

[250] D. G. Feitelson, D. Tsafrir, and D. Krakov, "*Experience with using the Parallel Workloads Archive*". *J. Parallel & Distributed Comput.* **74(10)**, pp. 2967–2982, Oct 2014, DOI: 10.1016/j.jpdc.2014.06.013.

[251] A. Feldmann, A. C. Gilbert, and W. Willinger, "*Data networks as cascades: Investigating the multifractal nature of Internet WAN traffic*". In *ACM SIGCOMM Conf.*, pp. 42–55, Sep 1998, DOI: 10.1145/285243.285256.

[252] A. Feldmann, A. C. Gilbert, W. Willinger, and T. G. Kurtz, "*The changing nature of network traffic: Scaling phenomena*". *Comput. Commun. Rev.* **28(2)**, pp. 5–29, Apr 1998, DOI: 10.1145/279345.279346.

[253] A. Feldmann and W. Whitt, "*Fitting mixtures of exponentials to long-tail distributions to analyze network performance models*". *Performance Evaluation* **31(3-4)**, pp. 245–279, Jan 1998, DOI: 10.1016/S0166-5316(97)00003-5.

[254] W. Feller, *An Introduction to Probability Theory and Its Applications*, vol. II. John Wiley & Sons, 2nd ed., 1971.

[255] T. Fenner, M. Levene, and G. Loizou, "*A stochastic model for the evolution of the web allowing link deletion*". *ACM Trans. Internet Technology* **6(2)**, pp. 117–130, May 2006, DOI: 10.1145/1149121.1149122.

[256] D. Ferrari, "*Workload characterization and selection in computer performance measurement*". *Computer* **5(4)**, pp. 18–24, Jul/Aug 1972, DOI: 10.1109/C-M.1972.216939.

[257] D. Ferrari, "*A generative model of working set dynamics*". In *SIGMETRICS Conf. Measurement & Modeling of Comput. Syst.*, pp. 52–57, Sep 1981, DOI: 10.1145/1010629.805474.

[258] D. Ferrari, "*On the foundations of artificial workload design*". In *SIGMETRICS Conf. Measurement & Modeling of Comput. Syst.*, pp. 8–14, Aug 1984, DOI: 10.1145/1031382.809309.

[259] K. Ferschweiler, M. Calzarossa, C. Pancake, D. Tessera, and D. Keon, "*A community databank for performance tracefiles*". In *Recent Advances in Parallel Virtual Machine and Message Passing Interface*, Y. Cotronis and J. Dongarra (eds.), pp. 233–240, Springer-Verlag, Sep 2001, DOI: 10.1007/3-540-45417-9_33. Lect. Notes Comput. Sci. vol. 2131.

[260] D. R. Figueiredo, B. Liu, A. Feldmann, V. Misra, D. Towsley, and W. Willinger, "*On TCP and self-similar traffic*". *Performance Evaluation* **61(2-3)**, pp. 129–141, Jul 2005, DOI: 10.1016/j.peva.2004.11.004.

[261] A. Finamore, M. Mellia, M. Meo, M. M. Munafò, and D. Rossi, "*Experiences of Internet traffic monitoring with Tstat*". *IEEE Network* **25(3)**, pp. 8–14, May-Jun 2011, DOI: 10.1109/MNET.2011.5772055.

[262] W. Fischer and K. Meier-Hellstern, "*The Markov-modulated Poisson process (MMPP) cookbook*". *Performance Evaluation* **18(2)**, pp. 149–171, Sep 1993, DOI: 10.1016/0166-5316(93)90035-S.

[263] P. J. Fleming and J. J. Wallace, "*How not to lie with statistics: The correct way to summarize benchmark results*". *Comm. ACM* **29(3)**, pp. 218–221, Mar 1986.

[264] S. Floyd, "*Simulation is crucial*". *IEEE Spectrum* **38(1)**, p. 76, Jan 2001.

[265] S. Floyd and V. Paxson, "*Difficulties in simulating the Internet*". *IEEE/ACM Trans. Networking* **9(4)**, pp. 392–403, Aug 2001, DOI: 10.1109/90.944338.

[266] R. Fonseca, V. Almeida, and M. Crovella, "*Locality in a web of streams*". *Comm. ACM* **48(1)**, pp. 82–88, Jan 2005, DOI: 10.1145/1039539.1039543.

[267] R. Fonseca, V. Almeida, M. Crovella, and B. Abrahao, "*On the intrinsic locality properties of web reference streams*". In 22nd *IEEE INFOCOM*, vol. 1, pp. 448–458, Mar 2003, DOI: 10.1109/INFCOM.2003.1208696.

[268] R. Fox and M. S. Taqqu, "*Large-sample properties of parameter estimates for strongly dependent stationary Gaussian time series*". *Ann. Statist.* **14(2)**, pp. 517–532, Jun 1986, DOI: 10.1214/aos/1176349936.

[269] E. Frachtenberg, D. G. Feitelson, F. Petrini, and J. Fernandez, "*Adaptive parallel job scheduling with flexible coscheduling*". *IEEE Trans. Parallel & Distributed Syst.* **16(11)**, pp. 1066–1077, Nov 2005, DOI: 10.1109/TPDS.2005.130.

[270] U. Frisch and D. Sornette, "*Extreme deviations and applications*". *J. Phys. I France* **7(9)**, pp. 1155–1171, Sep 1997, DOI: 10.1051/jp1:1997114.

[271] K. W. Froese and R. B. Bunt, "*The effect of client caching on file server workloads*". In 29th *Hawaii Intl. Conf. System Sciences*, vol. 1, pp. 150–159, Jan 1996, DOI: 10.1109/HICSS.1996.495458.

[272] E. Fuchs and P. E. Jackson, "*Estimates of distributions of random variables for certain computer communications traffic models*". *Comm. ACM* **13(12)**, pp. 752–757, Dec 1970, DOI: 10.1145/362814.362830.

[273] X. Gabaix, "*Zipf's law for cities: An explanation*". *Quart. J. Economics* **114(3)**, pp. 739–767, Aug 1999, DOI: 10.1162/003355399556133.

[274] M. Gaertler, "*Clustering*". In *Network Analysis: Methodological Foundations*, U. Brandes and T. Erlebach (eds.), chap. 8, pp. 178–215, Springer-Verlag, 2005, DOI: 10.1007/978-3-540-31955-9_8. Lect. Notes Comput. Sci. vol. 3418.

[275] A. Ganapathi, Y. Chen, A. Fox, R. Katz, and D. Patterson, "*Statistics-driven workload modeling for the cloud*". In 5th *Intl. Workshop Self-Managing Database Syst.*, pp. 87–92, Mar 2010, DOI: 10.1109/ICDEW.2010.5452742. (At 26th IEEE Intl. Conf. Data Engineering).

[276] A. Gandhi, T. Zhu, M. Harchol-Balter, and M. A. Kozuch, "*SOFTScale: Stealing opportunistically for transient scaling*". In 13th *Intl. Middleware Conf.*, pp. 142–163, Springer-Verlag, Dec 2012, DOI: 10.1007/978-3-642-35170-9_8. Lect. Notes Comput. Sci. vol. 7662.

[277] G. R. Ganger and Y. N. Patt, "*Using system-level models to evaluate I/O subsystem designs*". *IEEE Trans. Comput.* **47(6)**, pp. 667–678, Jun 1998, DOI: 10.1109/12.689646.

[278] D. F. García and J. García, "*TPC-W e-commerce benchmark evaluation*". *Computer* **36(2)**, pp. 42–48, Feb 2003, DOI: 10.1109/MC.2003.1178045.

[279] J. L. García-Dorado, A. Finamore, M. Mellia, M. Meo, and M. Munafò, "*Characterization of ISP traffic: Trends, user habits, and access technology impact*". *IEEE Trans. Network & Service Management* **9(2)**, pp. 142–155, Jun 2012, DOI: 10.1109/TNSM.2012.022412.110184.

[280] A. Gattiker, F. H. Gebara, H. P. Hofstee, J. D. Hayes, and A. Hylick, "*Big Data text-oriented banchmark creation for Hadoop*". *IBM J. Res. Dev.* **57(3/4)**, art. 10, May/Jul 2013, DOI: 10.1147/JRD.2013.2240732.

[281] N. Geens, J. Huysmans, and J. Vanthienen, "*Evaluation of web robot discovery techniques: A benchmarking study*". In 6th *Industrial Conf. Data Mining*, pp. 121–130, Jul 2006, DOI: 10.1007/11790853_10. Lect. Notes Comput. Sci. vol. 4065.

[282] R. S. Geiger and A. Halfaker, "*Using edit sessions to measure participation in Wikipedia*". In *Conf. Comput. Supported Cooperative Work*, pp. 861–870, Feb 2013, DOI: 10.1145/2441776.2441873.

[283] J. Geweke and S. Porter-Hudak, "*The estimation and application of long memory time series models*". *J. Time Series Analysis* **4(4)**, pp. 221–238, Jul 1983, DOI: 10.1111/j.1467-9892.1983.tb00371.x.

[284] R. Giladi and N. Ahituv, "*SPEC as a performance evaluation measure*". *Computer* **28(8)**, pp. 33–42, Aug 1995, DOI: 10.1109/2.402073.

[285] P. Gill, M. Arlitt, Z. Li, and A. Mahanti, "*YouTube traffic characterization: A view from the edge*". In 7th *Internet Measurement Conf.*, pp. 15–28, Oct 2007, DOI: 10.1145/1298306.1298310.

[286] R. Giorgi, C. A. Prete, G. Prina, and L. Ricciardi, "*Trace factory: Generating workloads for trace-driven simulation of shared-bus multiprocessors*". *IEEE Concurrency* **5(4)**, pp. 54–68, Oct-Dec 1997, DOI: 10.1109/4434.641627.

[287] G. Glass and P. Cao, "*Adaptive page replacement based on memory reference behavior*". In *SIGMETRICS Conf. Measurement & Modeling of Comput. Syst.*, pp. 115–126, Jun 1997, DOI: 10.1145/258623.258681.

[288] A. Gluhak, S. Krco, M. Nati, D. Pfisterer, N. Mitton, and T. Razafindralambo, "*A survey on facilities for experimental internet of things research*". *IEEE Commun. Mag.* **49(11)**, pp. 58–67, Nov 2011, DOI: 10.1109/MCOM.2011.6069710.

[289] C. Gniady, A. R. Butt, and Y. C. Hu, "*Program-counter-based pattern classification in buffer caching*". In 6th *Symp. Operating Systems Design & Implementation*, pp. 395–408, Dec 2004.

[290] C. M. Goldie and C. Klüppelberg, "*Subexponential distributions*". In *A Practical Guide to Heavy Tails*, R. J. Adler, R. E. Feldman, and M. S. Taqqu (eds.), pp. 436–459, Birkhäuser, 1998.

[291] M. L. Goldstein, S. A. Morris, and G. G. Yen, "*Problems with fitting to the power-law distribution*". *European Phys. J. B* **41(2)**, pp. 255–258, Sep 2004, DOI: 10.1140/epjb/e2004-00316-5.

[292] C. P. Gomes, B. Selman, N. Crato, and H. Kautz, "*Heavy-tailed phenomena in satisfiability and constraint satisfaction problems*". *J. Automated Reasoning* **24**, pp. 67–100, Feb 2000, DOI: 10.1023/A:1006314320276.

[293] M. E. Gómez and V. Santonja, "*Analysis of self-similarity in I/O workload using structural modeling*". In 7th *Modeling, Anal. & Simulation of Comput. & Telecomm. Syst.*, pp. 234–242, Oct 1999, DOI: 10.1109/MASCOT.1999.805060.

[294] M. E. Gómez and V. Santonja, "*A new approach in the analysis and modeling of disk access patterns*". In *IEEE Intl. Symp. Performance Analysis Syst. & Software*, pp. 172–177, Apr 2000, DOI: 10.1109/ISPASS.2000.842297.

[295] W.-B. Gong, Y. Liu, V. Misra, and D. Towsley, "*Self-similarity and long range dependence on the Internet: A second look at the evidence, origins and implications*". *Comput. Networks* **48(3)**, pp. 377–399, Jun 2005, DOI: 10.1016/j.comnet.2004.11.026.

[296] M. Gopshtein and D. G. Feitelson, "*Empirical quantification of opportunities for content adaptation in web servers*". In 3rd *Haifa Experimental Syst. Conf.*, art. 5, May 2010, DOI: 10.1145/1815695.1815702.

[297] M. Gopshtein and D. G. Feitelson, "*Trading off quality for throughput using content adaptation in web servers*". In 4th *Intl. Conf. Systems & Storage*, art. 6, May 2011, DOI: 10.1145/1987816.1987824.

[298] K. Goševa-Popstojanova, A. D. Singh, S. Mazimdar, and F. Li, "*Empirical characterization of session-based workload and reliability for web servers*". *Empirical Softw. Eng.* **11(1)**, pp. 71–117, Mar 2006, DOI: 10.1007/s10664-006-5966-7.

[299] C. W. J. Granger and R. Joyeux, "*An introduction to long-memory time series models and fractional differencing*". *J. Time Series Analysis* **1(1)**, pp. 15–29, Jan 1980, DOI: 10.1111/j.1467-9892.1980.tb00297.x.

[300] R. M. Gray, "*On the asymptotic eigenvalue distribution of Toeplitz matrices*". *IEEE Trans. Information Theory* **IT-18(6)**, pp. 725–730, Nov 1972, DOI: 10.1109/TIT.1972.1054924.

[301] M. Greenwald, "*Practical algorithms for self scaling histograms or better than average data collection*". *Performance Evaluation* **27&28**, pp. 19–40, Oct 1996, DOI: 10.1016/S0166-5316(96)90018-8.

[302] M. Greiner, M. Jobmann, and L. Lipsky, "*The importance of power-tail distributions for modeling queueing systems*". *Operations Research* **47(2)**, pp. 313–326, Mar/Apr 1999, DOI: 10.1287/opre.47.2.313.

[303] S. D. Gribble and E. A. Brewer, "*System design issues for internet middleware services: Deductions from a large client trace*". In *Symp. Internet Technologies & Syst.*, pp. 207–218, Dec 1997.

[304] S. D. Gribble, G. S. Manku, D. Roselli, E. A. Brewer, T. J. Gibson, and E. L. Miller, "*Self-similarity in file systems*". In *SIGMETRICS Conf. Measurement & Modeling of Comput. Syst.*, pp. 141–150, Jun 1998, DOI: 10.1145/277851.277894.

[305] G. R. Grimmett and D. R. Stirzaker, *Probability and Random Processes*. Oxford University Press, 2nd ed., 1992.

[306] K. Grimsrud, J. Archibald, R. Frost, and B. Nelson, "*Locality as a visualization tool*". *IEEE Trans. Comput.* **45(11)**, pp. 1319–1326, Nov 1996, DOI: 10.1109/12.544490.

[307] D. Gross, J. F. Shortle, M. J. Fisher, and D. M. B. Masi, "*Difficulties in simulating queues with Pareto service*". In *Winter Simul. Conf.*, vol. 1, pp. 407–415, Dec 2002, DOI: 10.1109/WSC.2002.1172911.

[308] J. A. Gubner, "*Theorems and fallacies in the theory of long-range-dependent processes*". *IEEE Trans. Information Theory* **51(3)**, pp. 1234–1239, Mar 2005, DOI: 10.1109/TIT.2004.842768.

[309] K. P. Gummadi, R. J. Dunn, S. Saroiu, S. D. Gribble, H. M. Levy, and J. Zahorjan, "*Measurement, modeling, and analysis of a peer-to-peer file-sharing workload*". In 19th *Symp. Operating Systems Principles*, pp. 314–329, Oct 2003, DOI: 10.1145/945445.945475.

[310] L. Guo, S. Chen, Z. Xiao, and X. Zhang, "*Analysis of multimedia workloads with implications for Internet streaming*". In 14th *Intl. World Wide Web Conf.*, pp. 519–528, May 2005, DOI: 10.1145/1060745.1060821.

[311] L. Guo, M. Crovella, and I. Matta, "*How does TCP generate pseudo-self-similarity?*" In 9th *Modeling, Anal. & Simulation of Comput. & Telecomm. Syst.*, pp. 215–223, Aug 2001, DOI: 10.1109/MASCOT.2001.948871.

[312] L. Guo, S. Jiang, L. Xiao, and X. Zhang, "*Fast and low-cost search schemes by exploiting localities in P2P networks*". *J. Parallel & Distributed Comput.* **65(6)**, pp. 729–742, Jun 2005, DOI: 10.1016/j.jpdc.2005.01.007.

[313] A. Gupta, B. Lin, and P. A. Dinda, "*Measuring and understanding user comfort with resource borrowing*". In 13th *Intl. Symp. High Performance Distributed Comput.*, pp. 214–224, Jun 2004, DOI: 10.1109/HPDC.2004.1323536.

[314] V. Gupta, M. Burroughs, and M. Harchol-Balter, "*Analysis of scheduling policies under correlated job sizes*". *Performance Evaluation* **67(11)**, pp. 996–1013, Nov 2010, DOI: 10.1016/j.peva.2010.08.010.

[315] R. Gusella, "*Characterizing the variability of arrival processes with indices of dispersion*". *IEEE J. Select Areas in Commun.* **9(2)**, pp. 203–211, Feb 1991, DOI: 10.1109/49.68448.

[316] M. R. Guthaus, J. S. Ringenberg, D. Ernst, T. M. Austin, T. Mudge, and R. B. Brown, "*MiBench: A free, commercially representative embedded benchmark suit*". In 4th *Workshop on Workload Characterization*, pp. 3–14, Dec 2001, DOI: 10.1109/WWC.2001.990739.

[317] M. Harchol-Balter, "*Task assignment with unknown duration*". *J. ACM* **49(2)**, pp. 260–288, Mar 2002, DOI: 10.1145/506147.506154.

[318] M. Harchol-Balter, N. Bansal, B. Schroeder, and M. Agrawal, "*SRPT scheduling for web servers*". In *Job Scheduling Strategies for Parallel Processing*, pp. 11–20, Springer-Verlag, 2001, DOI: 10.1007/3-540-45540-X_2. Lect. Notes Comput. Sci. vol. 2221.

[319] M. Harchol-Balter, M. E. Crovella, and C. D. Murta, "*On choosing a task assignment policy for a distributed server system*". *J. Parallel & Distributed Comput.* **59(2)**, pp. 204–228, Nov 1999, DOI: 10.1006/jpdc.1999.1577.

[320] M. Harchol-Balter and A. B. Downey, "*Exploiting process lifetime distributions for dynamic load balancing*". *ACM Trans. Comput. Syst.* **15(3)**, pp. 253–285, Aug 1997, DOI: 10.1145/263326.263344.

[321] H. Haugerud and S. Straumsnes, "*Simulation of user-driven computer behavior*". In 15th *Systems Administration Conf. (LISA)*, pp. 101–107, Dec 2001.

[322] M. Haungs, P. Sallee, and M. Farrens, "*Branch transition rate: A new metric for improved branch classification analysis*". In 6th *Intl. Symp. High Performance Comput. Arch.*, pp. 241–250, Jan 2000, DOI: 10.1109/HPCA.2000.824354.

[323] B. Hayes, "*The Britney Spears problem*". *American Scientist* **96(4)**, pp. 274–279, Jul-Aug 2008, DOI: 10.1511/2008.73.3822.

[324] D. He and A. Göker, "*Detecting session boundaries from web user logs*". In 22nd *BCS-IRSG Ann. Colloq. Information Retrieval Research*, pp. 57–66, 2000.

[325] J. Heffernan and J. Tawn, "*Extreme values in the dock*". *Significance* **1(1)**, pp. 13–17, Mar 2004, DOI: 10.1111/j.1740-9713.2004.00003.x.

[326] J. E. Heffernan and J. A. Tawn, "*An extreme value analysis for the investigation into the sinking of the M. V. Derbyshire*". *J. Royal Stat. Soc. C: App. Stat.* **52(3)**, pp. 337–354, Jul 2003, DOI: 10.1111/1467-9876.00408.

[327] T. Henderson and S. Bhatti, "*Modelling user behavior in networked games*". In 9th *ACM Intl. Conf. Multimedia*, pp. 212–220, Sep 2001, DOI: 10.1145/500141.500175.

[328] T. Henderson, D. Kotz, and I. Abyzov, "*The changing nature of a mature campus-wide wireless network*". In 10th *Mobile Comput. & Networking*, pp. 187–201, Sep 2004, DOI: 10.1145/1023720.1023739.

[329] J. L. Henning, "*SPEC CPU2000: Measuring CPU performance in the new millenium*". *Computer* **33(7)**, pp. 28–35, Jul 2000, DOI: 10.1109/2.869367.

[330] F. Hernández-Campos, K. Jeffay, and F. D. Smith, "*Tracking the evolution of web traffic: 1995–2003*". In 11th *Modeling, Anal. & Simulation of Comput. & Telecomm. Syst.*, pp. 16–25, Oct 2003, DOI: 10.1109/MASCOT.2003.1240638.

[331] F. Hernández-Campos, K. Jeffay, and F. D. Smith, "*Modeling and generating TCP application workloads*". In 4th *Broadband Comm., Netw. & Syst.*, pp. 280–289, Sep 2007, DOI: 10.1109/BROADNETS.2007.4550436.

[332] F. Hernández-Campos, J. S. Marron, G. Samorodnitsky, and F. D. Smith, "*Variable heavy tailed distributions in Internet traffic, part I: Understanding heavy tails*". In 10th *Modeling, Anal. & Simulation of Comput. & Telecomm. Syst.*, pp. 43–52, Oct 2002, DOI: 10.1109/MASCOT.2002.1167059.

[333] H. Herodotou and S. Babu, "*Profiling, what-if analysis, and cost-based optimization of MapReduce programs*". *Proc. VLDB Endowment* **4(11)**, pp. 1111–1122, Aug 2011.

[334] B. M. Hill, "*A simple general approach to inference about the tail of a distribution*". *Ann. Statist.* **3(5)**, pp. 1163–1174, Sep 1975.

[335] M. D. Hill and J. R. Larus, "*Cache considerations for multiprocessor programmers*". *Comm. ACM* **33(8)**, pp. 97–102, Aug 1990, DOI: 10.1145/79173.79180.

[336] H. Hlavacs and G. Kotsis, "*Modeling user behavior: A layered approach*". In 7th *Modeling, Anal. & Simulation of Comput. & Telecomm. Syst.*, pp. 218–225, Oct 1999, DOI: 10.1109/MASCOT.1999.805058.

[337] H. P. Hofstee, G. C. Chen, F. H. Gebara, K. Hall, J. Herring, D. Jamsek, J. Li, Y. Li, J. W. Shi, and P. W. Y. Wong, "*Understanding system design for Big Data workloads*". *IBM J. Res. Dev.* **57(3/4)**, art. 3, May/Jul 2013, DOI: 10.1147/JRD.2013.2242674.

[338] N. Hohn, D. Veitch, and P. Abry, "*Does fractal scaling at the IP level depend on TCP flow arrival processes?*" In 2nd *Internet Measurement Workshop*, pp. 63–68, Nov 2002, DOI: 10.1145/637201.637208.

[339] J. K. Hollingsworth, B. P. Miller, and J. Cargille, "*Dynamic program instrumentation for scalable performance tools*". In *Scalable High-Performance Comput. Conf.*, pp. 841–850, May 1994, DOI: 10.1109/SHPCC.1994.296728.

[340] G. Horn, A. Kvalbein, J. Blomskøld, and E. Nilsen, "*An empirical comparison of generators for self similar simulated traffic*". *Performance Evaluation* **64(2)**, pp. 162–190, Feb 2007, DOI: 10.1016/j.peva.2006.06.005.

[341] A. Horváth and M. Telek, "*Markovian modeling of real data traffic: Heuristic phase type and MAP fitting of heavy tailed and fractal like samples*". In *Performance Evaluation of Complex Systems: Techniques and Tools*, M. C. Calzarossa and S. Tucci (eds.), pp. 405–434, Springer-Verlag, 2002, DOI: 10.1007/3-540-45798-4_17. Lect. Notes Comput. Sci. vol. 2459.

[342] A. Horváth and M. Telek, "*PhFit: A general phase-type fitting tool*". In *Computer Performance Evaluation: Modelling Techniques and Tools*, T. Field et al. (eds.), pp. 82–91, Springer-Verlag, 2002, DOI: 10.1007/3-540-46029-2_5. Lect. Notes Comput. Sci. vol. 2324.

[343] J. R. M. Hosking, "*Fractional differencing*". *Biometrika* **68(1)**, pp. 165–176, Apr 1981, DOI: 10.1093/biomet/68.1.165.

[344] J. R. M. Hosking, "*Modeling persistence in hydrological time series using fractional differencing*". *Water Resources Res.* **20(12)**, pp. 1989–1908, Dec 1984, DOI: 10.1029/WR020i012p01898.

[345] K. Hoste and L. Eeckhout, "*Microarchitecture-independent workload characterization*". *IEEE Micro* **27(3)**, pp. 63–72, May/Jun 2007, DOI: 10.1109/MM.2007.56.

[346] S. Hotovy, "*Workload evolution on the Cornell Theory Center IBM SP2*". In *Job Scheduling Strategies for Parallel Processing*, pp. 27–40, Springer-Verlag, 1996, DOI: 10.1007/BFb0022285. Lect. Notes Comput. Sci. vol. 1162.

[347] W. W. Hsu and A. J. Smith, "*Characteristics of I/O traffic in personal computer and server workloads*". *IBM Syst. J.* **42(2)**, pp. 347–372, 2003, DOI: 10.1147/sj.422.0347.

[348] W. W. Hsu, A. J. Smith, and H. C. Young, "*I/O reference behavior of production database workloads and the TPC benchmarks — an analysis at the logical level*". *ACM Trans. Database Syst.* **26(1)**, pp. 96–143, Mar 2001, DOI: 10.1145/383734.383737.

[349] W. W. Hsu, A. J. Smith, and H. C. Young, "*Characteristics of production database workloads and the TCP benchmarks*". *IBM Syst. J.* **40(3)**, pp. 781–802, 2001, DOI: 10.1147/sj.403.0781.

[350] W. W. Hsu, A. J. Smith, and H. C. Young, "*The automatic improvement of locality in storage systems*". *ACM Trans. Comput. Syst.* **23(4)**, pp. 424–473, Nov 2005, DOI: 10.1145/1113574.1113577.

[351] S. Huang, J. Huang, J. Dai, T. Xie, and B. Huang, "*The HiBench benchmark suite: Characterization of the MapReduce-based data analysis*". In 2nd *Intl. Workshop Inf. & Softw. as Services*, pp. 41–51, Mar 2010, DOI: 10.1109/ICDEW.2010.5452747.

[352] B. A. Huberman and L. A. Adamic, *Evolutionary Dynamics of the World-Wide Web*. Tech. rep., Xerox Palo Alto Research Center, Jan 1999. (http://arxiv.org/abs/cond-mat/9901071v2).

[353] B. A. Huberman and L. A. Adamic, "*Growth dynamics of the world-wide web*". *Nature* **401**, p. 131, Sep 1999, DOI: 10.1038/43604.

[354] H. E. Hurst, "*Long-term storage capacity of reservoirs*". *Trans. Am. Soc. Civil Engineers* **116**, pp. 770–808, 1951.

[355] A. Hussain, A. Kapoor, and J. Heidemann, "*The effect of detail on Ethernet simulation*". In 18th *Parallel & Distrib. Sim.*, pp. 97–104, May 2004, DOI: 10.1109/PADS.2004.1301290.

[356] T. Huynh and J. Miller, "*Empirical observations on the session timeout threshold*". *Inf. Process. & Management* **45(5)**, pp. 513–528, Sep 2009, DOI: 10.1016/j.ipm.2009.04.007.

[357] A. Iosup and D. Epema, "*Grid computing workloads*". *IEEE Internet Comput.* **15(2)**, pp. 19–26, Mar/Apr 2011, DOI: 10.1109/MIC.2010.130.

[358] A. Iosup, D. H. J. Epema, C. Franke, A. Papaspyrou, L. Schley, B. Song, and R. Yahyapour, "*On grid performance evaluation using synthetic workloads*". In *Job Scheduling Strategies for Parallel Processing*, E. Frachtenberg and U. Schwiegelshohn (eds.), pp. 232–255, Springer-Verlag, 2006, DOI: 10.1007/978-3-540-71035-6_12. Lect. Notes Comput. Sci. vol. 4376.

[359] A. Iosup, M. Jan, O. Sonmez, and D. Epema, "*The characteristics and performance of groups of jobs in grids*". In *EuroPar*, pp. 382–393, Aug 2007, DOI: 10.1007/978-3-540-74466-5_42.

[360] A. Iosup, H. Li, M. Jan, S. Anoep, C. Dumitrescu, L. Wolters, and D. H. J. Epema, "*The grid workloads archive*". *Future Generation Comput. Syst.* **24(7)**, pp. 672–686, May 2008, DOI: 10.1016/j.future.2008.02.003.

[361] G. Irlam, "*Unix file size survey - 1993*". URL http://www.gordoni.com/ufs93.html.

[362] A. K. Iyengar, E. A. MacNair, M. S. Squillante, and L. Zhang, "*A general methodology for characterizing access patterns and analyzing web server performance*". In 6th *Modeling, Anal. & Simulation of Comput. & Telecomm. Syst.*, pp. 167–174, Jul 1998, DOI: 10.1109/MASCOT.1998.693691.

[363] V. S. Iyengar, L. H. Trevillyan, and P. Bose, "*Representative traces for processor models with infinite caches*". In 2nd *High Performance Comput. Arch.*, pp. 62–72, Feb 1996, DOI: 10.1109/HPCA.1996.501174.

[364] V. Jacobson, "*Congestion avoidance and control*". In *ACM SIGCOMM Conf.*, pp. 314–329, Aug 1988, DOI: 10.1145/52325.52356.

[365] A. K. Jain, M. N. Murty, and P. J. Flynn, "*Data clustering: A review*". *ACM Comput. Surv.* **31(3)**, pp. 264–323, Sep 1999, DOI: 10.1145/331499.331504.

[366] R. Jain, "*Characteristics of destination address locality in computer networks: A comparison of caching schemes*". *Computer Networks and ISDN Systems* **18(4)**, pp. 243–254, May 1990, DOI: 10.1016/0169-7552(90)90106-3.

[367] R. Jain, *The Art of Computer Systems Performance Analysis*. John Wiley & Sons, 1991.

[368] R. Jain and I. Chlamtac, "*The P^2 algorithm for dynamic calculation of quantiles and histograms without storing observations*". *Comm. ACM* **28(10)**, pp. 1076–1085, Oct 1985, DOI: 10.1145/4372.4378.

[369] R. Jain and S. A. Routhier, "*Packet trains—measurements and a new model for computer network traffic*". *IEEE J. Select Areas in Commun.* **SEC-4(6)**, pp. 986–995, Sep 1986, DOI: 10.1109/JSAC.1986.1146410.

[370] J. Jann, P. Pattnaik, H. Franke, F. Wang, J. Skovira, and J. Riodan, "*Modeling of workload in MPPs*". In *Job Scheduling Strategies for Parallel Processing*, pp. 95–116, Springer-Verlag, 1997, DOI: 10.1007/3-540-63574-2_18. Lect. Notes Comput. Sci. vol. 1291.

[371] B. J. Jansen, D. L. Booth, and A. Spink, "*Determining the informational, navigational, and transactional intent of web queries*". *Inf. Process. & Management* **44(3)**, pp. 1251–1266, May 2008, DOI: 10.1016/j.ipm.2007.07.015.

[372] B. J. Jansen, T. Mullen, A. Spink, and J. Pedersen, "*Automated gathering of web information: An in-depth examination of agents interacting with search engines*". *ACM Trans. Internet Technology* **6(4)**, pp. 442–464, Nov 2006, DOI: 10.1145/1183463.1183468.

[373] B. J. Jansen and A. Spink, "*An analysis of web searching by European AlltheWeb.com users*". *Inf. Process. & Management* **41(2)**, pp. 361–381, Mar 2005, DOI: 10.1016/S0306-4573(03)00067-0.

[374] B. J. Jansen, A. Spink, C. Blakely, and S. Koshman, "*Defining a session on web search engines*". *J. Am. Soc. Inf. Sci. & Tech.* **58(6)**, pp. 862–871, Apr 2007, DOI: 10.1002/asi.20564.

[375] B. J. Jansen, A. Spink, and J. Pedersen, "*A temporal comparison of AltaVista web searching*". *J. Am. Soc. Inf. Sci. & Tech.* **56(6)**, pp. 559–570, 2005, DOI: 10.1002/asi.20145.

[376] B. J. Jansen, A. Spink, and T. Saracevic, "*Real life, real users, and real needs: A study and analysis of user queries on the web*". *Inf. Process. & Management* **36**, pp. 207–227, 2000, DOI: 10.1016/S0306-4573(99)00056-4.

[377] M. Jeon, Y. Kim, J. Hwang, J. Lee, and E. Seo, "*Workload characterization and performance implications of large-scale blog servers*". *ACM Trans. Web* **6(4)**, art. 16, Nov 2012, DOI: 10.1145/2382616.2382619.

[378] H.-D. J. Jeong, J.-S. R. Lee, D. McNickle, and K. Pawlikowski, "*Suggestions of efficient self-similar generators*". *Simulation Modeling Practice & Theory* **15(3)**, pp. 328–353, Mar 2007, DOI: 10.1016/j.simpat.2006.10.002.

[379] H.-D. J. Jeong, J.-S. R. Lee, D. McNickle, and K. Pawlikowski, "*Comparison of various estimators in simulated FGN*". *Simulation Modeling Practice & Theory* **15(9)**, pp. 1173–1191, Oct 2007, DOI: 10.1016/j.simpat.2007.08.004.

[380] C. Jiang, Z. Yu, H. Jin, C. Xu, L. Eeckhout, W. Heirman, T. E. Carlson, and X. Liao, "*PCantorSim: Accelerating parallel architecture simulation through*

fractal-based sampling". *ACM Trans. Arch. & Code Optimization* **10(4)**, art. 49, Dec 2013, DOI: 10.1145/2541228.2555305.

[381] S. Jin and A. Bestavros, "*Sources and characteristics of web temporal locality*". In 8th *Modeling, Anal. & Simulation of Comput. & Telecomm. Syst.*, pp. 28–35, Aug 2000, DOI: 10.1109/MASCOT.2000.876426.

[382] F. T. Johnsen, T. Hafsøe, C. Griwodz, and P. Halvorsen, "*Workload characterization for news-on-demand streaming services*". In 26th *IEEE Intl. Performance, Comput. & Commun. Conf.*, pp. 314–323, Apr 2007, DOI: 10.1109/PCCC.2007.358909.

[383] N. L. Johnson, "*Systems of frequency curves generated by methods of translation*". *Biometrika* **36(1/2)**, pp. 149–176, Jun 1949.

[384] M. S. Johnstone and P. R. Wilson, "*The memory fragmentation problem: Solved?*" In 1st *Intl. Symp. Memory Management*, pp. 26–36, Oct 1998, DOI: 10.1145/301589.286864.

[385] R. Jones and K. L. Klinkner, "*Beyond the session timeout: Automatic hierarchical segmentation of search topics in query logs*". In 17th *ACM Conf. Inf. & Knowledge Management*, pp. 699–708, Oct 2008, DOI: 10.1145/1458082.1458176.

[386] R. Jones, R. Kumar, B. Pang, and A. Tomkins, "*"I know what you did last summer" — query logs and user privacy*". In 16th *Conf. Inf. & Knowledge Mgmt.*, pp. 909–914, Nov 2007, DOI: 10.1145/1321440.1321573.

[387] A. Joshi, L. Eeckhout, R. H. Bell Jr., and L. K. John, "*Distilling the essence of proprietary workloads into miniature benchmarks*". *ACM Trans. Arch. & Code Optimization* **5(2)**, art. 10, Aug 2008, DOI: 10.1145/1400112.1400115.

[388] A. Joshi, A. Phansalkar, L. Eeckhout, and L. K. John, "*Measuring benchmark similarity using inherent program characteristics*". *IEEE Trans. Comput.* **55(6)**, pp. 769–782, Jun 2006, DOI: 10.1109/TC.2006.85.

[389] J. Jung, B. Krishnamurthy, and M. Rabinovich, "*Flash crowds and denial of service attacks: Characterization and implications for CDNs and web sites*". In 11th *Intl. World Wide Web Conf.*, pp. 293–304, May 2002, DOI: 10.1145/511446.511485.

[390] N. Kammenhuber, J. Luxenburger, A. Feldmann, and G. Weikum, "*Web search clickstreams*". In 6th *Internet Measurement Conf.*, pp. 245–250, Oct 2006, DOI: 10.1145/1177080.1177110.

[391] E. L. Kaplan and P. Meier, "*Nonparametric estimation from incomplete observations*". *J. Am. Stat. Assoc.* **53(282)**, pp. 457–481, Jun 1958.

[392] T. Karagiannis, A. Broido, N. Brownlee, k. claffy, and M. Faloutsos, "*Is P2P dying or just hiding?*" In *IEEE Globecom*, vol. 3, pp. 1532–1538, Nov 2004, DOI: 10.1109/GLOCOM.2004.1378239.

[393] T. Karagiannis, A. Broido, M. Faloutsos, and K. claffy, "*Transport layer identification of P2P traffic*". In 4th *Internet Measurement Conf.*, pp. 121–134, Oct 2004, DOI: 10.1145/1028788.1028804.

[394] T. Karagiannis, M. Faloutsos, and M. Molle, "*A user-friendly self-similarity analysis tool*". *Comput. Commun. Rev.* **33(3)**, pp. 81–93, Jul 2003, DOI: 10.1145/956993.957004.

[395] T. Karagiannis, M. Molle, and M. Faloutsos, "*Long-range dependence: Ten years of Internet traffic modeling*". *IEEE Internet Computing* **8(5)**, pp. 57–64, Sep-Oct 2004, DOI: 10.1109/MIC.2004.46.

[396] D. Karamshuk, C. Boldrini, M. Conti, and A. Passarella, "*Human mobility models for opportunistic networks*". *IEEE Commun. Mag.* **49(12)**, pp. 157–165, Dec 2011, DOI: 10.1109/MCOM.2011.6094021.

[397] S. Kavalanekar, B. Worthington, Q. Zhang, and V. Sharda, "*Characterization of storage workload traces from production Windows servers*". In *IEEE Intl. Symp. Workload Characterization*, pp. 119–128, Sep 2008, DOI: 10.1109/IISWC.2008.4636097.

[398] K. Keeton, D. A. Patterson, Y. Q. He, R. C. Raphael, and W. E. Baker, "*Performance characterization of a quad Pentium Pro SMP using OLTP workloads*". In 25th *Ann. Intl. Symp. Computer Architecture Conf. Proc.*, pp. 15–26, Jun 1998, DOI: 10.1145/279361.279364.

[399] M. G. Kendall, "*A new measure of rank correlation*". *Biometrika* **30(1-2)**, pp. 81–93, 1938, DOI: 10.1093/biomet/30.1-2.81.

[400] M. G. Kendall and B. Babington Smith, "*The problem of m rankings*". *Ann. Math. Statist.* **10(3)**, pp. 275–287, Sep 1939.

[401] R. E. Kessler, M. D. Hill, and D. A. Wood, "*A comparison of trace-sampling techniques for multi-megabyte caches*". *IEEE Trans. Comput.* **43(6)**, pp. 664–675, Jun 1994, DOI: 10.1109/12.286300.

[402] M. Kim, D. Kotz, and S. Kim, "*Extracting a mobility model from real user traces*". In 25th *IEEE INFOCOM*, Apr 2006, DOI: 10.1109/INFOCOM.2006.173.

[403] D. N. Kimelman and T. A. Ngo, "*The RP3 program visualization environment*". *IBM J. Res. Dev.* **35(5/6)**, pp. 635–651, Sep/Nov 1991, DOI: 10.1147/rd.355.0635.

[404] D. L. Kiskis and K. G. Shin, "*SWSL: A synthetic workload specification language for real-time systems*". *IEEE Trans. Softw. Eng.* **20(10)**, pp. 798–811, Oct 1994, DOI: 10.1109/32.328992.

[405] S. D. Kleban and S. H. Clearwater, "*Hierarchical dynamics, interarrival times, and performance*". In *Supercomputing*, art. 28, Nov 2003, DOI: 10.1109/SC.2003.10040.

[406] A. J. KleinOsowski and D. J. Lilja, "*MinneSPEC: A new SPEC benchmark workload for simulation-based computer architecture research*". *IEEE Comput. Arch. Let.* **1(1)**, 2002, DOI: 10.1109/L-CA.2002.8.

[407] B. Klimt and Y. Yang, "*The Enron corpus: A new dataset for email classification research*". In 15th *European Conf. Machine Learning*, pp. 217–226, Springer-Verlag, 2004, DOI: 10.1007/978-3-540-30115-8_22. Lect. Notes Comput. Sci. vol. 3201.

[408] R. Koch, *The 80/20 Principle: The Secret to Achieving More with Less*. Doubleday, 1998.

[409] T. Kohno, A. Broido, and K. C. Claffy, "*Remote physical device fingerprinting*". *IEEE Trans. Dependable & Secure Comput.* **2(2)**, pp. 93–108, Apr-Jun 2005, DOI: 10.1109/TDSC.2005.26.

[410] E. J. Koldinger, S. J. Eggers, and H. M. Levy, "*On the validity of trace-driven simulation for multiprocessors*". In 18th *Ann. Intl. Symp. Computer Architecture Conf. Proc.*, pp. 244–253, May 1991, DOI: 10.1145/115953.115977.

[411] R. Koller, A. Verma, and R. Rangaswami, "*Generalized ERSS tree model: Revisiting working sets*". *Performance Evaluation* **67(11)**, pp. 1139–1154, Nov 2010, DOI: 10.1016/j.peva.2010.08.004.

[412] G. Kotsis, "*A systematic approach for workload modeling for parallel processing systems*". *Parallel Comput.* **22(13)**, pp. 1771–1787, Feb 1997, DOI: 10.1016/S0167-8191(96)00076-2.

[413] D. Kotz and C. S. Ellis, "*Practical prefetching techniques for parallel file systems*". In 1st *Intl. Conf. Parallel & Distributed Inf. Syst.*, pp. 182–189, Dec 1991, DOI: 10.1109/PDIS.1991.183101.

[414] D. Kotz and K. Essien, "*Analysis of a campus-wide wireless network*". *Wireless Netw.* **11(1-2)**, pp. 115–133, Jan 2005, DOI: 10.1007/s11276-004-4750-0.

[415] D. Kotz, C. Newport, R. S. Gray, J. Liu, Y. Yuan, and C. Elliot, "*Experimental evaluation of wireless simulation assumptions*". In 7th *ACM Intl. Symp. Modeling, Analysis, and Simul. Wireless & Mobile Syst.*, pp. 78–82, Oct 2004, DOI: 10.1145/1023663.1023679.

[416] D. Kotz and N. Nieuwejaar, "*File-system workload on a scientific multiprocessor*". *IEEE Parallel & Distributed Technology* **3(1)**, pp. 51–60, Spring 1995, DOI: 10.1109/88.384584.

[417] D. Krishnamurthy, J. A. Rolia, and S. Majumdar, "*A synthetic workload generation technique for stress testing session-based systems*". *IEEE Trans. Softw. Eng.* **32(11)**, pp. 868–882, Nov 2006, DOI: 10.1109/TSE.2006.106.

[418] P. Krueger and R. Chawla, "*The stealth distributed scheduler*". In 11th *Intl. Conf. Distributed Comput. Syst.*, pp. 336–343, May 1991, DOI: 10.1109/ICDCS.1991.148686.

[419] P. Krueger, T.-H. Lai, and V. A. Dixit-Radiya, "*Job scheduling is more important than processor allocation for hypercube computers*". *IEEE Trans. Parallel & Distributed Syst.* **5(5)**, pp. 488–497, May 1994, DOI: 10.1109/71.282559.

[420] Z. Kurmas, K. Keeton, and K. Mackenzie, "*Synthesizing representative I/O workloads using iterative distillation*". In 11th *Modeling, Anal. & Simulation of Comput. & Telecomm. Syst.*, pp. 6–15, Oct 2003, DOI: 10.1109/MASCOT.2003.1240637.

[421] D. Kushner, "*Playing dirty*". *IEEE Spectrum* **44(12INT)**, pp. 30–35, Dec 2007, DOI: 10.1109/MSPEC.2007.4390020.

[422] T. Lafage and A. Seznec, "*Choosing representative slices of program execution for microarchitecture simulations: A preliminary application to the data stream*". In 3rd *Workshop on Workload Characterization*, Sep 2000.

[423] J. Laherrére and D. Sornette, "*Stretched exponential distributions in nature and economy: "fat tails" with characteristic scales*". *European Physical J. B* **2(4)**, pp. 525–539, May 1998, DOI: 10.1007/s100510050276.

[424] M. S. Lam, E. E. Rothberg, and M. E. Wolf, "*The cache performance and optimizations of blocked algorithms*". In 4th *Intl. Conf. Architect. Support for Prog. Lang. & Operating Syst.*, pp. 63–74, Apr 1991, DOI: 10.1145/106972.106981.

[425] G. N. Lambert, "*A comparative study of system response time on program developer productivity*". *IBM Syst. J.* **23(1)**, pp. 36–43, 1984, DOI: 10.1147/sj.231.0036.

[426] K.-C. Lan and J. Heidemann, "*Rapid model parameterization from traffic measurements*". *ACM Trans. Modeling & Comput. Simulation* **12(3)**, pp. 201–229, Jul 2002, DOI: 10.1145/643114.643117.

[427] A. M. Law and W. D. Kelton, *Simulation Modeling and Analysis*. McGraw-Hill, 3rd ed., 2000.

[428] E. D. Lazowska, "*The use of percentiles in modeling CPU service time distributions*". In *Computer Performance*, K. M. Chandy and M. Reiser (eds.), pp. 53–66, North-Holland, 1977.

[429] J.-Y. Le Boudec, *Performance Evaluation of Computer and Communication Systems*. EPFL Press, 2010. (Available from http://perfeval.epfl.ch/).

[430] C. Lee, M. Potkonjak, and W. H. Mangione-Smith, "*MediaBench: A tool for evaluating and synthesizing multimedia and communication systems*". In 13th *Intl. Symp. Microarchitecture*, pp. 330–335, Dec 1997, DOI: 10.1109/MICRO.1997.645830.

[431] C. B. Lee, Y. Schwartzman, J. Hardy, and A. Snavely, "*Are user runtime estimates inherently inaccurate?*" In *Job Scheduling Strategies for Parallel Processing*, pp. 253–263, Springer-Verlag, 2004, DOI: 10.1007/11407522_14. Lect. Notes Comput. Sci. vol. 3277.

[432] D. C. Lee, P. J. Crowley, J.-L. Baer, T. E. Anderson, and B. N. Bershad, "*Execution characteristics of desktop applications on Windows NT*". In 25th *Ann. Intl. Symp. Computer Architecture Conf. Proc.*, pp. 27–38, Jun 1998, DOI: 10.1145/279358.279366.

[433] G. Lee, C. P. Kruskal, and D. J. Kuck, "*The effectiveness of combining in shared memory parallel computers in the presence of 'hot spots'*". In *Intl. Conf. Parallel Processing*, pp. 35–41, Aug 1986.

[434] W. LeFebvre, "*CNN.com: Facing a world crisis*". *;Login:* **27(1)**, p. 83, Feb 2002. (summary of invited talk at LISA 2001).

[435] W. E. Leland and T. J. Ott, "*Load-balancing heuristics and process behavior*". In *SIGMETRICS Conf. Measurement & Modeling of Comput. Syst.*, pp. 54–69, May 1986, DOI: 10.1145/317499.317539.

[436] W. E. Leland, M. S. Taqqu, W. Willinger, and D. V. Wilson, "*On the self-similar nature of Ethernet traffic*". *IEEE/ACM Trans. Networking* **2(1)**, pp. 1–15, Feb 1994, DOI: 10.1109/90.282603.

[437] T. G. Lewis, "*Where is client/server software headed?*" *Computer* **28(4)**, pp. 49–55, Apr 1995, DOI: 10.1109/2.375177.

[438] B. Li, G. Y. Keung, S. Xie, F. Liu, Y. Sun, and H. Yin, "*An empirical study of flash crowd dynamics in a P2P-based live video streaming system*". In *IEEE Global Telecom. Conf.*, art. 339, Nov 2008, DOI: 10.1109/GLOCOM.2008.ECP.339.

[439] H. Li and R. Buyya, "*Model-driven simulation of grid scheduling strategies*". In 3rd *IEEE Intl. Conf. e-Science & Grid Comput.*, pp. 287–294, Dec 2007, DOI: 10.1109/E-SCIENCE.2007.51.

[440] H. Li, M. Muskulus, and L. Wolters, "*Modeling correlated workloads by combining model based clustering and a localized sampling algorithm*". In 21st *Intl. Conf. Supercomputing*, pp. 64–72, Jun 2007, DOI: 10.1145/1274971.1274983.

[441] W. Li, "*Random texts exhibit Zipf's-law-like word frequency distribution*". *IEEE Trans. Inf. Theory* **38(6)**, pp. 1842–1845, Nov 1992, DOI: 10.1109/18.165464.

[442] W. Li, "*Zipf's law everywhere*". *Glottometrics* **5**, pp. 14–21, 2002.

[443] D. Lifka, "*The ANL/IBM SP scheduling system*". In *Job Scheduling Strategies for Parallel Processing*, pp. 295–303, Springer-Verlag, 1995, DOI: 10.1007/3-540-60153-8_35. Lect. Notes Comput. Sci. vol. 949.

[444] D. J. Lilja, *Measuring Computer Performance: A Practitioner's Guide*. Cambridge University Press, 2000.

[445] E. Limpert, W. A. Stahel, and M. Abbt, "*Log-normal distributions across the sciences: Keys and clues*". *BioScience* **51(5)**, pp. 341–352, May 2001, DOI: 10.1641/0006-3568(2001)051[0341:LNDATS]2.0.CO;2.

[446] Z. Liu, N. Niclausse, and C. Jalpa-Villanueva, "*Traffic model and performance evaluation of web servers*". *Performance Evaluation* **46(2-3)**, pp. 77–100, Oct 2001, DOI: 10.1016/S0166-5316(01)00046-3.

[447] A. W. Lo, "*Long-term memory in stock market prices*". *Econometrica* **59(5)**, pp. 1279–1313, Sep 1991.

[448] J. L. Lo, L. A. Barroso, S. J. Eggers, K. Gharachorloo, H. M. Levy, and S. S. Parekh, "*An analysis of database workload performance on simultaneous multithreaded processors*". In 25th *Ann. Intl. Symp. Computer Architecture Conf. Proc.*, pp. 39–50, Jun 1998, DOI: 10.1145/279358.279367.

[449] V. Lo and J. Mache, "*Job scheduling for prime time vs. non-prime time*". In 4th *Intl. Conf. Cluster Comput.*, pp. 488–493, Sep 2002, DOI: 10.1109/CLUSTR.2002.1137789.

[450] V. Lo, J. Mache, and K. Windisch, "*A comparative study of real workload traces and synthetic workload models for parallel job scheduling*". In *Job Scheduling Strategies for Parallel Processing*, pp. 25–46, Springer-Verlag, 1998, DOI: 10.1007/BFb0053979. Lect. Notes Comput. Sci. vol. 1459.

[451] K. S. Lomax, "*Business failures: Another example of the analysis of failure data*". *J. Am. Stat. Assoc.* **49(268)**, pp. 847–852, Dec 1954, DOI: 10.1080/01621459.1954.10501239.

[452] M. O. Lorenz, "*Methods of measuring the concentration of wealth*". *Pub. Am. Stat. Assoc.* **9(70)**, pp. 209–219, Jun 1905.

[453] D. Lu, P. Dinda, Y. Qiao, and H. Sheng, "*Effects and implications of file size/service time correlation on web server scheduling policies*". In 13th *Modeling, Anal. & Simulation of Comput. & Telecomm. Syst.*, pp. 258–267, 2005, DOI: 10.1109/MASCOTS.2005.30.

[454] U. Lublin and D. G. Feitelson, "*The workload on parallel supercomputers: Modeling the characteristics of rigid jobs*". *J. Parallel & Distributed Comput.* **63(11)**, pp. 1105–1122, Nov 2003, DOI: 10.1016/S0743-7315(03)00108-4.

[455] A. Madhukar and C. Williamson, "*A longitudinal study of P2P traffic classification*". In 14th *Modeling, Anal. & Simulation of Comput. & Telecomm. Syst.*, pp. 179–188, Sep 2006, DOI: 10.1109/MASCOTS.2006.6.

[456] A. W. Madison and A. P. Batson, "*Characteristics of program localities*". *Comm. ACM* **19(5)**, pp. 285–294, May 1976, DOI: 10.1145/360051.360227.

[457] M. V. Mahoney and P. K. Chan, "*Learning nonstationary models of normal network traffic for detecting novel attacks*". In 8th *Intl. Conf. Knowledge Discovery and Data Mining*, pp. 376–385, Jul 2002, DOI: 10.1145/775047.775102.

[458] G. Maier, A. Feldmann, V. Paxson, and M. Allman, "*On dominant characteristics of residential broadband Internet traffic*". In 9th *Internet Measurement Conf.*, pp. 90–102, Nov 2009, DOI: 10.1145/1644893.1644904.

[459] G. Maier, A. Feldmann, V. Paxson, R. Sommer, and M. Vallentin, "*An assessment of overt malicious activity manifest in residential networks*". In 8th *Detection of Intrusion & Malware, & Vulnerability Assessment*, pp. 144–163, Springer-Verlag, Jul 2011, DOI: 10.1007/978-3-642-22424-9_9. Lect. Notes Comput. Sci. vol. 6739.

[460] G. Maier, F. Schneider, and A. Feldmann, "*A first look at mobile hand-held device traffic*". In 11th *Passive & Active Measurement Conf.*, pp. 161–170, Springer-Verlag, Apr 2010, DOI: 10.1007/978-3-642-12334-4_17. Lect. Notes Comput. Sci. vol. 6032.

[461] G. Maier, F. Schneider, and A. Feldmann, "*NAT usage in residential broadband networks*". In 12th *Passive & Active Measurement Conf.*, pp. 32–41, Mar 2011, DOI: 10.1007/978-3-642-19260-9_4. Lect. Notes Comput. Sci. vol. 6579.

[462] O. Malcai, O. Biham, P. Richmond, and S. Solomon, "*Theoretical analysis and simulations of the generalized Lotka-Volterra model*". *Phys. Rev. E* **66(3)**, pp. 031102–1–031102–6, 2002, DOI: 10.1103/PhysRevE.66.031102.

[463] A. D. Malony, D. A. Reed, and H. A. G. Wijshoff, "*Performance measurement intrusion and perturbation analysis*". *IEEE Trans. Parallel & Distributed Syst.* **3(4)**, pp. 433–450, Jul 1992, DOI: 10.1109/71.149962.

[464] B. Mandelbrot, "*A note on a class of skew distribution functions: Analysis and critique of a paper by H. A. Simon*". *Info. & Control* **2(1)**, pp. 90–99, Apr 1959, DOI: 10.1016/S0019-9958(59)90098-1.

[465] B. Mandelbrot, "*How long is the coast of Britain? Statistical self-similarity and fractional dimension*". *Science* **156(3775)**, pp. 636–638, 5 May 1967, DOI: 10.1126/science.156.3775.636.

[466] B. B. Mandelbrot, *The Fractal Geometry of Nature*. W. H. Freeman and Co., 1982.

[467] B. B. Mandelbrot and M. S. Taqqu, "*Robust R/S analysis of long-run serial correlation*". *Bulletin of the I.S.I.* **48**, pp. 69–99, 1979.

[468] B. B. Mandelbrot and J. W. van Ness, "*Fractional Brownian motions, fractional noises and applications*". *SIAM Rev.* **10(4)**, pp. 422–437, Oct 1968, DOI: 10.1137/1010093.

[469] B. B. Mandelbrot and J. R. Wallis, "*Some long-run properties of geophysical records*". *Water Resources Res.* **5(2)**, pp. 321–340, Apr 1969, DOI: 10.1029/WR005i002p00321.

[470] G. S. Manku, S. Rajagopalan, and B. G. Lindsay, "*Approximate medians and other quantiles in one pass and with limited memory*". In *SIGMOD Intl. Conf. Management of Data*, pp. 426–435, Jun 1998, DOI: 10.1145/276304.276342.

[471] E. P. Markatos, "*Visualizing working sets*". *Operating Syst. Rev.* **31(4)**, pp. 3–11, Oct 1997, DOI: 10.1145/271019.271021.

[472] E. P. Markatos and T. J. LeBlanc, "*Load balancing vs. locality management in shared-memory multiprocessors*". In *Intl. Conf. Parallel Processing*, vol. I, pp. 258–265, Aug 1992.

[473] N. Markovitch and U. R. Krieger, "*The estimation of heavy-tailed probability density functions, their mixtures and quantiles*". *Comput. Networks* **40(3)**, pp. 459–474, Oct 2002, DOI: 10.1016/S1389-1286(02)00306-7.

[474] F. J. Massey, Jr., "*The Kolmogorov-Smirnov test for goodness of fit*". *J. Am. Stat. Assoc.* **46(253)**, pp. 68–78, Mar 1951, DOI: 10.1080/01621459.1951.10500769.

[475] R. L. Mattson, J. Gecsei, D. R. Slutz, and I. L. Traiger, "*Evaluation techniques for storage hierarchies*". *IBM Syst. J.* **9(2)**, pp. 78–117, 1970, DOI: 10.1147/sj.92.0078.

[476] K. S. McKinley and O. Temam, "*Quantifying loop nest locality using SPEC'95 and the Perfect benchmarks*". *ACM Trans. Comput. Syst.* **17(4)**, pp. 288–336, Nov 1999, DOI: 10.1145/329466.329484.

[477] M. McKusick, W. Joy, S. Leffler, and R. Fabry, "*A fast file system for UNIX*". *ACM Trans. Comput. Syst.* **2(3)**, pp. 181–197, Aug 1984, DOI: 10.1145/989.990.

[478] G. McLachlan and D. Peel, *Finite Mixture Models*. John Wiley & Sons, 2000.

[479] G. J. McLachlan and T. Krishnan, *The EM Algorithm and Extensions*. John Wiley & Sons, 2nd ed., 2008.

[480] M. P. McLaughlin, "*"...the very game..." : A tutorial on mathematical modeling*". URL www.causascientia.org/math_stat/Tutorial.pdf, 1999.

[481] M. P. McLaughlin, "*A compendium of common probability distributions*". URL www.causascientia.org/math_stat/Dists/Compendium.html, 2001.

[482] O. B. McManus, A. L. Blatz, and K. L. Magleby, "*Sampling, log binning, fitting, and plotting durations of open and shut intervals from single channels and the effect of noise*". *Pflügers Archiv Euro. J. Physiology* **410(4-5)**, pp. 530–553, Nov 1987, DOI: 10.1007/BF00586537.

[483] D. Mehrzadi and D. G. Feitelson, "*On extracting session data from activity logs*". In 5th *Intl. Syst. & Storage Conf.*, art. 3, Jun 2012, DOI: 10.1145/2367589.2367592.

[484] M. Mellia, R. Lo Cigno, and F. Neri, "*Measuring IP and TCP behavior on edge nodes with Tstat*". *Comput. Networks* **47(1)**, pp. 1–21, Jan 2005, DOI: 10.1016/j.comnet.2004.06.026.

[485] D. A. Menascé, V. A. F. Almeida, and L. W. Dowdy, *Performance by Design: Computer Capacity Planning by Example*. Prentice Hall, 2004.

[486] D. A. Menascé, V. A. F. Almeida, R. Fonseca, and M. A. Mendes, "*A methodology for workload characterization of e-commerce sites*". In 1st *ACM Conf. Electronic Commerce*, pp. 119–128, Nov 1999, DOI: 10.1145/336992.337024.

[487] D. A. Menascé, V. A. F. Almeida, R. Riedi, F. Ribeiro, R. Fonseca, and W. Meira Jr., "*A hierarchical and multiscale approach to analyze E-business workloads*". *Performance Evaluation* **54(1)**, pp. 33–57, Sep 2003, DOI: 10.1016/S0166-5316(02)00228-6.

[488] A. Milenković and M. Milenković, "*An efficient single-pass trace compression technique utilizing instruction streams*". *ACM Trans. Modeling & Comput. Simulation* **17(1)**, pp. 2.1–2.17, Jan 2007, DOI: 10.1145/1189756.1189758.

[489] G. A. Miller, "*Some effects of intermittent silence*". *Am. J. Psychology* **70(2)**, pp. 311–314, Jun 1957, DOI: 10.2307/1419346.

[490] T. N. Minh and L. Walters, "*Towards a profound analysis of bags-of-tasks in parallel systems and their performance impact*". In 20th *Intl. Symp. High Performance Distributed Comput.*, pp. 111–122, Jun 2011, DOI: 10.1145/1996130.1996148.

[491] A. K. Mishra, J. L. Hellerstein, W. Cirne, and C. R. Das, "*Towards characterizing cloud backend workloads: Insights from Google compute clusters*". *Performance Evaluation Rev.* **37(4)**, pp. 34–41, Mar 2010, DOI: 10.1145/1773394.1773400.

[492] A. Mislove, M. Marcon, K. P. Gummadi, P. Druschel, and B. Bhattacharjee, "*Measurement and analysis of online social networks*". In 7th *Internet Measurement Conf.*, pp. 29–42, Oct 2007, DOI: 10.1145/1298306.1298311.

[493] T. Mitra and T.-c. Chiueh, "*Dynamic 3D graphics workload characterization and the architectural implications*". In 32nd *Intl. Symp. Microarchitecture*, pp. 62–71, Nov 1999, DOI: 10.1109/MICRO.1999.809444.

[494] S. Mitterhofer, C. Kruegel, E. Kirda, and C. Platzer, "*Server-side bot detection in massively multiplayer online games*". *IEEE Security & Privacy* **7(3)**, pp. 29–36, May 2009, DOI: 10.1109/MSP.2009.78.

[495] S. Mittnik, S. T. Rachev, and M. S. Paolella, "*Stable Paretian modeling in finance: Some empirical and theoretical aspects*". In *A Practical Guide to Heavy Tails*, R. J. Adler, R. E. Feldman, and M. S. Taqqu (eds.), pp. 79–110, Birkhäuser, 1998.

[496] M. Mitzenmacher, "*The power of two choices in randomized load balancing*". *IEEE Trans. Parallel & Distributed Syst.* **12(10)**, pp. 1094–1104, Oct 2001, DOI: 10.1109/71.963420.

[497] M. Mitzenmacher, "*A brief history of generative models for power law and lognormal distributions*". *Internet Math.* **1(2)**, pp. 226–251, 2003.

[498] M. Mitzenmacher, "*Dynamic models for file sizes and double Pareto distributions*". *Internet Math.* **1(3)**, pp. 305–333, 2004.

[499] T. Mohan, B. R. de Supinski, S. A. McKee, F. Mueller, A. Yoo, and M. Schulz, "*Identifying and exploiting spatial regularity in data memory references*". In *Supercomputing*, Nov 2003, DOI: 10.1145/1048935.1050199.

[500] A. L. Montgomery and C. Faloutsos, "*Identifying web browsing trends and patterns*". *Computer* **34(7)**, pp. 94–95, Jul 2001, DOI: 10.1109/2.933515.

[501] D. C. Montgomery and G. C. Runger, *Applied Statistics and Probability for Engineers*. John Wiley & Sons, 2nd ed., 1999.

[502] D. S. Moore, "*Tests of Chi-squared type*". In *Goodness-of-Fit Techniques*, R. B. D'Agostino and M. A. Stephens (eds.), pp. 63–95, Marcel Dekker, Inc., 1986.

[503] P. Morillo, J. M. Orduña, and M. Fernández, "*Workload characterization in multiplayer online games*". In *Intl. Conf. Computational Science & Applications*, pp. 490–499, May 2006, DOI: 10.1007/11751540_52. Lect. Notes Comput. Sci. vol. 3980.

[504] R. Morris and Y. C. Tay, *A Model for Analyzing the Roles of Network and User Behavior in Congestion Control*. Tech. Rep. MIT-LCS-TR898, MIT Lab. Computer Science, May 2003.

[505] MPI Forum, "*MPI: A message-passing interface standard*". *Intl. J. Supercomput. Appl. & High Performance Comput.* **8(3/4)**, pp. 159–416, Fall/Winter 1994.

[506] MPI Forum, "*MPI2: A message passing interface standard*". *Intl. J. High Performance Comput. Appl.* **12(1/2)**, pp. 1–299, Spring/Summer 1998.

[507] A. W. Mu'alem and D. G. Feitelson, "*Utilization, predictability, workloads, and user runtime estimates in scheduling the IBM SP2 with backfilling*". *IEEE Trans. Parallel & Distributed Syst.* **12(6)**, pp. 529–543, Jun 2001, DOI: 10.1109/71.932708.

[508] R. Murphy, A. Rodrigues, P. Kogge, and K. Underwood, "*The implications of working set analysis on supercomputing memory hierarchy design*". In 19th *Intl. Conf. Supercomputing*, pp. 332–340, Jun 2005, DOI: 10.1145/1088149.1088193.

[509] G. C. Murray, J. Lin, and A. Chowdhury, "*Identification of user sessions with hierarchical agglomerative clustering*". In *Proc. Am. Soc. Inf. Sci. & Tech.*, vol. 43, 2006, DOI: 10.1002/meet.14504301312.

[510] M. Mutka and M. Livny, "*The available capacity of a privately owned workstation environment*". *Performance Evaluation* **12(4)**, pp. 269–284, Jul 1991, DOI: 10.1016/0166-5316(91)90005-N.

[511] A. Nazir, S. Raza, and C.-N. Chuah, "*Unveiling Facebook: A measurement study of social based applications*". In 8th *Internet Measurement Conf.*, pp. 43–56, Oct 2008, DOI: 10.1145/1452520.1452527.

[512] A. A. Neath and J. E. Cavanaugh, "*The Bayesian information criterion: Background, derivation, and applications*". *WIREs Comput. Stat.* **4(2)**, pp. 199–203, Mar/Apr 2012, DOI: 10.1002/wics.199.

[513] M. E. J. Newman, "*The structure and function of complex networks*". *SIAM Rev.* **45(2)**, pp. 167–256, 2003, DOI: 10.1137/S003614450342480.

[514] M. E. J. Newman, "*Power laws, Pareto distributions and Zipf's law*". *Contemporary Physics* **46(5)**, pp. 323–351, Sep-Oct 2005, DOI: 10.1080/00107510500052444.

[515] T. D. Nguyen, R. Vaswani, and J. Zahorjan, "*Parallel application characterization for multiprocessor scheduling policy design*". In *Job Scheduling Strategies for Parallel Processing*, pp. 175–199, Springer-Verlag, 1996, DOI: 10.1007/BFb0022294. Lect. Notes Comput. Sci. vol. 1162.

[516] N. Nieuwejaar and D. Kotz, "*Low-level interfaces for high-level parallel I/O*". In *Input/Output in Parallel and Distributed Computer Systems*, R. Jain, J. Werth, and J. C. Browne (eds.), pp. 205–223, Kluwer Academic Publishers, 1996.

[517] N. Nieuwejaar, D. Kotz, A. Purakayastha, C. S. Ellis, and M. L. Best, "*File-access characteristics of parallel scientific workloads*". *IEEE Trans. Parallel & Distributed Syst.* **7(10)**, pp. 1075–1089, Oct 1996, DOI: 10.1109/71.539739.

[518] G. E. Noether, "*Why Kendall tau?*" *Teaching Statistics* **3(2)**, pp. 41–43, May 1981, DOI: 10.1111/j.1467-9639.1981.tb00422.x.

[519] J. P. Nolan, "*Univariate stable distributions: Parameterization and software*". In *A Practical Guide to Heavy Tails*, R. J. Adler, R. E. Feldman, and M. S. Taqqu (eds.), pp. 527–533, Birkhäuser, 1998.

[520] J. P. Nolan, *Stable Distributions: Models for Heavy Tailed Data*. Birkhäusear, 2015. (In progress).

[521] D. Nurmi, J. Brevik, and R. Wolski, "*QBETS: Queue bounds estimation from time series*". In *Job Scheduling Strategies for Parallel Processing*, E. Frachtenberg and U. Schwiegelshohn (eds.), pp. 76–101, Springer-Verlag, Jun 2007, DOI: 10.1007/978-3-540-78699-3_5. Lect. Notes Comput. Sci. vol. 4942.

[522] C. Nuzman, I. Saniee, W. Sweldens, and A. Weiss, "*A compound model for TCP connection arrivals for LAN and WAN applications*". *Comput. Networks* **40(3)**, pp. 319–337, Oct 2002, DOI: 10.1016/S1389-1286(02)00298-0.

[523] P. Ohm, D. Sicker, and D. Grunwald, "*Legal issues surrounding monitoring during network research*". In 7th *Internet Measurement Conf.*, pp. 141–148, Oct 2007, DOI: 10.1145/1298306.1298307.

[524] M. Oikonomakos, K. Christodoulopoulos, and E. M. Varvarigos, "*Profiling computation jobs in grid systems*". In 7th *IEEE Intl. Symp. Cluster Comput. & Grid*, pp. 197–204, May 2007, DOI: 10.1109/CCGRID.2007.87.

[525] A. Oke and R. Bunt, "*Hierarchical workload characterization for a busy web server*". In *TOOLS*, T. Field et al. (eds.), pp. 309–328, Springer-Verlag, Apr 2002, DOI: 10.1007/3-540-46029-2_23. Lect. Notes Comput. Sci. vol. 2324.

[526] R. Osman and W. J. Knottenbelt, "*Database system performance evaluation models: A survey*". *Performance Evaluation* **69(10)**, pp. 471–493, Oct 2012, DOI: 10.1016/j.peva.2012.05.006.

[527] T. Osogami and M. Harchol-Balter, "*Necessary and sufficient conditions for representing general distributions by Coxians*". In *Computer Performance Evaluations, Modelling Techniques and Tools*, P. Kemper and W. H. Sanders (eds.), pp. 182–199, Springer-Verlag, Sep 2003, DOI: 10.1007/978-3-540-45232-4_12. Lect. Notes Comput. Sci. vol. 2794.

[528] T. Osogami and M. Harchol-Balter, "*A closed-form solution for mapping general distributions to minimal PH distributions*". In *Computer Performance Evaluations, Modelling Techniques and Tools*, P. Kemper and W. H. Sanders (eds.), pp. 200–217, Springer-Verlag, 2003, DOI: 10.1007/978-3-540-45232-4_13. Lect. Notes Comput. Sci. vol. 2794.

[529] J. K. Ousterhout, "*Scheduling techniques for concurrent systems*". In 3rd *Intl. Conf. Distributed Comput. Syst.*, pp. 22–30, Oct 1982.

[530] J. K. Ousterhout, H. Da Costa, D. Harrison, J. A. Kunze, M. Kupfer, and J. G. Thompson, "*A trace-driven analysis of the UNIX 4.2 BSD file system*". In 10th *Symp. Operating Systems Principles*, pp. 15–24, Dec 1985, DOI: 10.1145/323627.323631.

[531] F. Paganini, Z. Wang, J. C. Doyle, and S. H. Low, "*Congestion control for high performance, stability and fairness in general networks*". *IEEE/ACM Trans. Networking* **13(1)**, pp. 43–56, Feb 2005, DOI: 10.1109/TNET.2004.842216.

[532] R. Pang, M. Allman, M. Bennett, J. Lee, V. Paxson, and B. Tierney, "*A first look at modern enterprise traffic*". In 5th *Internet Measurement Conf.*, pp. 15–28, Oct 2005.

[533] R. Pang, M. Allman, V. Paxson, and J. Lee, "*The devil and packet trace anonymization*". *Comput. Commun. Rev.* **36(1)**, pp. 29–38, Jan 2006, DOI: 10.1145/1111322.1111330.

[534] "*Parallel Workloads Archive*". URL http://www.cs.huji.ac.il/labs/parallel/workload/.

[535] C. Park, F. Hernández-Campos, J. S. Marron, and F. D. Smith, "*Long-range dependence in a changing Internet traffic mix*". *Comput. Networks* **48(3)**, pp. 401–422, Jun 2005, DOI: 10.1016/j.comnet.2004.11.018.

[536] C. Park, H. Shen, J. S. Marron, F. Hernández-Campos, and D. Veitch, "*Capturing the elusive Poissonity in web traffic*". In 14th *Modeling, Anal. & Simulation of Comput. & Telecomm. Syst.*, pp. 189–196, Sep 2006, DOI: 10.1109/MASCOTS.2006.17.

[537] K. Park and W. Willinger (eds.), *Self-Similar Network Traffic and Performance Evaluation*. John Wiley & Sons, 2000.

[538] K. Park and W. Willinger, "*Self-similar network traffic: An overview*". In *Self-Similar Network Traffic and Performance Evaluation*, K. Park and W. Willinger (eds.), pp. 1–38, John Wiley & Sons, 2000, DOI: 10.1002/047120644X.ch1.

[539] G. Pass, A. Chowdhury, and C. Torgeson, "*A picture of search*". In 1st *Intl. Conf. Scalable Information Syst.*, Jun 2006, DOI: 10.1145/1146847.1146848.

[540] V. Paxson and S. Floyd, "*Wide-area traffic: The failure of Poisson modeling*". *IEEE/ACM Trans. Networking* **3(3)**, pp. 226–244, Jun 1995, DOI: 10.1109/90.392383.

[541] A. Peleg and U. Weiser, "*MMX technology extension to the Intel architecture*". *IEEE Micro* **16(4)**, pp. 42–50, Aug 1996, DOI: 10.1109/40.526924.

[542] A. Peleg, S. Wilkie, and U. Weiser, "*Intel MMX for multimedia PCs*". *Comm. ACM* **40(1)**, pp. 25–38, Jan 1997, DOI: 10.1145/242857.242865.

[543] V. G. J. Peris, M. S. Squillante, and V. K. Naik, "*Analysis of the impact of memory in distributed parallel processing systems*". In *SIGMETRICS Conf. Measurement & Modeling of Comput. Syst.*, pp. 5–18, May 1994, DOI: 10.1145/183018.183021.

[544] R. Perline, "*Zipf's law, the central limit theorem, and the random division of the unit interval*". *Phys. Rev. E* **54(1)**, pp. 220–223, Jul 1996, DOI: 10.1103/PhysRevE.54.220.

[545] E. E. Peters, *Fractal Market Analysis*. John Wiley & Sons, 1994.

[546] F. Petrini, E. Frachtenberg, A. Hoisie, and S. Coll, "*Performance evaluation of the Quadrics interconnection network*". *Cluster Comput.* **6(2)**, pp. 125–142, Apr 2003, DOI: 10.1023/A:1022852505633.

[547] G. F. Pfister and V. A. Norton, "*"Hot-spot" contention and combining in multistage interconnection networks*". *IEEE Trans. Comput.* **C-34(10)**, pp. 943–948, Oct 1985, DOI: 10.1109/TC.1985.6312198.

[548] V. Phalke and B. Gopinath, "*An inter-reference gap model for temporal locality in program behavior*". In *SIGMETRICS Conf. Measurement & Modeling of Comput. Syst.*, pp. 291–300, May 1995, DOI: 10.1145/223587.223620.

[549] P. Pierce, "*A concurrent file system for a highly parallel mass storage subsystem*". In 4th *Conf. Hypercubes, Concurrent Comput., & Appl.*, vol. I, pp. 155–160, Mar 1989.

[550] E. Pilkington and A. Jha, "*Google predicts spread of flu using huge search data*". The Guardian, 13 Nov 2008. URL http://www.guardian.co.uk/technology/2008/nov/13/google-internet.

[551] B. Piwowarski and H. Zaragoza, "*Predictive user click models based on click-through history*". In 16th *Conf. Inf. & Knowledge Mgmt.*, pp. 175–182, Nov 2007, DOI: 10.1145/1321440.1321467.

[552] R. S. Prasad and C. Dovrolis, "*Measuring the congesion responsiveness of Internet traffic*". In 8th *Passive & Active Measurement Conf.*, pp. 176–185, Apr 2007, DOI: 10.1007/978-3-540-71617-4_18.

[553] C. A. Prete, G. Prina, and L. Ricciardi, "*A trace-driven simulator for performance evaluation of cache-based multiprocessor systems*". *IEEE Trans. Parallel & Distributed Syst.* **6(9)**, pp. 915–929, Sep 1995, DOI: 10.1109/71.466630.

[554] D. d. S. Price, "*A general theory of bibliometric and other cumulative advantage processes*". *J. Am. Soc. Inf. Sci.* **27(5)**, pp. 292–306, Sep 1976, DOI: 10.1002/asi.4630270505.

[555] A. Purakayastha, C. S. Ellis, D. Kotz, N. Nieuwejaar, and M. Best, "*Characterizing parallel file-access patterns on a large-scale multiprocessor*". In 9th *Intl. Parallel Processing Symp.*, pp. 165–172, Apr 1995, DOI: 10.1109/IPPS.1995.395928.

[556] K. E. E. Raatikainen, "*Cluster analysis and workload classification*". *Performance Evaluation Rev.* **20(4)**, pp. 24–30, May 1993, DOI: 10.1145/155775.155781.

[557] L. R. Rabiner, "*A tutorial on hidden Markov models and selected applications in speech recognition*". *Proc. IEEE* **77(2)**, pp. 257–286, Feb 1989, DOI: 10.1109/5.18626.

[558] S. V. Raghavan, D. Vasukiammaiyar, and G. Haring, "*Generative workload models for a single server environment*". In *SIGMETRICS Conf. Measurement & Modeling of Comput. Syst.*, pp. 118–127, May 1994, DOI: 10.1145/183018.183031.

[559] S. Ramany, R. Honicky, and D. Sawyer, "*Workload modeling of stateful protocols using HMMs*". In 31st *Comput. Measurement Group Intl. Conf.*, pp. 673–682, Dec 2005.

[560] R. Ramaswamy and T. Wolf, "*High-speed prefix-preserving IP address anonymization for passive measurement systems*". *IEEE/ACM Trans. Networking* **15(1)**, pp. 26–39, Feb 2007, DOI: 10.1109/TNET.2006.890128.

[561] R. A. Redner and H. F. Walker, "*Mixture densities, maximum likelihood and the EM algorithm*". *SIAM Rev.* **26(2)**, pp. 195–239, Apr 1984, DOI: 10.1137/1026034.

[562] W. J. Reed and B. D. Hughes, "*From gene families and genera to incomes and internet file sizes: Why power laws are so common in nature*". *Phys. Rev. E* **66(6)**, pp. 067103–1–067103–4, 2002, DOI: 10.1103/PhysRevE.66.067103.

[563] C. Reiss, A. Tumanov, G. R. Ganger, R. H. Katz, and M. A. Kozuch, "*Heterogeneity and dynamicity of clouds at scale: Google trace analysis*". In 3rd *Symp. Cloud Comput.*, art. 7, Oct 2012, DOI: 10.1145/2391229.2391236.

[564] C. Reiss, J. Wilkes, and J. L. Hellerstein, "*Obfuscatory obscanturism: Making workload traces of commercially-sensitive systems safe to release*". In *Network Operations & Management Symp.*, pp. 1279–1286, Apr 2012, DOI: 10.1109/NOMS.2012.6212064.

[565] S. I. Resnick, "*Why non-linearities can ruin the heavy-tailed modeler's day*". In *A Practical Guide to Heavy Tails*, R. J. Adler, R. E. Feldman, and M. S. Taqqu (eds.), pp. 219–239, Birkhäuser, 1998.

[566] L. F. Richardson, "*Variation of the frequency of fatal quarrels with magnitude*". *J. Am. Stat. Assoc.* **43(244)**, pp. 523–546, Dec 1948, DOI: 10.2307/2280704.

[567] R. H. Riedi, M. S. Crouse, V. J. Ribeiro, and R. G. Baraniuk, "*A multifractal wavelet model with applications to network traffic*". *IEEE Trans. Information Theory* **45(3)**, pp. 992–1018, Apr 1999, DOI: 10.1109/18.761337.

[568] A. Riska, V. Diev, and E. Smirni, "*Efficient fitting of long-tailed data sets into phase-type distributions*". *Performance Evaluation Rev.* **30(3)**, pp. 6–8, Dec 2002, DOI: 10.1145/605521.605525.

[569] A. Riska, V. Diev, and E. Smirni, "*An EM-based technique for approximating long-tailed data sets with PH distributions*". *Performance Evaluation* **55(1-2)**, pp. 147–164, Jan 2004, DOI: 10.1016/S0166-5316(03)00101-9.

[570] A. Riska and E. Riedel, "*Disk drive level workload characterization*". In *USENIX Ann. Tech. Conf.*, pp. 97–102, Jun 2006.

[571] A. Riska and E. Smirni, "*M/G/1-type Markov processes: A tutorial*". In *Performance Evaluation of Complex Systems: Techniques and Tools*, M. C. Calzarossa and S. Tucci (eds.), pp. 36–63, Springer-Verlag, 2002, DOI: 10.1007/3-540-45798-4_3. Lect. Notes Comput. Sci. vol. 2459.

[572] C. Roadknight, I. Marshall, and D. Vearer, "*File popularity characterisation*". *Performance Evaluation Rev.* **27(4)**, pp. 45–50, Mar 2000, DOI: 10.1145/346000.346014.

[573] F. Robinson, A. Apon, D. Brewer, L. Dowdy, D. Hoffman, and B. Lu, "*Initial starting point analysis for k-means clustering: A case study*". In *Conf. Applied Research in Information Tech.*, Mar 2006.

[574] J. Roca, V. Moya, C. Gonzáles, C. Solís, A. Fernández, and R. Espasa, "*Workload characterization of 3D games*". In *IEEE Intl. Symp. Workload Characterization*, pp. 17–26, Oct 2006, DOI: 10.1109/IISWC.2006.302726.

[575] C. Rolland, J. Ridoux, and B. Baynat, "*LiTGen, a lightweight traffic generator: Applications to P2P and mail wireless traffic*". In 8th *Passive & Active Measurement Conf.*, pp. 52–62, Springer-Verlag, Apr 2007, DOI: 10.1007/978-3-540-71617-4_6. Lect. Notes Comput. Sci. vol. 4427.

[576] C. Rolland, J. Ridoux, and B. Baynat, "*Catching IP traffic burstiness with a lightweight generator*". In 6th *IFIP Networking 2007*, pp. 924–934, Springer-Verlag, May 2007, DOI: 10.1007/978-3-540-72606-7_79. Lect. Notes Comput. Sci. vol. 4479.

[577] M. Rosenblum, S. A. Herrod, E. Witchel, and A. Gupta, "*Complete computer system simulation: The SimOS approach*". *IEEE Parallel & Distributed Technology* **3(4)**, pp. 34–43, Winter 1995, DOI: 10.1109/88.473612.

[578] M. Rosenblum and J. K. Ousterhout, "*The design and implementation of a log-structured file system*". *ACM Trans. Comput. Syst.* **10(1)**, pp. 26–52, Feb 1992, DOI: 10.1145/146941.146943.

[579] M. Rosenstein, "*What is actually taking place on web sites: E-commerce lessons from web server logs*". In *ACM Conf. Electronic Commerce*, pp. 38–43, Oct 2000, DOI: 10.1145/352871.352876.

[580] R. F. Rosin, "*Determining a computing center environment*". *Comm. ACM* **8(7)**, pp. 465–468, Jul 1965, DOI: 10.1145/364995.365690.

[581] S. M. Ross, *Introduction to Probability Models*. Academic Press, 5th ed., 1993.

[582] D. Rossi, M. Mellia, and C. Casetti, "*User patience and the web: A hands-on investigation*". In *IEEE Globecom*, vol. 7, pp. 4163–4168, Dec 2003, DOI: 10.1109/GLOCOM.2003.1259011.

[583] E. Rosti, G. Serazzi, E. Smirni, and M. S. Squillante, "*Models of parallel applications with large computation and I/O requirements*". *IEEE Trans. Softw. Eng.* **28(3)**, pp. 286–307, Mar 2002, DOI: 10.1109/32.991321.

[584] A. Roubos and O. Jouini, "*Call centers with hyperexponential patience modeling*". *Intl. J. Production Economics* **141(1)**, pp. 307–315, Jan 2013, DOI: 10.1016/j.ijpe.2012.08.011.

[585] R. V. Rubin, L. Rudolph, and D. Zernik, "*Debugging parallel programs in parallel*". In *Workshop on Parallel and Distributed Debugging*, pp. 216–225, SIGPLAN/SIGOPS, May 1988, DOI: 10.1145/68210.69236.

[586] C. Ruemmler and J. Wilkes, "*UNIX disk access patterns*". In *USENIX Winter Conf. Proc.*, pp. 405–420, Jan 1993.

[587] C. Ruemmler and J. Wilkes, "*An introduction to disk drive modeling*". *Computer* **27(3)**, pp. 17–28, Mar 1994, DOI: 10.1109/2.268881.

[588] R. J. Rummel, "*Understanding correlation*". URL http://www.mega.nu:8080/ampp/rummel/uc.htm, 1976.

[589] G. Samorodnitsky, "*Long range dependence*". *Foundations & Trends in Stochastic Systems* **1(3)**, pp. 163–257, 2007, DOI: 10.1561/0900000004.

[590] G. Samorodnitsky and M. S. Taqqu, *Stable Non-Gaussian Random Processes: Stochastic Models with Infinite Variance*. Chapman & Hall, 1994.

[591] N. Sarrar, S. Uhlig, A. Feldmann, R. Sherwood, and X. Huang, "*Leveraging Zipf's law for traffic offloading*". *Comput. Commun. Rev.* **42(1)**, pp. 17–22, Jan 2012, DOI: 10.1145/2096149.2096152.

[592] S. Sarvotham, R. Riedi, and R. Baraniuk, "*Connection-level analysis and modeling of network traffic*". In 1st *Internet Measurement Workshop*, pp. 99–103, Nov 2001, DOI: 10.1145/505202.505215.

[593] M. Satyanarayanan, "*The evolution of Coda*". *ACM Trans. Comput. Syst.* **20(2)**, pp. 85–124, May 2002, DOI: 10.1145/507052.507053.

[594] B. K. Schmidt, M. S. Lam, and J. D. Northcutt, "*The interactive performance of SLIM: A stateless thin-client architecture*". In 17th *Symp. Operating Systems Principles*, pp. 32–47, Dec 1999, DOI: 10.1145/319344.319154.

[595] F. Schneider, S. Agarwal, T. Alpcan, and A. Feldmann, "*The new web: Characterizing AJAX traffic*". In 9th *Passive & Active Measurement Conf.*, pp. 31–40, Apr 2008, DOI: 10.1007/978-3-540-79232-1_4. Lect. Notes Comput. Sci. vol. 4979.

[596] F. Schneider, B. Ager, G. Maier, A. Feldmann, and S. Uhlig, "*Pitfalls in HTTP traffic measurements and analysis*". In 13th *Passive & Active Measurement Conf.*, pp. 242–251, Springer-Verlag, Mar 2012, DOI: 10.1007/978-3-642-28537-0_24. Lect. Notes Comput. Sci. vol. 7192.

[597] F. Schneider, J. Wallerich, and A. Feldmann, "*Packet capture in 10-Gigabit Ethernet environments using contemporary commodity hardware*". In 8th *Passive & Active Measurement Conf.*, pp. 207–217, Springer-Verlag, Apr 2007, DOI: 10.1007/978-3-540-71617-4_21. Lect. Notes Comput. Sci. vol. 4427.

[598] B. Schroeder and G. A. Gibson, "*Disk failures in the real world: What does an MTTF of 1,000,000 hours mean to you?*" In 5th *USENIX Conf. File & Storage Technologies*, pp. 1–16, Feb 2007.

[599] B. Schroeder and M. Harchol-Balter, "*Evaluation of task assignment policies for supercomputing servers: The case for load unbalancing and fairness*". *Cluster Comput.* **7(2)**, pp. 151–161, Apr 2004, DOI: 10.1023/B:CLUS.0000018564.05723.a2.

[600] B. Schroeder and M. Harchol-Balter, "*Web servers under overload: How scheduling can help*". *ACM Trans. Internet Technology* **6(1)**, pp. 20–52, Feb 2006.

[601] B. Schroeder, A. Wierman, and M. Harchol-Balter, "*Open versus closed: A cautionary tale*". In 3rd *Networked Systems Design & Implementation*, pp. 239–252, May 2006.

[602] M. Schroeder, *Fractals, Chaos, Power Laws*. W. H. Freeman and Co., 1991.

[603] G. Schwarz, "*Estimating the dimension of a model*". *Ann. Statist.* **6(2)**, pp. 461–464, Mar 1978.

[604] D. W. Scott, "*On optimal and data-based histograms*". *Biometrika* **66(3)**, pp. 605–610, 1979, DOI: 10.1093/biomet/66.3.605.

[605] S. L. Scott and G. S. Sohi, "*The use of feedback in multiprocessors and its application to tree saturation control*". *IEEE Trans. Parallel & Distributed Syst.* **1(4)**, pp. 385–398, Oct 1990, DOI: 10.1109/71.80178.

[606] E. Selberg and O. Etzioni, "*The MetaCrawler architecture for resource aggregation on the web*". *IEEE Expert* **12(1)**, pp. 11–14, Jan-Feb 1997, DOI: 10.1109/64.577468.

[607] K. C. Sevcik, "*Characterization of parallelism in applications and their use in scheduling*". In *SIGMETRICS Conf. Measurement & Modeling of Comput. Syst.*, pp. 171–180, May 1989, DOI: 10.1145/75108.75391.

[608] K. C. Sevcik, "*Application scheduling and processor allocation in multiprogrammed parallel processing systems*". *Performance Evaluation* **19(2-3)**, pp. 107–140, Mar 1994, DOI: 10.1016/0166-5316(94)90036-1.

[609] K. C. Sevcik and X.-N. Tan, "*An interconnection network that exploits locality of communication*". In 8th *Intl. Conf. Distributed Comput. Syst.*, pp. 74–85, Jun 1988, DOI: 10.1109/DCS.1988.12502.

[610] A. Shaikh, J. Rexford, and K. G. Shin, "*Load-sensitive routing of long-lived IP flows*". In *ACM SIGCOMM Conf.*, pp. 215–226, Aug 1999, DOI: 10.1145/316194.316225.

[611] X. Shen, C. Zhang, C. Ding, M. L. Scott, S. Dwarkadas, and M. Ogihara, "*Analysis of input-dependent program behavior using active profiling*". In *Workshop on Experimental Computer Science*, art. 5, Jun 2007, DOI: 10.1145/1281700.1281705.

[612] S. Sherman, F. Baskett III, and J. C. Browne, "*Trace-driven modeling and analysis of CPU scheduling in a multiprogramming system*". *Comm. ACM* **15(12)**, pp. 1063–1069, Dec 1972, DOI: 10.1145/361598.361626.

[613] T. Sherwood, E. Perelman, G. Hamerly, and B. Calder, "*Automatically characterizing large scale program behavior*". In 10th *Intl. Conf. Architect. Support for Prog. Lang. & Operating Syst.*, pp. 45–57, Oct 2002, DOI: 10.1145/605397.605403.

[614] E. Shmueli and D. G. Feitelson, "*Using site-level modeling to evaluate the performance of parallel system schedulers*". In 14th *Modeling, Anal. & Simulation of Comput. & Telecomm. Syst.*, pp. 167–176, Sep 2006, DOI: 10.1109/MASCOTS.2006.50.

[615] E. Shmueli and D. G. Feitelson, "*Uncovering the effect of system performance on user behavior from traces of parallel systems*". In 15th *Modeling, Anal. & Simulation of Comput. & Telecomm. Syst.*, pp. 274–280, Oct 2007, DOI: 10.1109/MASCOTS.2007.67.

[616] E. Shmueli and D. G. Feitelson, "*On simulation and design of parallel-systems schedulers: Are we doing the right thing?*" *IEEE Trans. Parallel & Distributed Syst.* **20(7)**, pp. 983–996, Jul 2009, DOI: 10.1109/TPDS.2008.152.

[617] B. Shneiderman, *Designing the User Interface*. Addison Wesley, 3rd ed., 1998.

[618] E. Shriver and M. Hansen, *Search Session Extraction: A User Model of Searching*. Tech. rep., Bell Labs, Jan 2002.

[619] R. H. Shumway and D. S. Stoffer, *Time Series Analysis and Its Applications*. Springer, 3rd ed., 2011.

[620] F. N. Sibai, "*Performance analysis and workload characterization of the 3DMark05 benchmark on modern parallel computer platforms*". *Comput. Arch. News* **35(3)**, pp. 44–52, Jun 2007, DOI: 10.1145/1294313.1294315.

[621] H. S. Sichel, "*On a distribution law for word frequencies*". *J. Am. Stat. Assoc.* **70(351)**, pp. 542–547, Sep 1975, DOI: 10.1080/01621459.1975.10482469.

[622] B. Sikdar and K. S. Vastola, "*On the contribution of TCP to the self-similarity of network traffic*". In *Intl. Workshop Digital Commun.*, S. Palazzo (ed.), pp. 596–613, Springer-Verlag, Sep 2001, DOI: 10.1007/3-540-45400-4_38. Lect. Notes Comput. Sci. vol. 2170.

[623] C. Silverstein, M. Henzinger, H. Marais, and M. Moricz, "*Analysis of a very large web search engine query log*". *SIGIR Forum* **33(1)**, pp. 6–12, Fall 1999, DOI: 10.1145/331403.331405.

[624] H. Simitci and D. A. Reed, "*A comparison of logical and physical parallel I/O patterns*". *Intl. J. High Performance Comput. Appl.* **12(3)**, pp. 364–380, Sep 1998, DOI: 10.1177/109434209801200305.

[625] M. V. Simkin and V. P. Roychowdhury, "*A mathematical theory of citing*". *J. Am. Soc. Inf. Sci. & Tech.* **58(11)**, pp. 1661–1673, Sep 2007, DOI: 10.1002/asi.20653.

[626] M. V. Simkin and V. P. Roychowdhury, "*Re-inventing Willis*". *Phys. Rep.* **502(1)**, pp. 1–35, May 2011, DOI: 10.1016/j.physrep.2010.12.004.

[627] H. A. Simon, "*On a class of skew distribution functions*". *Biometrika* **42(3/4)**, pp. 425–440, Dec 1955, DOI: 10.1093/biomet/42.3-4.425.

[628] H. A. Simon, "*Some further notes on a class of skew distribution functions*". *Info. & Control* **3(1)**, pp. 80–88, Mar 1960, DOI: 10.1016/S0019-9958(60)90302-8.

[629] A. Singh and Z. Segall, "*Synthetic workload generation for experimentation with multiprocessors*". In 3rd *Intl. Conf. Distributed Comput. Syst.*, pp. 778–785, Oct 1982.

[630] J. P. Singh, J. L. Hennessy, and A. Gupta, "*Scaling parallel programs for multiprocessors: Methodology and examples*". *Computer* **26(7)**, pp. 42–50, Jul 1993, DOI: 10.1109/MC.1993.274941.

[631] J. P. Singh, H. S. Stone, and D. F. Thiebaut, "*A model of workloads and its use in miss-rate prediction for fully associative caches*". *IEEE Trans. Comput.* **41(7)**, pp. 811–825, Jul 1992, DOI: 10.1109/12.256450.

[632] J. P. Singh, W.-D. Weber, and A. Gupta, "*SPLASH: Stanford parallel applications for shared-memory*". *Comput. Arch. News* **20(1)**, pp. 5–44, Mar 1992, DOI: 10.1145/130823.130824.

[633] N. T. Slingerland and A. J. Smith, "*Design and characterization of the Berkeley multimedia workload*". *Multimedia Systems* **8(4)**, pp. 315–327, Jul 2002, DOI: 10.1007/s005300200052.

[634] E. Smirni and D. A. Reed, "*Workload characterization of input/output intensive parallel applications*". In 9th *Intl. Conf. Comput. Performance Evaluation*, pp. 169–180, Springer-Verlag, Jun 1997, DOI: 10.1007/BFb0022205. Lect. Notes Comput. Sci. vol. 1245.

[635] A. J. Smith, "*Cache memories*". *ACM Comput. Surv.* **14(3)**, pp. 473–530, Sep 1982, DOI: 10.1145/356887.356892.

[636] A. J. Smith, "*Workloads (creation and use)*". *Comm. ACM* **50(11)**, pp. 45–50, Nov 2007, DOI: 10.1145/1297797.1297821.

[637] J. E. Smith, "*A study of branch prediction strategies*". In 8th *Ann. Intl. Symp. Computer Architecture Conf. Proc.*, pp. 135–148, May 1981.

[638] J. E. Smith, "*Characterizing computer performance with a single number*". *Comm. ACM* **31(10)**, pp. 1202–1206, Oct 1988, DOI: 10.1145/63039.63043.

[639] K. A. Smith and M. I. Seltzer, "*File system aging—increasing the relevance of file system benchmarks*". In *SIGMETRICS Conf. Measurement & Modeling of Comput. Syst.*, pp. 203–213, Jun 1997, DOI: 10.1145/258612.258689.

[640] M. Snir, S. W. Otto, S. Huss-Lederman, D. W. Walker, and J. Dongarra, *MPI: The Complete Reference*. MIT Press, 1996.

[641] J. Sommers and P. Barford, "*Self-configuring network traffic generation*". In 4th *Internet Measurement Conf.*, pp. 68–81, Oct 2004, DOI: 10.1145/1028788.1028798.

[642] J. Sommers, V. Yegneswaran, and P. Barford, "*A framework for malicious workload generation*". In 4th *Internet Measurement Conf.*, pp. 82–87, Oct 2004, DOI: 10.1145/1028788.1028799.

[643] B. Song, C. Ernemann, and R. Yahyapour, "*Parallel computer workload modeling with Markov chains*". In *Job Scheduling Strategies for Parallel Processing*, pp. 47–62, Springer-Verlag, 2004, DOI: 10.1007/11407522_3. Lect. Notes Comput. Sci. vol. 3277.

[644] C. Spearman, "*The proof and measurement of association between two things*". *Am. J. Psychology* **15(1)**, pp. 72–101, Jan 1904.

[645] I. Spence, N. Kutlesa, and D. L. Rose, "*Using color to code quantity in spatial displays*". *J. Experimental Psychology: Applied* **5(4)**, pp. 393–412, 1999.

[646] A. Spink, B. J. Jansen, D. Wolfram, and T. Saracevic, "*From e-sex to e-commerce: Web search changes*". *Computer* **35(3)**, pp. 107–109, Mar 2002, DOI: 10.1109/2.989940.

[647] A. Spink, H. Partridge, and B. J. Jansen, "*Sexual and pornographic web searching: Trends analysis*". *First Monday* **11(9)**, Sep 2006. URL http://www.firstmonday.org/issues/issue11_9/spink/.

[648] J. R. Spirn, *Program Behavior: Models and Measurements*. Elsevier North Holland Inc., 1977.

[649] J. R. Spirn and P. J. Denning, "*Experiments with program locality*". In *AFIPS Fall Joint Comput. Conf.*, vol. I, pp. 611–621, Dec 1972, DOI: 10.1145/1479992.1480078.

[650] K. Springborn and P. Barford, "*Impression fraud in online advertising via pay-per-view networks*". In 22nd *USENIX Security Symp.*, pp. 211–226, Aug 2013.

[651] B. Sprunt, "*The basics of performance-monitoring hardware*". *IEEE Micro* **22(4)**, pp. 64–71, Jul-Aug 2002, DOI: 10.1109/MM.2002.1028477.

[652] B. Sprunt, "*Pentium 4 performance-monitoring features*". *IEEE Micro* **22(4)**, pp. 72–82, Jul-Aug 2002, DOI: 10.1109/MM.2002.1028478.

[653] M. S. Squillante, D. D. Yao, and L. Zhang, "*Analysis of job arrival patterns and parallel scheduling performance*". *Performance Evaluation* **36–37**, pp. 137–163, Aug 1999, DOI: 10.1016/S0166-5316(99)00035-8.

[654] K. Sreenivasan and A. J. Kleinman, "*On the construction of a representative synthetic workload*". *Comm. ACM* **17(3)**, pp. 127–133, Mar 1974, DOI: 10.1145/360860.360863.

[655] "*Standard performance evaluation corporation*". URL http://www.spec.org.

[656] Standard Performance Evaluation Corp., "*SPECweb2005*". URL http://www.spec.org/web2005/, 2005.

[657] D. Starobinski and D. Tse, "*Probabilistic methods in web caching*". *Performance Evaluation* **46(2-3)**, pp. 125–137, Oct 2001, DOI: 10.1016/S0166-5316(01)00045-1.

[658] A. Stassopoulou and M. D. Dikaiakos, "*Web robot detection: A probabilistic reasoning approach*". *Computer Networks* **53(3)**, pp. 265–278, Feb 2009, DOI: 10.1016/j.comnet.2008.09.021.

[659] M. A. Stephens, "*EDF statistics for goodness of fit and some comparisons*". *J. Am. Stat. Assoc.* **69(347)**, pp. 730–737, Sep 1974, DOI: 10.1080/01621459.1974.10480196.

[660] M. A. Stephens, "*Tests based on EDF statistics*". In *Goodness-of-Fit Techniques*, R. B. D'Agostino and M. A. Stephens (eds.), pp. 97–193, Marcel Dekker, Inc., 1986.

[661] R. Stets, K. Gharachorloo, and L. A. Barroso, "*A detailed comparison of two transaction processing workloads*". In 5th *Workshop on Workload Characterization*, pp. 37–48, Nov 2002, DOI: 10.1109/WWC.2002.1226492.

[662] C. Stewart, T. Kelly, and A. Zhang, "*Exploiting nonstationarity for performance prediction*". In 2nd *EuroSys*, pp. 31–44, Mar 2007, DOI: 10.1145/1272996.1273002.

[663] S. Stoev, M. S. Taqqu, C. Park, and J. S. Marron, "*On the wavelet spectrum diagnostic for Hurst parameter estimation in the analysis of Internet traffic*". *Comput. Networks* **48(3)**, pp. 423–445, Jun 2005, DOI: 10.1016/j.comnet.2004.11.017.

[664] P. Stoica and R. Moses, *Spectral Analysis of Signals*. Prentice-Hall, 2005.

[665] S. J. Stolfo, S. Hershkop, C.-W. Hu, W.-J. Li, O. Nimeskern, and K. Wang, "*Behavior-based modeling and its application to email analysis*". *ACM Trans. Internet Technology* **6(2)**, pp. 187–221, May 2006, DOI: 10.1145/1149121.1149125.

[666] B. Stone, "*Spam back to 94% of all e-mail*". New York Times (Technology Section), 31 Mar 2009.

[667] R. A. Sugumar and S. G. Abraham, "*Set-associative cache simulation using generalized binomial trees*". *ACM Trans. Comput. Syst.* **13(1)**, pp. 32–56, Feb 1995, DOI: 10.1145/200912.200918.

[668] M. Sutter and M. G. Kocher, "*Power laws of research output: Evidence for journals of economics*". *Scientometrics* **51(2)**, pp. 405–414, Jun 2001, DOI: 10.1023/A:1012757802706.

[669] D. Talby and D. G. Feitelson, "*Improving and stabilizing parallel computer performance using adaptive backfilling*". In 19th *Intl. Parallel & Distributed Processing Symp.*, Apr 2005, DOI: 10.1109/IPDPS.2005.252.

[670] D. Talby, D. G. Feitelson, and A. Raveh, "*Comparing logs and models of parallel workloads using the Co-plot method*". In *Job Scheduling Strategies for Parallel Processing*, pp. 43–66, Springer-Verlag, 1999, DOI: 10.1007/3-540-47954-6_3. Lect. Notes Comput. Sci. vol. 1659.

[671] D. Talby, D. G. Feitelson, and A. Raveh, "*A co-plot analysis of logs and models of parallel workloads*". *ACM Trans. Modeling & Comput. Simulation* **12(3)**, art. 12, Jul 2007, DOI: 10.1145/1243991.1243993.

[672] N. N. Taleb, *Fooled by Randomness*. Texere, 2004.

[673] D. Talla, L. K. John, and D. Berger, "*Bottlenecks in multimedia processing with SIMD style extensions and architectural enhancements*". *IEEE Trans. Comput.* **52(8)**, pp. 1015–1031, Aug 2003, DOI: 10.1109/TC.2003.1223637.

[674] A. S. Tanenbaum, J. N. Herder, and H. Bos, "*File size distribution on UNIX systems—then and now*". *Operating Syst. Rev.* **40(1)**, pp. 100–104, Jan 2006, DOI: 10.1145/1113361.1113364.

[675] W. Tang, Y. Fu, L. Cherkasova, and A. Vahdat, "*Modeling and generating realistic streaming media server workloads*". *Comput. Networks* **51(1)**, pp. 336–356, Jan 2007, DOI: 10.1016/j.comnet.2006.05.003.

[676] M. S. Taqqu and V. Teverovsky, "*Robustness of Whittle-type estimators for time series with long-range dependence*". *Stochastic Models* **13(4)**, pp. 723–757, 1997, DOI: 10.1080/15326349708807449.

[677] M. S. Taqqu and V. Teverovsky, "*On estimating the intensity of long-range dependence in finite and infinite variance time series*". In *A Practical Guide to Heavy Tails*, R. J. Adler, R. E. Feldman, and M. S. Taqqu (eds.), pp. 177–217, Birkhäuser, 1998.

[678] M. S. Taqqu, V. Teverovsky, and W. Willinger, "*Is network traffic self-similar or multifractal?*" *Fractals* **5(1)**, pp. 63–73, Mar 1997, DOI: 10.1142/S0218348X97000073.

[679] M. S. Taqqu, W. Willinger, and R. Sherman, "*Proof of a fundamental result in self-similar traffic modeling*". *Comput. Commun. Rev.* **27(2)**, pp. 5–23, Apr 1997, DOI: 10.1145/263876.263879.

[680] R. P. Taylor, "*Order in Pollock's chaos*". *Scientific American* **287(6)**, pp. 84–89, Dec 2002.

[681] J. Teevan, E. Adar, R. Jones, and M. A. S. Potts, "*Information re-retrieval: Repeat queries in Yahoo's logs*". In 30th *SIGIR Conf. Information Retrieval*, pp. 151–158, Jul 2007, DOI: 10.1145/1277741.1277770.

[682] V. Teverovsky, M. S. Taqqu, and W. Willinger, "*A critical look at Lo's modified R/S statistic*". *J. Statistical Planning & Inference* **80(1-2)**, pp. 211–227, Aug 1999, DOI: 10.1016/S0378-3758(98)00250-X.

[683] D. Thiébaut, "*On the fractal dimension of computer programs and its application to the prediction of the cache miss ratio*". *IEEE Trans. Comput.* **38(7)**, pp. 1012–1026, Jul 1989, DOI: 10.1109/12.30852.

[684] D. Thiébaut, J. L. Wolf, and H. S. Stone, "*Synthetic traces for trace-driven simulation of cache memories*". *IEEE Trans. Comput.* **41(4)**, pp. 388–410, Apr 1992, DOI: 10.1109/12.135552. (Corrected in *IEEE Trans. Comput.* **42(5)** p. 635, May 1993).

[685] A. Thusoo, J. Sen Sarma, N. Jain, Z. Shao, P. Chakka, S. Anthony, H. Liu, P. Wyckoff, and R. Murthy, "*Hive – a warehousing solution over a map-reduce framework*". *Proc. VLDB Endowment* **2(2)**, pp. 1626–1629, Aug 2009.

[686] G. L. Tietjen, "*The analysis and detection of outliers*". In *Goodness-of-Fit Techniques*, R. B. D'Agostino and M. A. Stephens (eds.), pp. 497–522, Marcel Dekker, Inc., 1986.

[687] A. Traeger, E. Zadok, N. Joukov, and C. P. Wright, "*A nine year study of file system and storage benchmarking*". *ACM Trans. Storage* **4(2)**, pp. 5:1–5:56, May 2008, DOI: 10.1145/1367829.1367831.

[688] D. N. Tran, W. T. Ooi, and Y. C. Tay, "*SAX: A tool for studying congestion-induced surfer behavior*". In 7th *Passive & Active Measurement Conf.*, Mar 2006.

[689] Transaction Processing Performance Council (TPC), "*TPC benchmark DS standard specification version 1.1.0*". URL http://www.tpc.org/tpcds/spec/tpcds_1.1.0.pdf, Apr 2012.

[690] O. Tripp and D. G. Feitelson, *Zipf's Law Revisited*. Tech. Rep. 2007-115, School of Computer Science and Engineering, The Hebrew University of Jerusalem, Aug 2007.

[691] K. S. Trivedi, *Probability and Statistics with Reliability, Queueing, and Computer Science Applications*. John Wiley & Sons, 2nd ed., 2002.

[692] K. S. Trivedi and K. Vaidyanathan, "*Software reliability and rejuvenation: Modeling and analysis*". In *Performance Evaluation of Complex Systems: Techniques and Tools*, M. C. Calzarossa and S. Tucci (eds.), pp. 318–345, Springer-Verlag, 2002, DOI: 10.1007/3-540-45798-4_14. Lect. Notes Comput. Sci. vol. 2459.

[693] D. Tsafrir, "*Using inaccurate estimates accurately*". In *Job Scheduling Strategies for Parallel Processing*, E. Frachtenberg and U. Schwiegelshohn (eds.), pp. 208–221, Springer, 2010, DOI: 10.1007/978-3-642-16505-4_12. Lect. Notes Comput. Sci. vol. 6253.

[694] D. Tsafrir, Y. Etsion, and D. G. Feitelson, "*Modeling user runtime estimates*". In *Job Scheduling Strategies for Parallel Processing*, pp. 1–35, Springer-Verlag, 2005, DOI: 10.1007/11605300_1. Lect. Notes Comput. Sci. vol. 3834.

[695] D. Tsafrir, Y. Etsion, and D. G. Feitelson, "*Backfilling using system-generated predictions rather than user runtime estimates*". *IEEE Trans. Parallel & Distributed Syst.* **18(6)**, pp. 789–803, Jun 2007, DOI: 10.1109/TPDS.2007.70606.

[696] D. Tsafrir and D. G. Feitelson, "*Instability in parallel job scheduling simulation: The role of workload flurries*". In 20th *Intl. Parallel & Distributed Processing Symp.*, Apr 2006, DOI: 10.1109/IPDPS.2006.1639311.

[697] D. Tsafrir and D. G. Feitelson, "*The dynamics of backfilling: Solving the mystery of why increased inaccuracy may help*". In *IEEE Intl. Symp. Workload Characterization*, pp. 131–141, Oct 2006, DOI: 10.1109/IISWC.2006.302737.

[698] J. J. P. Tsai, K.-Y. Fang, and H.-Y. Chen, "*A noninvasive architecture to monitor real-time distributed systems*". *Computer* **23(3)**, pp. 11–23, Mar 1990, DOI: 10.1109/2.50269.

[699] E. R. Tufte, *The Visual Display of Quantitative Information*. Graphics Press, 1983.

[700] E. R. Tufte, *Envisioning Information*. Graphics Press, 1990.

[701] E. R. Tufte, *Visual Explanations*. Graphics Press, 1997.

[702] J. W. Tukey, *Exploratory Data Analysis*. Addison-Wesley, 1977.

[703] J. Turek, W. Ludwig, J. L. Wolf, L. Fleischer, P. Tiwari, J. Glasgow, U. Schwiegelshohn, and P. S. Yu, "*Scheduling parallelizable tasks to minimize average response time*". In 6th *Symp. Parallel Algorithms & Architectures*, pp. 200–209, Jun 1994, DOI: 10.1145/181014.181331.

[704] S. K. Tyler and J. Teevan, "*Large scale query log analysis of re-finding*". In 3rd *Web Search & Data Mining*, pp. 191–200, Feb 2010, DOI: 10.1145/1718487.1718512.

[705] R. A. Uhlig and T. N. Mudge, "*Trace-driven memory simulation: A survey*". *ACM Comput. Surv.* **29(2)**, pp. 128–170, Jun 1997, DOI: 10.1145/254180.254184.

[706] L. G. Valiant, "*A bridging model for parallel computation*". *Comm. ACM* **33(8)**, pp. 103–111, Aug 1990.

[707] D. Veitch, N. Hohn, and P. Abry, "*Multifractality in TCP/IP traffic: The case against*". *Comput. Networks* **48(3)**, pp. 293–313, Jun 2005, DOI: 10.1016/j.comnet.2004.11.011.

[708] J. S. Vetter and F. Mueller, "*Communication characteristics of large-scale scientific applications for contemporary cluster architectures*". *J. Parallel & Distributed Comput.* **63(9)**, pp. 853–865, Sep 2003, DOI: 10.1016/S0743-7315(03)00104-7.

[709] K. Vieira, A. Schulter, C. B. Westphall, and C. M. Westphall, "*Intrusion detection for grid and cloud computing*". *ITProfessional* **12(4)**, pp. 38–43, Jul-Aug 2010, DOI: 10.1109/MITP.2009.89.

[710] K. V. Vishwanath and A. Vahdat, "*Swing: Realistic and responsive network traffic generation*". *IEEE/ACM Trans. Networking* **17(3)**, pp. 712–725, Jun 2009, DOI: 10.1109/TNET.2009.2020830.

[711] W. Vogels, "*File system usage in Windows NT 4.0*". In 17th *Symp. Operating Systems Principles*, pp. 93–109, Dec 1999, DOI: 10.1145/319344.319158.

[712] J. Voldman, B. Mandelbrot, L. W. Hoevel, J. Knight, and P. Rosenfeld, "*Fractal nature of software-cache interaction*". *IBM J. Res. Dev.* **27(2)**, pp. 164–170, Mar 1983, DOI: 10.1147/rd.272.0164.

[713] E. M. Voorhees, "*TREC: Improving information access through evaluation*". *Bulletin Am. Soc. Information science & Tech.* **32(1)**, pp. 16–21, Oct/Nov 2005, DOI: 10.1002/bult.2003.1720320105.

[714] E. M. Voorhees, "*TREC: Continuing information retrieval's tradition of experimentation*". *Comm. ACM* **50(11)**, pp. 51–54, Nov 2007, DOI: 10.1145/1297797.1297822.

[715] E. M. Voorhees, "*On test collections for adaptive information retrieval*". *Inf. Process. & Management* **44(6)**, pp. 1879–1885, Nov 2008, DOI: 10.1016/j.ipm.2007.12.011.

[716] E. M. Voorhees and D. K. Harman (eds.), *TREC: Experiment and Evaluation in Information Retrieval*. MIT Press, 2005.

[717] E. S. Walter and V. L. Wallace, "*Further analysis of a computing center environment*". *Comm. ACM* **10(5)**, pp. 266–272, May 1967, DOI: 10.1145/363282.363294.

[718] M. Wan, R. Moore, G. Kremenek, and K. Steube, "*A batch scheduler for the Intel Paragon with a non-contiguous node allocation algorithm*". In *Job Scheduling Strategies for Parallel Processing*, pp. 48–64, Springer-Verlag, 1996, DOI: 10.1007/BFb0022287. Lect. Notes Comput. Sci. vol. 1162.

[719] G. Wang, A. R. Butt, P. Pandey, and K. Gupta, "*A simulation approach to evaluating design decisions in MapReduce setups*". In 17th *Modeling, Anal. & Simulation of Comput. & Telecomm. Syst.*, Sep 2009, DOI: 10.1109/MASCOT.2009.5366973.

[720] M. Wang, A. Ailamaki, and C. Faloutsos, "*Capturing the spatio-temporal behavior of real traffic data*". *Performance Evaluation* **49(1-4)**, pp. 147–163, Aug 2002, DOI: 10.1016/S0166-5316(02)00108-6.

[721] C. Ware, *Information Visualization*. Morgan Kaufmann, 2nd ed., 2004.

[722] P. P. Ware, T. W. Page, Jr., and B. L. Nelson, "*Automatic modeling of file system workloads using two-level arrival processes*". *ACM Trans. Modeling & Comput. Simulation* **8(3)**, pp. 305–330, Jul 1998, DOI: 10.1145/290274.290317.

[723] R. Weicker, "*Benchmarking*". In *Performance Evaluation of Complex Systems: Techniques and Tools*, M. C. Calzarossa and S. Tucci (eds.), pp. 179–207, Springer-Verlag, 2002, DOI: 10.1007/3-540-45798-4_9. Lect. Notes Comput. Sci. vol. 2459.

[724] R. P. Weicker, "*An overview of common benchmarks*". *Computer* **23(12)**, pp. 65–75, Dec 1990, DOI: 10.1109/2.62094.

[725] M. C. Weigle, P. Adurthi, F. Hernández-Campos, K. Jeffay, and F. D. Smith, "*Tmix: A tool for generating realistic TCP application workloads in ns-2*". *Comput. Commun. Rev.* **36(3)**, pp. 67–76, Jul 2006, DOI: 10.1145/1140086.1140094.

[726] H. Weinreich, H. Obendorf, and E. Herder, "*Data cleaning methods for client and proxy logs*". In *Workshop on Logging Traces of Web Activity: The Mechanics of Data Collection*, May 2006.

[727] H. Weinreich, H. Obendorf, E. Herder, and M. Mayer, "*Not quite the average: An empirical study of web use*". *ACM Trans. Web* **2(1)**, art. 5, Feb 2008, DOI: 10.1145/1326561.1326566.

[728] B. P. Welford, "*Note on a method for calculating corrected sums of squares and products*". *Technometrics* **4(3)**, pp. 419–420, Aug 1962, DOI: 10.1080/00401706.1962.10490022.

[729] A. Williams, M. Arlitt, C. Williamson, and K. Barker, "*Web workload characterization: Ten years later*". In *Web Content Delivery*, X. Tang, J. Xu, and S. T. Chanson (eds.), chap. 1, pp. 3–21, Springer Science+Business Media, Inc., 2005, DOI: 10.1007/0-387-27727-7_1.

[730] S. Williams, A. Waterman, and D. Patterson, "*Roofline: An insightful visual performance model for multicore architectures*". *Comm. ACM* **52(4)**, pp. 65–76, Apr 2009, DOI: 10.1145/1498765.1498785.

[731] W. Willinger, V. Paxson, and M. S. Taqqu, "*Self-similarity and heavy tails: Structural modeling of network traffic*". In *A Practical Guide to Heavy Tails*, R. J. Adler, R. E. Feldman, and M. S. Taqqu (eds.), pp. 27–53, Birkhäuser, 1998.

[732] W. Willinger, M. S. Taqqu, R. Sherman, and D. V. Wilson, "*Self-similarity through high variability: Statistical analysis of Ethernet LAN traffic at the source level*". *IEEE/ACM Trans. Networking* **5(1)**, pp. 71–86, Feb 1997, DOI: 10.1109/90.554723.

[733] R. R. Willis, "*A man/machine workload model*". *ACM SIGSIM Simulation Digest* **8(1)**, pp. 45–52, Oct 1976, DOI: 10.1145/1103189.1103203.

[734] P. R. Wilson, M. S. Johnstone, M. Neely, and D. Boles, "*Dynamic storage allocation: A survey and critical review*". In *Intl. Workshop Memory Management*, Springer-Verlag, Sep 1995, DOI: 10.1007/3-540-60368-9_19. Lect. Notes Comput. Sci. vol. 986.

[735] K. Windisch, V. Lo, R. Moore, D. Feitelson, and B. Nitzberg, "*A comparison of workload traces from two production parallel machines*". In 6th *Symp. Frontiers Massively Parallel Comput.*, pp. 319–326, Oct 1996, DOI: 10.1109/FMPC.1996.558107.

[736] N. Wisitpongphan and J. M. Peha, "*Effect of TCP on self-similarity of network traffic*". In 12th *IEEE Intl. Conf. Computer Communications & Networks*, pp. 370–373, Oct 2003, DOI: 10.1109/ICCCN.2003.1284196.

[737] J. L. Wolf, P. S. Yu, J. Turek, and D. M. Dias, "*A parallel hash join algorithm for managing data skew*". *IEEE Trans. Parallel & Distributed Syst.* **4(12)**, pp. 1355–1371, Dec 1993, DOI: 10.1109/71.250117.

[738] T. Wolf, R. Ramaswamy, S. Bunga, and N. Yang, "*An architecture for distributed real-time passive network measurement*". In 14th *Modeling, Anal. & Simulation of Comput. & Telecomm. Syst.*, pp. 335–344, Sep 2006, DOI: 10.1109/MASCOTS.2006.11.

[739] R. Wolski, "*Dynamically forecasting network performance using the network weather service*". *Cluster Comput.* **1(1)**, pp. 119–132, 1998, DOI: 10.1023/A:1019025230054.

[740] A. Wong, L. Oliker, W. Kramer, T. Kaltz, and D. Bailey, "*System utilization benchmark on the Cray T3E and IBM SP2*". In *Job Scheduling Strategies for Par-*

allel Processing, pp. 56–67, Springer-Verlag, 2000, DOI: 10.1007/3-540-39997-6_4. Lect. Notes Comput. Sci. vol. 1911.

[741] S. C. Woo, M. Ohara, E. Torrie, J. P. Singh, and A. Gupta, "*The SPLASH-2 programs: Characterization and methodological considerations*". In 22nd *Ann. Intl. Symp. Computer Architecture Conf. Proc.*, pp. 24–36, Jun 1995, DOI: 10.1145/223982.223990.

[742] P. H. Worley, "*The effect of time constraints on scaled speedup*". *SIAM J. Sci. Statist. Comput.* **11(5)**, pp. 838–858, Sep 1990, DOI: 10.1137/0911050.

[743] Y. Yan, X. Zhang, and Z. Zhang, "*Cacheminer: A runtime approach to exploit cache locality on SMP*". *IEEE Trans. Parallel & Distributed Syst.* **11(4)**, pp. 357–374, Apr 2000, DOI: 10.1109/71.850833.

[744] S. Yang and G. de Veciana, "*Bandwidth sharing: The role of user impatience*". In *IEEE Globecom*, vol. 4, pp. 2258–2262, Nov 2001, DOI: 10.1109/GLOCOM.2001.966181.

[745] J. J. Yi and D. J. Lilja, "*Simulation of computer architectures: Simulators, benchmarks, methodologies, and recommendations*". *IEEE Trans. Comput.* **55(3)**, pp. 268–280, Mar 2006, DOI: 10.1109/TC.2006.44.

[746] J. J. Yi, R. Sendag, L. Eeckhout, A. Joshi, D. J. Lilja, and L. K. John, "*Evaluating benchmark subsetting approaches*". In *IEEE Intl. Symp. Workload Characterization*, pp. 93–104, Oct 2006, DOI: 10.1109/IISWC.2006.302733.

[747] H. Yu, D. Zheng, B. Y. Zhao, and W. Zheng, "*Understanding user behavior in large-scale video-on-demand systems*". In *EuroSys*, pp. 333–344, Apr 2006, DOI: 10.1145/1217935.1217968.

[748] G. U. Yule, "*A mathematical theory of evolution, based on the conclusions of Dr. J. C. Willis, F.R.S.*" *Phil. Trans. Royal Soc. London Series B* **213(402-410)**, pp. 21–87, Jan 1925, DOI: 10.1098/rstb.1925.0002.

[749] N. Zakay and D. G. Feitelson, "*On identifying user session boundaries in parallel workload logs*". In *Job Scheduling Strategies for Parallel Processing*, W. Cirne et al. (eds.), pp. 216–234, Springer-Verlag, 2012, DOI: 10.1007/978-3-642-35867-8_12. Lect. Notes Comput. Sci. vol. 7698.

[750] N. Zakay and D. G. Feitelson, "*Workload resampling for performance evaluation of parallel job schedulers*". In 4th *Intl. Conf. Performance Engineering*, pp. 149–159, Apr 2013, DOI: 10.1145/2479871.2479893.

[751] N. Zakay and D. G. Feitelson, "*Workload resampling for performance evaluation of parallel job schedulers*". *Concurrency & Computation — Pract. & Exp.* **26(12)**, pp. 2079–2105, Aug 2014, DOI: 10.1002/cpe.3240.

[752] N. Zakay and D. G. Feitelson, "*Preserving user behavior characteristics in trace-based simulation of parallel job scheduling*". In 22nd *Modeling, Anal. & Simulation of Comput. & Telecomm. Syst.*, pp. 51–60, Sep 2014, DOI: 10.1109/MASCOTS.2014.15.

[753] D. Zaragoza and C. Belo, "*Experimental validation of the ON–OFF packet-level model for IP traffic*". *Comput. Commun.* **30(5)**, pp. 975–989, Mar 2007, DOI: 10.1016/j.comcom.2006.08.029.

[754] H. F. Zhang, Y. T. Shu, and O. Yang, "*Estimation of Hurst parameter by variance-time plots*". In *IEEE Pacific Rim Conf. Comm., Comput. & Signal Proc.*, vol. 2, pp. 883–886, Aug 1997, DOI: 10.1109/PACRIM.1997.620401.

[755] J. Zhang, A. Sivasubramaniam, H. Franke, N. Gautam, Y. Zhang, and S. Nagar, "*Synthesizing representative I/O workloads for TPC-H*". In 10th *Intl. Symp. High Performance Comput. Arch.*, pp. 142–151, 2004, DOI: 10.1109/HPCA.2004.10019.

[756] L. Zhang, Z. Liu, A. Riabov, M. Schulman, C. Xia, and F. Zhang, "*A comprehensive toolset for workload characterization, performance modeling, and online control*". In *Computer Performance Evaluations, Modelling Techniques and Tools*, P. Kemper and W. H. Sanders (eds.), pp. 63–77, Springer-Verlag, Sep 2003, DOI: 10.1007/978-3-540-45232-4_5. Lect. Notes Comput. Sci. vol. 2794.

[757] Q. Zhang, A. Riska, W. Sun, E. Smirni, and G. Ciardo, "*Workload-aware load balancing for clustered web servers*". *IEEE Trans. Parallel & Distributed Syst.* **16(3)**, pp. 219–233, Mar 2005, DOI: 10.1109/TPDS.2005.38.

[758] Y. Zhang and A. Moffat, "*Some observations on user search behavior*". In 11th *Australasian Document Computing Symp.*, Dec 2006.

[759] Y. Zhang, J. Yang, and R. Gupta, "*Frequent value locality and value-centric data cache design*". In 9th *Intl. Conf. Architect. Support for Prog. Lang. & Operating Syst.*, pp. 150–159, Nov 2000, DOI: 10.1145/356989.357003.

[760] Y. Zhang, J. Zhang, A. Sivasubramaniam, C. Liu, and H. Franke, "*Decision-support workload characteristics on a clustered database server from the OS perspective*". In 23rd *Intl. Conf. Distributed Comput. Syst.*, pp. 386–393, May 2003, DOI: 10.1109/ICDCS.2003.1203488.

[761] M. Zhou and A. J. Smith, "*Analysis of personal computer workloads*". In 7th *Modeling, Anal. & Simulation of Comput. & Telecomm. Syst.*, pp. 208–217, Oct 1999, DOI: 10.1109/MASCOT.1999.805057.

[762] S. Zhuravlev, S. Blagodurov, and A. Fedorova, "*Addressing shared resource contention in multicore processors via scheduling*". In 15th *Intl. Conf. Architect. Support for Prog. Lang. & Operating Syst.*, pp. 129–142, Mar 2010, DOI: 10.1145/1736020.1736036.

[763] J. Zilber, O. Amit, and D. Talby, "*What is worth learning from parallel workloads? A user and session based analysis*". In 19th *Intl. Conf. Supercomputing*, pp. 377–386, Jun 2005, DOI: 10.1145/1088149.1088200.

[764] G. K. Zipf, *Human Behavior and the Principle of Least Effort*. Addison-Wesley, 1949.

[765] D. Zotkin and P. J. Keleher, "*Job-length estimation and performance in backfilling schedulers*". In 8th *Intl. Symp. High Performance Distributed Comput.*, pp. 236–243, Aug 1999, DOI: 10.1109/HPDC.1999.805303.

[766] A. Zuck, S. Toledo, D. Sotnikov, and D. Harnik, "*Compression and SSDs: Where and how?*" In 2nd *Workshop Interactions of NVM/Flash with Operating Systems & Workloads*, Oct 2014.

[767] A. Zygmund, *Trigonometric Series*. Cambridge University Press, 2nd ed., 1959. (Two volumes).

Index