

Performance Evaluation

Operating system courses are typically concerned with *how to do* various things, such as processor scheduling, memory allocation, etc. Naturally, this should be coupled with a discussion of *how well* the different approaches work.

2.1 Performance Metrics

Interestingly, the answer to how well a system works may depend on how you quantify “wellness”, that is, on your metric for good performance.

Being fast is good

The most common metrics are related to time. If something takes less time, this is good for two reasons. From the user’s or client’s perspective, being done sooner means that you don’t have to wait as long. From the system’s perspective, being done sooner means that we are now free to deal with other things.

While being fast is good, it still leaves the issue of units. Should the task of listing a directory, which takes milliseconds, be on the same scale as the task of factoring large numbers, which can take years? Thus it is sometimes better to consider time relative to some yardstick, and use different yardsticks for different jobs.

Being productive is good

A system’s client is typically only concerned with his own work, and wants it completed as fast as possible. But from the system’s perspective the sum of all clients is typically more important than any particular one of them. Making many clients moderately happy may therefore be better than making one client very happy, at the expense of others.

The metric of “happy clients per second” is called throughput. Formally, this is the number of jobs done in a unit of time. It should be noted that these can be various

types of jobs, e.g. applications being run, files being transferred, bank accounts being updated, etc.

Exercise 13 *Are response time and throughput simply two facets of the same thing?*

Being busy is good

Finally, another common metric is utilization. This metric is especially common when discussing systems, as it represents the system owner's point of view: if the system is being utilized, we are getting our money's worth. However, as we'll see below, there is often a tradeoff between utilization and responsiveness. If the system is driven to very high utilization, the average response time (the time from when a request arrives until it is done) rises precipitously.

Being up is good

The above metrics are concerned with the amount of useful work that gets done. There is also a whole group of metrics related to getting work done at all. These are metrics that measure system availability and reliability. For example, we can talk about the mean time between failures (MTBF), or the fraction of time that the system is down.

A special case in this class of metrics is the supportable load. Every system becomes overloaded if it is subjected to extreme load conditions. The question is, first, at what level of load this happens, and second, the degree to which the system can function under these conditions.

Keeping out of the way is good

A major goal of the operating system is to run as little as possible, and enable user jobs to use the computer's resources. Therefore, when the operating system does run, it should do so as quickly and unobtrusively as possible. In other words, the operating system's *overhead* should be low. In addition, it should not hang or cause other inconveniences to the users.

Exercise 14 *Which of the following metrics is related to time, throughput, utilization, or reliability?*

- *The probability that a workstation is idle and can therefore execute remote jobs*
- *The response time of a computer program*
- *The probability that a disk fails*
- *The bandwidth of a communications network*
- *The latency of loading a web page*
- *The number of transactions per second processed by a database system*

2.2 Workload Considerations

Workload = sequence of things to do

The workload on a system is the stream of jobs submitted to the system. From the system's point of view, the important attributes of the workload are when each job arrives, and what resources it needs. Similar considerations apply to subsystems. For example, for the file system the workload is the sequence of I/O operations, and the most important attribute is the number of bytes accessed.

Exercise 15 *What characterizes the workload on a memory subsystem?*

For reactive systems, the workload is the input

In algorithm design, we know that sometimes there is a significant gap between the worst-case behavior and the average behavior. For example, quick-sort has a worst-case cost of $O(n^2)$, but on average it is $O(n \log n)$ with a rather low constant, making it better than other algorithms that have a worst-case cost of $O(n \log n)$. The observed behavior depends on the input in each instance.

Similarly, the performance of a system depends not only on the system design and implementation but also on the workload to which it is subjected. We will see that in some cases one design is better for one workload, while a different design is better for another workload.

Performance evaluation must therefore be done subject to the results of a detailed workload analysis. Usually this is based on past workload measurements, which are used to create a workload model. Alternatively, the recorded workload from the past can be used directly to drive the evaluation of a new design.

Exercise 16 *You are designing a kiosk that allows people to check their email. You are told to expect some 300 users a day, each requiring about 2 minutes. Based on this you decide to deploy two terminals (there are 600 minutes in 10 hours, so two terminals provide about double the needed capacity — a safe margin). Now you are told that the system will be deployed in a school, for use during recess. Should you change your design?*

There are few benchmarks

Computer architecture design also depends on detailed analysis that is based on the workload. In that case, the workload used is a canonized set of applications that are recognized as being representative of general workloads; such selected applications are called benchmarks. A well-known example is the SPEC benchmarks, which are updated every few years [17].

For operating systems there are few such agreed benchmarks. In addition, creating a representative workload is more difficult because it has a *temporal* component.

This means that in a typical system additional work arrives unexpectedly at different times, and there are also times at which the system is idle simply because it has nothing to do. Thus an important component of modeling the workload is modeling the arrival process (as in the bursty example in Exercise 16). By contradistinction, in computer architecture studies all the work is available at the outset when the application is loaded, and the CPU just has to execute all of it as fast as possible.

2.2.1 Statistical Characterization of Workloads

The workload is randomly sampled from a distribution

One view of the workload on a system is that there exists a population of possible jobs to do, and every item is sampled at random from this population. The population is characterized by one or more distributions. For example, an important attribute of jobs is their runtime, and we can consider jobs as being sampled from a distribution of possible runtimes. By characterizing the distribution (e.g. the distribution of runtimes), we characterize the workload.

In many cases, the distributions turn out to be “a lot of low values and a few high values”. Moreover, the high values are sometimes *VERY HIGH*, and there are enough of them to make a difference. For example, consider the distribution of file sizes. Most files are very small, no more than a few dozen bytes. But some files are extremely big, spanning several gigabytes. While the probability of having a very large file is small, e.g. 0.0001, it is not negligible: in a system with 10,000 files we will probably have at least one such file. And because of its size, its disk space consumption is equivalent to that of a very large number of the smaller files.

Technically speaking, such distributions are said to possess a “fat tail”. Examples include the Pareto distribution and the lognormal distribution. If you have not heard of them, it is probably due to the regrettable tendency of introductory probability and statistics courses to focus on distributions that have simple mathematical properties, rather than on distributions which are known to provide a good representation of real experimental data.

Details: fat-tailed distributions

Fat tailed distributions are somewhat counter-intuitive, so we will start with some examples. Note that this discussion relates to distributions on positive numbers, that only have a right tail; negative numbers are typically meaningless in our applications (a file cannot have a negative size, and a job cannot have a negative runtime).

The distribution of the tail is fatter than the distribution as a whole

Consider the distribution of job runtimes, and ask the following question: given that a job has already run for time t , how much longer do you expect it to run?

The answer depends on the distribution of runtimes. A trivial case is when all jobs run for exactly T seconds. In this case, a job that has already run for t seconds has another $T - t$ seconds to run. In other words, the longer you have waited already, and less additional time you expect to have to wait for the job to complete. This is true not only for this trivial case, but for all distributions that do not have much of a tail.

The boundary of such distributions is the exponential distribution, defined by the pdf $f(x) = \lambda e^{-\lambda x}$. This distribution has the remarkable property of being memoryless. Its mean is $1/\lambda$, and this is *always* the value of the time you can expect to wait. When the job just starts, you can expect to wait $1/\lambda$ seconds. 7 seconds later, you can expect to wait an additional $1/\lambda$ seconds. And the same also holds half an hour later, or the next day. In other words, if you focus on the tail of the distribution, its shape is identical to the shape of the distribution as a whole.

fat-tailed distributions are even more counter-intuitive. They are characterized by the fact that the more you wait, *the more additional time* you should expect to wait. In other words, if you focus on the tail of the distribution, its shape has a fatter tail than the distribution as a whole.

Exercise 17 You are designing a system that handles overload conditions by actively killing one job, in the hope of reducing the load and thus being able to better serve the remaining jobs. Which job should you select to kill, assuming you know the type of distribution that best describes the job sizes?

Formal definition of heavy tails

The above can be formalized mathematically using the definition of a *heavy tail*. A distribution is said to possess a heavy tail if its tail decays according to a power law. This means that the probability of seeing very large values grows smaller and smaller, but not as fast as an exponential reduction. The equation is

$$\Pr[X > x] \approx x^{-\alpha} \quad 0 < \alpha < 2$$

The simplest example of such a distribution is the Pareto distribution, which we discuss below.

But real-life is not formal

There are two main problems with applying this formal definition to real-life situations.

First, the definition applies to values of x tending to infinity. In real life, everything is bounded to relatively small values. For example, we are typically not interested in jobs that run for more than a year (a mere 31,536,000 seconds), or in files of more than a terabyte or so; the highest values seen in most systems are considerably smaller.

Second, it is typically very hard to assess whether or not the tail really decays according to a power law: there are simply not enough observations. And from a practical point

of view, it typically doesn't really matter. Therefore we prefer not to get into arguments about formal definitions.

What does indeed matter is that there is a non-negligible probability to encounter extremely high values. For example, in an exponential distribution the probability of seeing a value that is 100 times (or more) larger than the mean is less than 10^{-43} . This is negligible, and can safely be ignored. In a fat-tailed distribution this probability can be as high as 10^{-5} . While still very small, this is non-negligible, and any non-trivial system can expect to encounter events with such a probability. In order to avoid formal arguments, we will therefore generally talk about fat-tailed distributions, and not claim that they necessarily conform with the formal definition of having a "heavy tail".

Example distributions

The Pareto distribution is defined on the range $x > 1$, and has a simple power-law CDF:

$$F(x) = 1 - x^{-a}$$

where a must be positive and is called the *shape parameter* — the lower a is, the heavier the tail of the distribution. In fact, the distribution only has a mean if $a > 1$ (otherwise the calculation of the mean does not converge), and only has a variance if $a > 2$ (ditto). The pdf is

$$f(x) = ax^{-(a+1)}$$

This is the simplest example of the group of heavy-tail distributions.

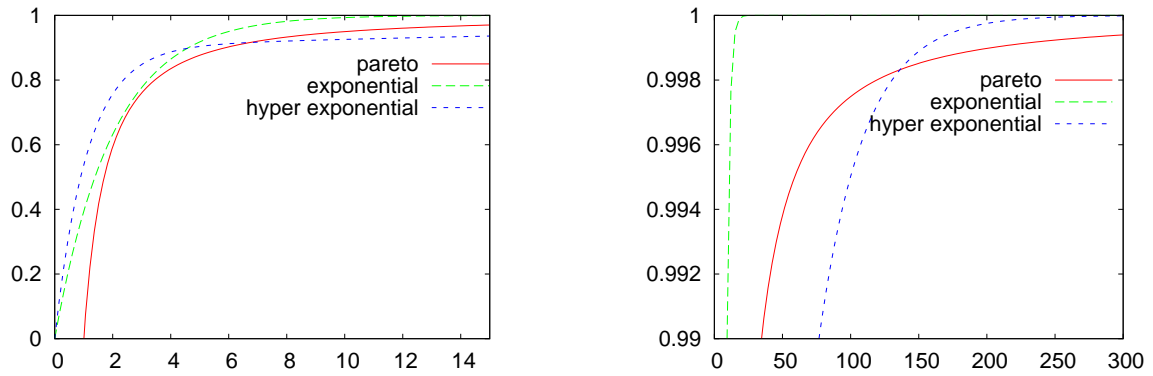
Exercise 18 *What is the mean of the Pareto distribution when it does exist?*

For modeling purposes, it is common to use a hyper-exponential distribution. This is the probabilistic combination of several exponentials, and can be tailored to mimic different tail shapes. For example, a two-stage hyper-exponential is

$$f(x) = p\lambda_1 e^{-\lambda_1 x} + (1-p)\lambda_2 e^{-\lambda_2 x}$$

for some $0 < p < 1$. By a judicious choice of λ_1 , λ_2 , and p , one can create a distribution in which the standard deviation is as large as desired relative to the mean, indicating a fat tail (as opposed to the exponential distribution, in which the standard deviation is always equal to the mean). However, this is not a heavy tail.

The following graphs compare the CDFs of the exponential, Pareto, and hyper-exponential distributions. Of course, these are special cases, as the exact shape depends on the parameters chosen for each one. The right graph focuses on the tail.



In many cases, the important characteristic of the workload is not the mathematical description of the tail, but the phenomenon of *mass-count disparity*. This means that there are many small elements, and a few huge ones, and moreover, that most of the “mass” is in the huge ones (technically, it means that if we integrate the distribution’s pdf, much of the sum comes from the tail). Instances of workloads that are often found to be fat-tailed and exhibit significant mass-count disparity include the following:

- The distribution of job runtimes. In this case, most of the jobs are short, but most of the CPU time is spent running long jobs.
- The distribution of file sizes in a file system. Here most of the files are small, but most of the disk space is devoted to storing large files.
- The distribution of flow sizes on the Internet. As with files, most flows are short, but most bytes transferred belong to long flows.

We will consider the details of specific examples when we need them.

Exercise 19 *The distribution of traffic flows in the Internet has been characterized as being composed of “mice and elephants”, indicating many small flows and few big ones. Is this a good characterization? Hint: think about the distribution’s modes.*

A special case of fat tailed distributions is the Zipf distribution. This distribution applies to a surprising number of instances of ranking by popularity, e.g. the popularity of web pages.

Details: the Zipf distribution

If the items being considered are ordered by rank, Zipf’s Law postulates that

$$\Pr(x) \propto \frac{1}{x}$$

that is, the probability of using each item is inversely proportional to its rank: if the most popular item is used k times, the second most popular is used $k/2$ times, the third $k/3$

times, and so on. Note that we can't express this in the conventional way using a probability density function, because $\int \frac{1}{x} dx = \infty!$ The only way out is to use a normalizing factor that is proportional to the number of observations being made. Assuming N observations, the probability of observing the item with rank x is then

$$f(x) = \frac{1}{x \ln N}$$

and the CDF is

$$F(x) = \frac{\ln x}{\ln N}$$

To read more: Advanced books on performance evaluation typically include some treatment of the above distributions. Examples include those by Jain [7] and Law and Kelton [11]. The volume edited by Adler et al. contains a collection of chapters on the subject, but fails to provide a basic and cohesive introduction [1]. Relatively readable papers on heavy tails include those by Crovella [2] and Downey [4]. Mass-count disparity is described by Feitelson [5]. Zipf's Law was introduced in 1949 based on empirical evidence regarding the relative frequency in which different words occur in written language [18].

Workload attributes may be inter-dependent

A problem with characterizing the workload only by the distributions is that the different workload attributes may be correlated with each other. For example, it may be that jobs that run longer also use more memory.

The conventional probabilistic definition of correlation measures the *linear* dependence between two random variables: a high correlation indicates that when one is above its mean, so is the other, and by approximately the same degree. Again, in real life things can be more complicated and harder to define.

Exercise 20 *Would you expect CPU usage and I/O activity to be correlated with each other?*

2.2.2 Workload Behavior Over Time

Long-range dependence is common

A special case of dependence in workloads is dependence along time, rather than among different parameters. For example, the load on the system at time t may be correlated to the load at time $t + \delta$ for some δ . In fact, empirical evidence shows that such correlations are quite common, and exist over long time frames.

The results of such long-range dependence is that the workload is bursty. In other words, workloads are usually not spread out evenly, but rather come in bursts. This occurs at many time scales, from sub-second to days and weeks. As a result the fluctuation in the workload observed during short periods may look similar to the fluctuations over long periods, an effect known as *self-similarity*. This is discussed briefly in Appendix B.

And there may be a local structure

While random sampling is a convenient model, it does not capture the dynamics of real workloads. In a real system, the workload over a relatively short time frame may be substantially different from the average of a whole year. For example, the characteristics of the load on a server in the Computer Science department may change from week to week depending on the programming exercise that everyone is working on that week.

In addition, workloads often display a daily cycle as a result of being generated by human beings who go to sleep at night. And there are also weekly and even yearly cycles (e.g. the workload in late December may be much lower than average).

Exercise 21 *What is the significance of the daily cycle in each of the following cases?*

1. *Access to web pages over the Internet*
2. *Usage of a word processor on a privately owned personal computer*
3. *Execution of computationally heavy tasks on a shared system*

2.3 Analysis, Simulation, and Measurement

There are three main approaches to get a handle on performance issue:

Analysis — use mathematical analysis from first principles to evaluate the system.

Simulation — simulate the system's operation. This is akin to measuring a small-scale partial implementation.

Measurement — implement the system in full and measure its performance directly.

Analysis provides insights

The major benefit of using mathematical analysis is that it provides the best insights: the result of analysis is a functional expression that shows how the performance depends on system parameters. For example, you can use analysis to answer the question of whether it is better to configure a system with one fast disk or two slower disks. We will see an example of this below, in Section 2.5.

Exercise 22 *Consider performance metrics like job response time and network bandwidth. What are system parameters that they may depend on?*

The drawback of analysis is that it is hard to do, in the sense that it is not always possible to arrive at a closed-form solution. Even when it is possible, various approximations and simplifications may have to be made in the interest of mathematical tractability. This reduces the confidence in the relevance of the results, because the simplifications may create unrealistic scenarios.

Simulation is flexible

The main advantage of simulation is its flexibility: you can make various modifications to the simulation model, and check their effect easily. In addition, you can make some parts of the model more detailed than others, if they are thought to be important. In particular, you do not need to make simplifying assumptions (although this is sometimes a good idea in order to get faster results). For example, you can simulate the system with either one or two disks, at many different levels of detail; this refers both to the disks themselves (e.g. use an average value for the seek time, or a detailed model that includes the acceleration and stabilization of the disk head) and to the workload that they serve (e.g. a simple sequence of requests to access consecutive blocks, or a sequence of requests recorded on a real system).

The drawback of simulations is that they are often perceived to be unreliable. After all, this is not a full system implementation but only the parts you think are important. But what if there are some effects you didn't think about in advance?

Measurements are convincing

The most convincing approach is to measure a real system. However, this only provides a single data point, and is not useful to compare different approaches or different configurations. For example, you can get detailed data about how the time for an I/O operation depends on the number of bytes accessed, but only for your current system configuration.

The perception of being “the real thing” may also be misguided. It is not easy to measure the features of a complex system. In many cases, the performance is a function of many different things, and it is impossible to uncover each one's contribution. Moreover, it often happens that the measurement is affected by factors that were not anticipated in advance, leading to results that don't seem to make sense.

Exercise 23 How would you go about measuring something that is very short, e.g. the overhead of trapping into the operating system?

In the end, simulation is often the only viable alternative

It would be wrong to read the preceding paragraphs as if all three alternatives have equal standing. The bottom line is that simulation is often used because it is the only viable alternative. Analysis may be too difficult, or may require too many simplifying assumptions. Once all the assumptions are listed, the confidence in the relevance of the results drops to the point that they are not considered worth the effort. And difficulties arise not only from trying to make realistic assumptions, but also from size. For example, it may be possible to analyze a network of half a dozen nodes, but not one with thousands of nodes.

Measurement is sometimes impossible because the system does not exist or it would take too much time. In other cases it is irrelevant because we cannot change the configuration as desired.

2.4 Modeling: the Realism/Complexity Tradeoff

Except for direct measurements, performance evaluations depend on a model of the studied system. This model can be made as detailed as desired, so as to better reflect reality. However, this leads to two problems. First, a more detailed model requires more data about the workload, the environment, and the system itself, data that is not always available. Second, a more detailed model is naturally harder to evaluate.

Statics are simpler than dynamics

One way to simplify the model is to use a static workload rather than a dynamic one. For example, a scheduler can be evaluated by how well it handles a given job mix, disregarding the changes that occur when additional jobs arrive or existing ones terminate.

The justification is that the static workload is considered to be a snapshot of the real dynamic workload. By freezing the dynamic changes in the workload, one saves the need to explicitly model these dynamics. By repeatedly evaluating the system under different static workloads, one may endeavor to capture the behavior of the system as it would be in different instants of its dynamic evolution.

But real systems are typically dynamic

The problem with using static workloads is that this leads to lesser accuracy and less confidence in the evaluations results. This happens because incremental work, as in a dynamically evolving real system, modifies the conditions under which the system operates. Creating static workloads may miss such effects.

For example, the layout of files on a disk is different if they are stored in a disk that was initially empty, or in a disk that has been heavily used for some time. When storing data for the first time in a new file system, blocks will tend to be allocated consecutively one after the other. Even if many different mixes of files are considered, they will each lead to consecutive allocation, because each time the evaluation starts from scratch. But in a real file system after heavy usage — including many file deletions — the available disk space will become fragmented. Thus in a live file system there is little chance that blocks will be allocated consecutively, and evaluations based on allocations that start from scratch will lead to overly optimistic performance results. The solution in this case is to develop a model of the steady state load on a disk, and use this to prime each evaluation rather than starting from scratch [15].

And of course distributions are important too

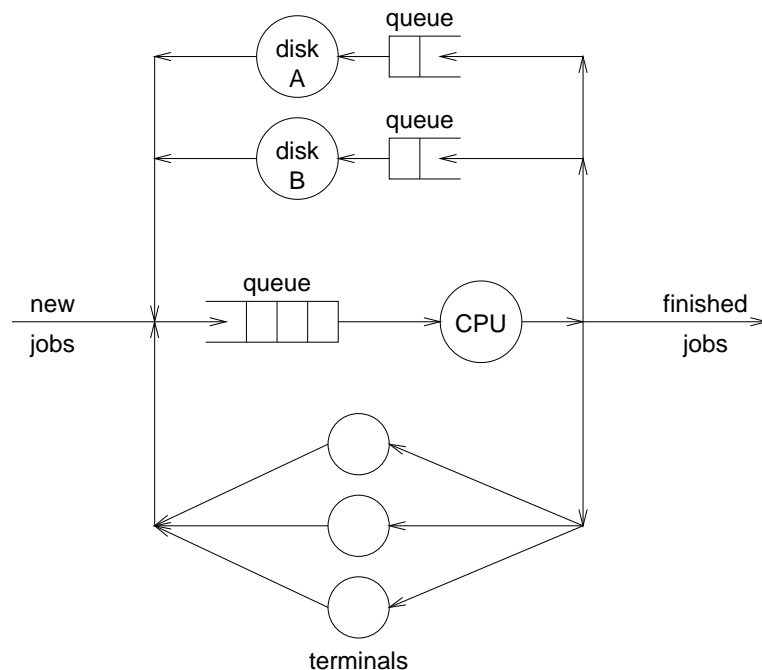
Another aspect of the realism/complexity tradeoff is the choice of distributions used to represent the workload. The whole previous section on workloads is actually about the need to create more complex and realistic models. If simple distributions are chosen, this may adversely affect the validity of the evaluation results.

2.5 Queueing Systems

2.5.1 Waiting in Queues

A system is composed of queues and servers

In real life, you often need to wait in queue for a service: there may be people ahead of you in the post office, the supermarket, the traffic light, etc. Computer systems are the same. A program might have to wait when another is running, or using the disk. It also might have to wait for a user to type in some input. Thus computer systems can be viewed as networks of queues and servers. Here is an example:



In this figure, the CPU is one service station, and there is a queue of jobs waiting for it. After running for some time, a job may either terminate, require service from one of two disks (where it might have to wait in queue again), or require input. Input terminals are modeled as a so called “delay center”, where you don’t have to wait in a queue (each job has its own user), but you do need to wait for the service (the user’s input).

Exercise 24 *Draw queuing networks describing queueing in a supermarket, a bank, and a cafeteria.*

Events happen at random times

It is most realistic to model arrival times and service times as random processes. This means that when we say that “on average, there are 3 jobs per minute”, we do *not* mean that jobs arrive exactly 20 seconds apart. On the contrary, in some cases they may come 2 seconds apart, and in some cases 3 minutes may go by with no new job. The probability of this happening depends on the distribution of interarrival times.

Randomness is what makes you wait in queue

The random nature of arrivals and service times has profound implications on performance.

Consider an example where each job takes exactly 100 ms (that is, one tenth of a second). Obviously, if exactly one such job arrives every second then it will be done in 100 ms, and the CPU will be idle 90% of the time. If jobs arrive exactly half a second apart, they still will be serviced immediately, and the CPU will be 80% idle. Even if these jobs arrive each 100 ms they can still be serviced immediately, and we can achieve 100% utilization.

But if jobs take 100 ms *on average*, it means that some may be much longer. And if 5 such jobs arrive each second *on average*, it means that there will be seconds when many more than 5 arrive together. If either of these things happens, jobs will have to await their turn, and this may take a long time. It is not that the CPU cannot handle the load on average — in fact, it is 50% idle! The problem is that it cannot handle multiple jobs at once when a burst of activity happens at random.

Exercise 25 *Is it possible that a system will not be able to process all its workload, even if it is idle much of the time?*

We would therefore expect the average response time to rise when the load increases. But how much?

2.5.2 Queueing Analysis

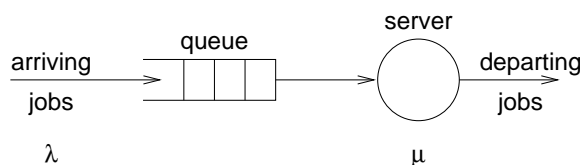
Queueing analysis is used to gain insights into the effect of randomness on waiting time, and show that these effects are derived from basic principles. Similar effects should be observed when using simulations and measurements, assuming they relate to the same system models.

To read more: The following is only a very brief introduction to the ideas of queueing theory. A short exposition on queueing analysis was given in early editions of Stallings [16, appendix A]. A more detailed discussion is given by Jain [7, Part VI]. Krakowiak [10, Chap. 8] bases the

general discussion of resource allocation on queueing theory. Then there are whole books devoted to queueing theory and its use in computer performance evaluation, such as Lazowska et al. [12] and Kleinrock [8, 9].

The simplest system has one queue and one server

Queueing analysis models the system as a collection of queues and servers. For example, a computer that runs jobs one after the other in the order that they arrive is modeled as a queue (representing the ready queue of waiting jobs) followed by a server (representing the CPU).



The main parameters of the model are the arrival rate and the service rate.

The *arrival rate*, denoted by λ , is the average number of clients (jobs) arriving per unit of time. For example, $\lambda = 2$ means that on average two jobs arrive every second. It also means that the average *interarrival time* is half a second.

The *service rate*, denoted by μ , is the average number of clients (jobs) that the server can service per unit of time. For example, $\mu = 3$ means that on average the server can service 3 jobs per second. It also means that the average *service time* is one third of a second.

The number of clients that the server actually serves depends on how many arrive. If the arrival rate is higher than the service rate, the queue will grow without a bound, and the system is said to be *saturated*. A *stable* system, that does not saturate, requires $\lambda < \mu$. The load or utilization of the system is $\rho = \lambda/\mu$.

Lots of interesting things can be measured

While only two numerical parameters are used, many different metrics can be quantified. A partial list of quantities that are commonly studied is

- The waiting time w .
- The service time s . According to our previous definition, $E[s] = 1/\mu$.
- The response time $r = w + s$. This is often the most direct metric for performance.
- the number of jobs in the system n (including those being serviced now). This is important in order to assess the size of system buffers and tables.

Note that the process by which jobs arrive at the queue and are serviced is a *random* process. The above quantities are therefore *random variables*. What we want to find out is usually the average values of metrics such as the response time and number of jobs in the system. We shall denote averages by a bar above the variable, as in \bar{n} .

Little's Law provides an important relationship

An important relationship between the above quantities, known as Little's Law, states that

$$\bar{n} = \lambda \cdot \bar{r}$$

Intuitively, this means that if λ jobs arrive each second, and they stay in the system for r seconds each, we can expect to see $\lambda \cdot r$ jobs in the system at any given moment. As a concrete example, consider a bar where 6 new customers arrive (on average) every hour, and each stays in the bar for (an average of) 3 hours. The number of customers we may expect to find in the bar is then 18: the 6 that arrived in the last hour, the 6 that arrived an hour earlier, and the 6 that arrived 2 hours ago. Any customers that arrived earlier than that are expected to have departed already.

This relationship is very useful, because if we know λ , and can find \bar{n} from our analysis, then we can compute \bar{r} , the average response time, which is the metric for performance.

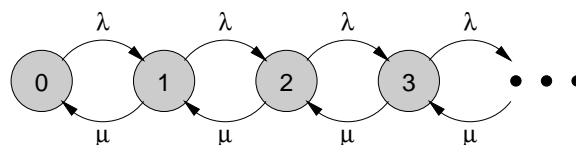
Exercise 26 Can you derive a more formal argument for Little's Law? Hint: look at the cumulative time spent in the system by all the jobs that arrive and are serviced during a long interval T .

The classic example is the M/M/1 queue

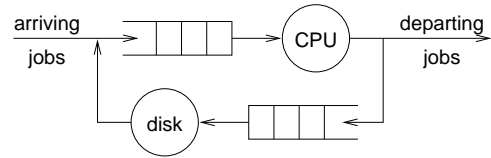
The simplest example is the so-called M/M/1 queue. This is a special case of the arrive-queue-server-done system pictured above. The first M means that interarrival times are exponentially distributed (the "M" stands for "memoryless"). The second M means that service times are also exponentially distributed. The 1 means that there is only one server.

Details of the analysis

The way to analyze such a queue (and indeed, more complicated queueing systems as well) is to examine its *state space*. For an M/M/1 queue, the state space is very simple. The states are labeled by integers 0, 1, 2, and so on, and denote the number of jobs currently in the system. An arrival causes a transition from state i to state $i + 1$. The average rate at which such transitions occur is simply λ , the arrival rate of new jobs. A departure (after a job is serviced) causes a transition from state i to state $i - 1$. This happens at an average rate of μ , the server's service rate.



Exercise 27 *What is the state space for a system with a CPU and a disk?*
Hint: think 2D.



The nice thing about this is that it is a *Markov chain*. The probability of moving from state i to state $i + 1$ or $i - 1$ does not depend on the history of how you got to state i (in general, it may depend on which state you are in, namely on i . For the simple case of an M/M/1 queue, it does not even depend on i).

An important property of Markov chains is that there is a limiting distribution on the states. This means that if we observe the system for a very long time, we will find that it spends a certain fraction of the time in state 0, a certain fraction of the time in state 1, and so on¹. We denote these long-term probabilities to be in the different states by π_i , i.e. π_0 will be the probability to be in state 0, π_1 the probability to be in state 1, etc.

Given the fact that such long-term probabilities exist, we realize that the flow between neighboring states must be balanced. In other words, for every transition from state i to state $i + 1$, there must be a corresponding transition back from state $i + 1$ to state i . But transitions occur at a known rate, that only depends on the fact that we are in the given state to begin with. This allows us to write a set of equations that express the balanced flow between neighboring states. For example, the flow from state 0 to state 1 is

$$\lambda \pi_0$$

because when we are in state 0 the flow occurs at a rate of λ . Likewise, the flow back from state 1 to state 0 is

$$\mu \pi_1$$

The balanced flow implies that

$$\lambda \pi_0 = \mu \pi_1$$

or that

$$\pi_1 = \frac{\lambda}{\mu} \pi_0$$

Now let's proceed to the next two states. Again, balanced flow implies that

$$\lambda \pi_1 = \mu \pi_2$$

which allows us to express π_2 as

$$\pi_2 = \frac{\lambda}{\mu} \pi_1$$

Substituting the expression for π_1 we derived above then leads to

¹Actually a Markov chain must satisfy several conditions for such limiting probabilities to exist; for example, there must be a path from every state to every other state, and there should not be any periodic cycles. For more on this see any book on probabilistic models, e.g. Ross [14]

$$\pi_2 = \left(\frac{\lambda}{\mu}\right)^2 \pi_0$$

and it is not hard to see that we can go on in this way and derive the general expression

$$\pi_i = \rho^i \pi_0$$

where $\rho = \frac{\lambda}{\mu}$.

Exercise 28 *What is the meaning of ρ ? Hint: it is actually the utilization of the system. Why is this so?*

Given that the probabilities for being in all the states must sum to 1, we have the additional condition that

$$\sum_{i=0}^{\infty} \pi_0 \rho^i = 1$$

Taking π_0 out of the sum and using the well-known formula for a geometric sum, $\sum_{i=0}^{\infty} \rho^i = \frac{1}{1-\rho}$, this leads to

$$\pi_0 = 1 - \rho$$

This even makes sense: the probability of being in state 0, where there are no jobs in the system, is 1 minus the utilization.

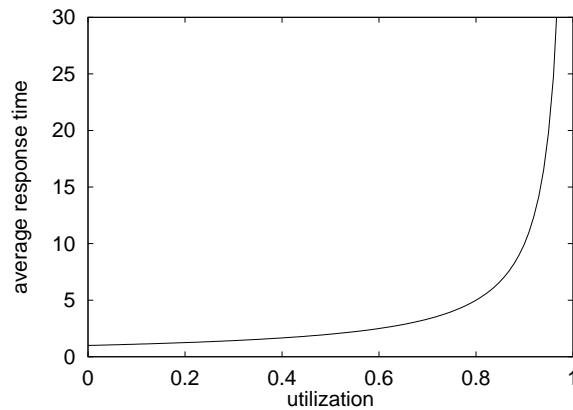
We're actually nearly done. Given the above, we can find the expected number of jobs in the system: it is

$$\begin{aligned} \bar{n} &= \sum_i i \pi_i \\ &= \sum_i i (1 - \rho) \rho^i \\ &= \frac{\rho}{1 - \rho} \end{aligned}$$

Finally, we use Little's Law to find the expected response time. It is

$$\begin{aligned} \bar{r} &= \frac{\bar{n}}{\lambda} \\ &= \frac{\rho}{\lambda(1 - \rho)} \\ &= \frac{1}{\mu - \lambda} \end{aligned}$$

The end result of all this analysis looks like this (by setting $\mu = 1$ and letting λ range from 0 to 1):



For low loads, the response time is good. Above 80% utilization, it becomes very bad.

Exercise 29 *What precisely is the average response time for very low loads? Does this make sense?*

The shape of this graph is characteristic of practically all queueing analyses for open systems. It is a result of the randomness of the arrivals and service times. Because of this randomness, customers sometimes cluster together and have to wait a long time. As the utilization approaches 100%, the system is idle less, and the chances of having to wait increase. Reducing the variability in arrivals and/or service times reduces the average response time.

This analysis is based on a host of simplifying assumptions

You probably didn't notice it, but the above analysis is based on many simplifying assumptions. The most important ones are

- The interarrival times and service times are exponentially distributed.
- The service discipline is FCFS.
- The population of clients (or jobs) is infinite, and the queue can grow to unbounded size.
- At any instant only one arrival or one departure may occur.

Exercise 30 *Can you identify at which point in the derivation each assumption was used?*

Nevertheless, the resulting analysis demonstrates the way in which response time depends on load. It gives a mathematical explanation to the well-known phenomenon that as the system approaches saturation, the queue length grows to infinity. This means that if we want short response times, we must accept the fact that utilization will be less than 100%.

2.5.3 Open vs. Closed Systems

Open systems lack feedback

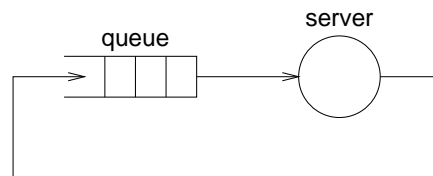
The M/M/1 queue considered above is an example of an *open system*: jobs arrive from an infinite population, pass through the system, and depart. In particular, there is no feedback from the service to the arrivals: new jobs continue to arrive at the same rate even if the service is abysmal.

The open system model is nevertheless quite useful, as situations in which arrivals are independent of the service quality indeed do exist. For example, this model may be very suitable for arrivals of requests at a web server, as such requests can be generated from around the world, and the users generating them may indeed not have prior experience with this server. Thus bad service to current requests does not translate into reduced future requests, because future requests are independent of the current ones. In addition, an open system model is the desired case when the system provides adequate service for the needs of its users.

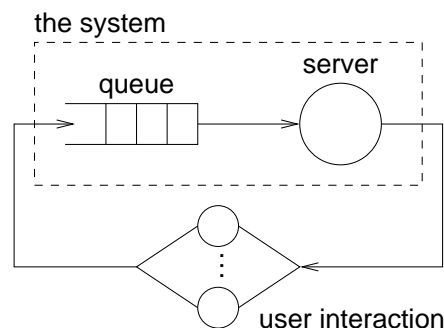
Closed systems model feedback explicitly

However, in many situations requests are not independent. For example, a local computer system may service a relatively small community of users. If these users learn (through experience or word of mouth) that the computer is not providing adequate service, they will probably stop submitting jobs.

Such scenarios are modeled by closed systems, in which the population of jobs is finite, and they continuously circulate through the system. In the basic, “pure” closed case, every job termination is immediately translated into a new job arrival:



A more realistic model is an interactive system, where the return path goes through a user-interaction component; in this case the number of jobs actually in the system may fluctuate, as different numbers may be waiting for user interaction at any given instance.



Exercise 31 *Consider the following scenarios. Which is better modeled as an open system, and which as a closed system?*

1. *A group of students have to complete an exercise in the operating systems course by 10 PM tonight.*
2. *A group of professors are trying to use the departmental server for non-urgent administrative chores related to courses they teach.*
3. *Tourists strolling by an information kiosk in a museum stop by to get some information.*

The metrics are different

As we saw, the performance of an open system like the M/M/1 queue can be quantified by the functional relationship of the response time on the load. For closed systems, this is irrelevant. A simple closed system like the one pictured above operates at full capacity all the time (that is, at the coveted 100% utilization), because whenever a job terminates a new one arrives to take its place. At the same time, it does not suffer from infinite response times, because the population of jobs is bounded.

The correct metrics for closed systems are therefore throughput metrics, and not response time metrics. The relevant question is how many jobs were completed in a unit of time, or in other words, how many cycles were completed.

2.6 Simulation Methodology

Analytical models enable the evaluation of simplified mathematical models of computer systems, typically in steady state, and using restricted workload models (e.g. exponential distributions). Simulations are not thus restricted — they can include whatever the modeler wishes to include, at the desired level of detail. Of course this is also a drawback, as they do not include whatever the modeler did not consider important.

To read more: Again, we only touch upon this subject here. Standard simulation methodology is covered in many textbooks, e.g. Jain [7, Part V], as well as textbooks dedicated to the topic such as Law and Kelton [11]. The issues discussed here are surveyed in [13].

2.6.1 Incremental Accuracy

One way to use simulations is to evaluate systems that cannot be solved analytically, but retaining the framework of steady state conditions. This means that the system is simulated until the measured features converge.

First you need to get into steady state

In order to simulate the system in a steady state, you must first ensure that it is in a steady state. This is not trivial, as simulations often start from an empty system. Thus the first part of the simulation is often discarded, so as to start the actual evaluation only when the system finally reaches the desired state.

For example, when trying to evaluate the response time of jobs in an open system, the first jobs find an empty system (no need to wait in queue), and fully available resources. Later jobs typically need to wait in the queue for some time, and the resources may be fragmented.

The decision regarding when steady state is achieved typically also depends on the convergence of the measured metrics.

The simulation length is determined by the desired accuracy

Once the steady state is reached, the simulation is continued until the desired accuracy is reached. As the simulation unfolds, we sample more and more instances of the performance metrics in which we are interested. For example, as we simulate more and more jobs, we collect samples of job response times. The average value of the samples is considered to be an estimator for the true average value of the metric, as it would be in a real system. The longer the simulation, the more samples we have, and the more confidence we have in the accuracy of the results. This derives from the fact that the size of the confidence interval (the range of values in which we think the true value resides with a certain degree of confidence) is inversely proportional to the square root of the number of samples.

Exercise 32 What sort of confidence levels and accuracies are desirable in systems evaluation?

2.6.2 Workloads: Overload and (Lack of) Steady State

While the simulation of systems at steady state is definitely useful, it is not clear that it captures the whole picture. In fact, are real systems ever in “steady state”?

Extreme conditions happen

It is well known that the largest strain on the worldwide telecommunications infrastructure occurs each year on Mother’s Day. Likewise, extreme loads on the Internet have occurred during live-cast events ranging from rock concerts to Victoria’s Secret fashion shows. Thus it would seem that systems actually have to be evaluated in two different modes: the steady state, in which “normal” conditions are studied, and the metrics are typically related to average response time, and overload conditions, in which the metrics are typically related to throughput. Note that paradoxically closed systems are the ones that model overload, despite the fact that overload results from

an effectively infinite population of clients that is unaware of the system conditions. This is so because whenever the system manages to get rid of a customer, another immediately comes in.

Naturally, one of the most important metrics for a system is the point at which it makes the transition from functioning to overloaded. This is also related to the question of the distribution of loads: is this a bimodal distribution, with most workloads falling in the well-behaved normal mode, and just some extreme discrete cases creating overload, or is there a continuum? The answer seems to be a combination of both. The load experienced by most systems is not steady, but exhibits fluctuations at many different time scales. The larger fluctuations, corresponding to the more extreme loads, occur much more rarely than the smaller fluctuations. The distinction into two modes is not part of the workload, but a feature of the system: above some threshold it ceases to handle the load, and the dynamics change. But as far as the workload itself is concerned, the above-threshold loads are just a more extreme case, which seems to be discrete only because it represents the tail of the distribution and occurs relatively rarely.

Simulations results are limited to a certain timeframe

What is the implication to simulations? The problem with extreme conditions is that they occur very rarely, because they are from the tail of the distribution of possible events. Thus if you observe the behavior of a real system over, say, one month, you *might* chance upon such an event, but most probably you will not. A good simulation would be the same: if you simulate a month, you might sample a high-load event, but most probably you will not. Thus your results are guaranteed to be wrong: if you do not have a high-load event, you fail to predict the outcome and effect of such events, and if you do, you fail to predict the normal behavior of the system.

The best solution to this problem is to acknowledge it. Simulations with bursty workloads should explicitly define a time horizon, and prohibit events that are not expected to occur within this time. The results are then at least relevant for an “average” time window of this duration [3].

Rare-event simulation techniques can be used to evaluate extreme cases

But what if the rare events are actually what we are interested in? For example, when evaluating communication networks, we are not interested only in the average latency of messages under normal conditions. We also want to know what is the probability of a buffer overflow that causes data to be lost. In a reasonable network this should be a rather rare event, and we would like to estimate just how rare.

Exercise 33 *What are other examples of such rare events that are important for the evaluation of an operating system?*

If, for example, the events in which we are interested occur with a probability of one in a million (10^{-6}), we will need to simulate billions of events to get a decent measurement of this effect. And 99.9999% of this work will be wasted because it is just filler between the events of interest. Clearly this is not an efficient way to perform such evaluations.

An alternative is to use rare event simulation techniques. This is essentially a combination of two parts: one is the assessment of how often the rare events occur, and the other is what happens when they do. By combining these two parts, one can determine the effect of the rare events on the system. The details of how to do this correctly and efficiently are non-trivial, and not yet part of standard methodology [6].

2.7 Summary

Performance is an important consideration for operating systems. It is true that the main consideration is functionality — that is, that the system will actually work. But it is also important that it will work efficiently and quickly.

The main points made in this chapter are:

- Performance evaluation must be done subject to realistic workloads. Workload analysis and modeling are indispensable. Without them, the results of a performance evaluation are largely meaningless.

In particular, realistic workloads are bursty and are characterized by heavy tailed distributions. They often do not conform to mathematically nice distributions such as the exponential and normal distributions.

- In many senses, operating systems are queueing systems, and handle jobs that wait for services. In open systems, where the job population is very big and largely independent of system performance, this means that
 1. Randomness in arrival and service times causes the average response time to tend to infinity as the load approaches each sub-system's capacity.
 2. This is a very general result and can be derived from basic principles.
 3. You cannot achieve 100% utilization, and in fact might be limited to much less.

In closed systems, on the other hand, there is a strong feedback from the system's performance to the creation of additional work. Therefore the average response time only grows linearly with the population size.

Disclaimer

Performance is paramount in system design, and research papers on computer systems always proudly exhibit a section professing the promising results of a performance evaluation. But all too often this is mainly a *characterization* of a proposed

system, not a deep evaluation of alternatives and tradeoffs. In addition, comparisons are hampered by the lack of agreed test conditions, metrics, and workloads.

In particular, the study of systems working in overloaded conditions, and of the characteristics and effect of workloads on a system's behavior, is in its infancy. Thus most of these notes do not make enough use of the issues discussed in this chapter — not enough research on these issues has been performed, and too little empirical data is available.

Bibliography

- [1] R. J. Adler, R. E. Feldman, and M. S. Taqqu (eds.), *A Practical Guide to Heavy Tails*. Birkhäuser, 1998.
- [2] M. E. Crovella, “Performance evaluation with heavy tailed distributions”. In *Job Scheduling Strategies for Parallel Processing*, D. G. Feitelson and L. Rudolph (eds.), pp. 1–10, Springer Verlag, 2001. Lect. Notes Comput. Sci. vol. 2221.
- [3] M. E. Crovella and L. Lipsky, “Long-lasting transient conditions in simulations with heavy-tailed workloads”. In *Winter Simulation conf.*, Dec 1997.
- [4] A. B. Downey, “Lognormal and Pareto distributions in the Internet”. *Comput. Commun.* **28(7)**, pp. 790–801, May 2005.
- [5] D. G. Feitelson, “Metrics for mass-count disparity”. In *14th Modeling, Anal. & Simulation of Comput. & Telecomm. Syst.*, pp. 61–68, Sep 2006.
- [6] P. Heidelberger, “Fast simulation of rare events in queueing and reliability models”. *ACM Trans. Modeling & Comput. Simulation* **5(1)**, pp. 43–85, Jan 1995.
- [7] R. Jain, *The Art of Computer Systems Performance Analysis*. John Wiley & Sons, 1991.
- [8] L. Kleinrock, *Queueing Systems, Vol I: Theory*. John Wiley & Sons, 1975.
- [9] L. Kleinrock, *Queueing Systems, Vol II: Computer Applications*. John Wiley & Sons, 1976.
- [10] S. Krakowiak, *Principles of Operating Systems*. MIT Press, 1988.
- [11] A. M. Law and W. D. Kelton, *Simulation Modeling and Analysis*. McGraw Hill, 3rd ed., 2000.
- [12] E. D. Lazowska, J. Zahorjan, G. S. Graham, and K. C. Sevcik, *Quantitative System Performance: Computer System Analysis Using Queueing Network Models*. Prentice-Hall, Inc., 1984.

- [13] K. Pawlikowski, “Steady-state simulation of queueing processes: a survey of problems and solutions”. *ACM Comput. Surv.* **22(2)**, pp. 123–170, Jun 1990.
- [14] S. M. Ross, *Introduction to Probability Models*. Academic Press, 5th ed., 1993.
- [15] 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.
- [16] W. Stallings, *Operating Systems: Internals and Design Principles*. Prentice-Hall, 3rd ed., 1998.
- [17] “Standard performance evaluation corporation”. URL <http://www.spec.org>.
- [18] G. K. Zipf, *Human Behavior and the Principle of Least Effort*. Addison-Wesley, 1949.