Experience with Using the Parallel Workloads Archive

Dror G. Feitelson[1]* Dan Tsafrir[2] David Krakov[1]

[1]*Dept. Computer Science, The Hebrew University, 91904 Jerusalem, Israel*
[2]*Computer Science Dept., Technion – Israel Institute of Technology, 32000 Haifa, Israel*

Abstract

Science is based upon observation. The scientific study of complex computer systems should therefore be based on observation of how they are used in practice, as opposed to how they are assumed to be used or how they were designed to be used. In particular, detailed workload logs from real computer systems are invaluable for research on performance evaluation and for designing new systems.

Regrettably, workload data may suffer from quality issues that might distort the study results, just as scientific observations in other fields may suffer from measurement errors. The cumulative experience with the Parallel Workloads Archive, a repository of job-level usage data from large-scale parallel supercomputers, clusters, and grids, has exposed many such issues. Importantly, these issues were not anticipated when the data was collected, and uncovering them was not trivial. As the data in this archive is used in hundreds of studies, it is necessary to describe and debate procedures that may be used to improve its data quality. Specifically, we consider issues like missing data, inconsistent data, erroneous data, system configuration changes during the logging period, and unrepresentative user behavior. Some of these may be countered by filtering out the problematic data items. In other cases, being cognizant of the problems may affect the decision of which datasets to use. While grounded in the specific domain of parallel jobs, our findings and suggested procedures can also inform similar situations in other domains.

Keywords: Workload log, Data quality, Parallel job scheduling

1. Introduction

The study and design of computer systems requires good data regarding the workload to which these systems are subjected, because the workload has a decisive effect on the observed performance [1, 15, 38]. As an example, consider the question of scheduling parallel jobs on a large-scale cluster or supercomputer. As each job may require a different number of processors, this is akin to bin packing [7, 25, 36, 48]. Hence the best scheduling algorithm may depend on the distribution of job sizes, or on the possible correlation between job size and runtime [27].

But how can we know what the distribution is going to be? The common approach is to collect data logs from existing systems and to assume that future workloads will be similar. The Parallel Workloads Archive, whose data is the focus of this paper, is a repository of such logs; it is accessible at URL www.cs.huji.ac.il/labs/parallel/workload/. The archived logs (see Table 1) contain accounting data about the jobs that executed on parallel supercomputers, clusters, and grids, which is necessary in order to evaluate schedulers for such systems. These logs have been used in many hundreds of research papers since the archive was started in 1999. Figure 1 shows the accumulated number of hits that the parallel workload archive gets when searching for it in Google Scholar (supplemented by the number of hits associated with the Grid Workloads Archive [21], which serves a similar purpose). The high citation count bears witness to the need for such data in the research community and highlights the importance of using the data judiciously.

At first blush it seems that accounting logs should provide reliable and consistent data. After all, this is just a mechanistic and straightforward recording of events that happened on a computer system (as opposed to, say, genome

* Contact information: email feit@cs.huji.ac.il, phone/fax +97225494555.

Table 1: *Main logs in the Parallel Workloads Archive. (Some additional logs with mainly serial jobs or low utilizations are not listed.)*

log	*period*	*months*	*PEs*	*users*	*jobs*	*util.*	*file*	*cleaned*
NASA iPSC	10/93–12/93	3	128	69	42,264	0.47	NASA-iPSC-1993-3.swf	yes
LANL CM5	10/94–09/96	24	1,024	213	201,387	0.75	LANL-CM5-1994-4.swf	yes
SDSC Par95	12/94–12/95	12	400	98	76,872	0.72	SDSC-Par-1995-3.swf	yes
SDSC Par96	12/95–12/96	12	400	60	38,719	0.76	SDSC-Par-1996-3.swf	yes
CTC SP2	06/96–05/97	11	338	679	79,302	0.85	CTC-SP2-1996-3.swf	yes
KTH SP2	09/96–08/97	11	100	214	28,489	0.70	KTH-SP2-1996-2.swf	
SDSC SP2	04/98–04/00	24	128	437	73,496	0.84	SDSC-SP2-1998-4.swf	yes
LANL O2K	11/99–04/00	5	2,048	337	122,233	0.70	LANL-O2K-1999-2.swf	
OSC cluster	01/00–11/01	22	178	254	80,714	0.14	OSC-Clust-2000-3.swf	yes
SDSC Blue	04/00–01/03	32	1,152	468	250,440	0.77	SDSC-BLUE-2000-4.swf	yes
Sandia Ross	11/01–01/05	37	1,524	204	85,355	0.50	Sandia-Ross-2001-1.swf	
HPC2N	07/02–01/06	42	240	258	527,371	0.70	HPC2N-2002-2.swf	yes
SDSC Datastar	03/04–04/05	13	1,664	460	96,089	0.63	SDSC-DS-2004-2.swf	
SHARCNET	12/05–01/07	13	6,828	412	1,195,242	n/a	SHARCNET-2005-2.swf	
LLNL uBGL	11/06–06/07	7	2,048	62	112,611	0.56	LLNL-uBGL-2006-2.swf	
LLNL Atlas	11/06–06/07	8	9,216	132	60,332	0.64	LLNL-Atlas-2006-2.swf	yes
LLNL Thunder	01/07–06/07	5	4,008	283	128,662	0.88	LLNL-Thunder-2007-1.swf	yes
MetaCentrum	12/08–06/09	7	806	147	103,656	0.36	METACENTRUM-2009-2.swf	
ANL Intrepid	01/09–09/09	8	163,840	236	68,936	0.60	ANL-Intrepid-2009-1.swf	
PIK IPLEX	04/09–07/12	40	2,560	225	742,965	0.38	PIK-IPLEX-2009-1.swf	
RICC	05/10–09/10	5	8,192	176	447,794	0.87	RICC-2010-2.swf	
CEA Curie	02/11–10/12	20	93,312	722	773,138	0.29	CEA-Curie-2011-2.swf	yes

"PEs" was nodes or CPUs in old logs, today it typically represents cores.

"util" is the system utilization, i.e. the fraction of the resources that were allocated to jobs. It is not computed for SHARCNET because this is a grid system, and the constituent clusters became available at different times.

File names include a version number, as most logs were re-converted to swf when errors were found or new considerations were introduced.

"cleaned" specifies whether a cleaned version exists, where problematic data has been filtered out.

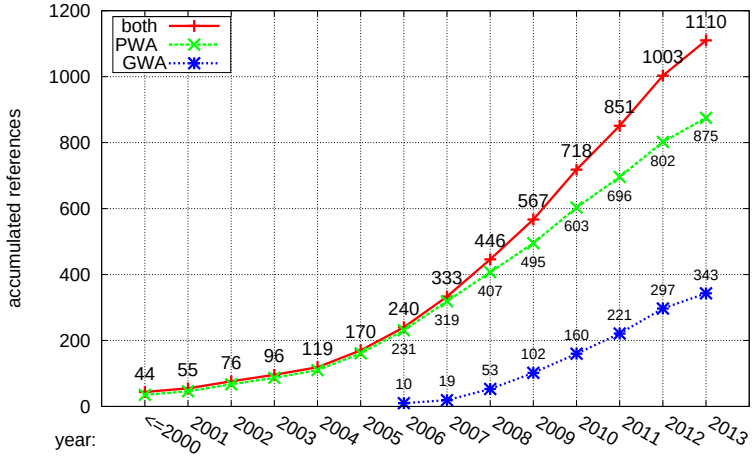

Figure 1: *Accumulated yearly number of hits received when searching for the Parallel Workloads Archive (PWA) and the Grid Workloads Archive (GWA) in Google Scholar as of 28 October 2013. GWA contains those logs from PWA that pertain to grid systems, as well as a few other grid logs. The query used was "Parallel Workload(s) Archive" (both singular and plural) and the archive's URL, and likewise for the grid archive. Papers that cite both archives are only counted once in "both".*

data, which is obtained via complex experimental procedures that lead to intrinsic errors [30]). But upon inspection, we find that the available logs are often deficient. This is not a specific problem with the data that is available to us. All such logs have data quality problems, and in fact the logs available in the Parallel Workloads archive actually represent relatively good data. We have additional logs that were never made public in the archive because an initial investigation found the data contained in them to be so lacking.

The issue of data quality has a long history (the International Conference on Information Quality has been held annually since 1996). The most general definition of data quality is "fitness for use", implying that it is not an objective but rather a context-sensitive attribute [45]. Indeed, work on data quality has identified no less than 20 dimensions of data quality, the top five of which are accuracy, consistency, security, timeliness, and completeness [23]. In the context of computer systems, practically all discussions have been about the quality of data *handled* by the system, e.g. the data contained in enterprise databases [6, 28]. Low quality data has been blamed for bad business decisions, lost revenue, and even implicated in catastrophes leading to the loss of human life [16, 17, 31]. The quality of data in scientific repositories, such as biological genome data, has also been studied, both to assess the quality of existing repositories and to suggest ways to improve data quality [19, 26, 30]. Likewise, there have been problems with repositories used for empirical software engineering research; for example, massive repetitions of records taint evaluations of learning schemes that attempt to identify defective modules, by causing overlaps between the training and test datasets [18, 34].

At the same time, there has been little if any work on the quality of data *describing* computer systems, such as workload data. In this paper we report on our experience with the data available in the Parallel Workloads Archive. We start the discussion by considering log formats in Section 2. The main problem here is representational aspects of data quality, where the same field in different logs may have slightly different semantics. The bulk of the paper is contained in Section 3, which lists and classifies known problems in the different logs. These are mainly intrinsic correctness problems, such as inconsistency (redundant data fields should not contradict each other), errors (data should not imply that the number of processors being used at a certain instant is larger than the number available in the machine), and missing data in certain records and fields. In addition, there are problems of representativeness, as when logs include high-volume abnormal activity by a small set of users. Due to the data quality problems we have found, using log data as-is (even as input to a statistical analysis) might lead to unreliable results. Section 4 then outlines actions that we have taken to improve data quality and make the logs more useful. The conclusions are presented in Section 5, and include a perspective of our work in relation to the work on data quality in other domains.

The main contribution of this work is to promote solid experimental work on the evaluation of parallel systems, and to strengthen the scientific basis of such studies. Science is based, among other things, on observation. The experimental procedures used to obtain data are an important part of any science. Regrettably, Computer Science lags behind in this respect, and we do not have a data-driven culture as in other fields [10]. In particular, researchers are often unaware of data quality issues. This paper is dedicated to improving this situation by recording the considerations behind the procedures that were used to handle the data made available in the Parallel Workloads Archive. These procedures represent over a decade of research on data quality issues in these logs, including the identification of many unexpected problems. The evaluation of the data is also important in order to provide context for the hundreds of papers that use this data, and to validate the data on which they are based. It should be noted that the procedures we use are non-trivial and not self evident. By publicizing them, we hope to also initiate a debate about data quality and data cleaning in experimental computer systems research, a subject which has not received sufficient attention to date.

2. Log Formats

A pre-requisite for analyzing logs is being able to parse them. In some classes of systems, such as web servers, standard log formats have been defined. Regrettably, there is no such standard for parallel job schedulers, and each one has defined its own format with its own idiosyncrasies. To ease work with the logs, we defined a Standard Workload Format[1] for use in the archive [3]. This format was proposed by David Talby and refined through discussions with James Patton Jones and others.

[1] Files in the **s**tandard **w**orkload **f**ormat were naturally denoted by the suffix .swf. Unfortunately, this suffix was later also adopted for **s**hock**w**ave **f**lash files.

The considerations applied in designing the standard format included the following.

- It should be easy to parse. The chosen format is an ASCII file with one line per job, space-separated fields, and exclusive use of numerical values (that is, no strings and special date or time formats). Fields for which data is unavailable are given as -1.
- It should be well defined. We sacrificed extensibility in the interest of standardization, and require that data be expressed in given units. Regrettably, this also means that sometimes data that is actually available in a log does not have a corresponding field in the format, and is therefore lost in the conversion process. For example, this happens for the data about suspending and resuming jobs that is available in the SHARCNET log. It is therefore important to also maintain the original log file.
- It should be general. In particular, the same format is suitable both for logs from production machines and for statistical models. For example, this consideration favors the use of the time triplet $\langle$submit, wait, run$\rangle$ over the triplet $\langle$submit, start, end$\rangle$, because wait and run times better separate the effect of the scheduler and the application. When used for the output of a model, the wait time can be left undefined.
- It should be safe [32]. To preserve privacy, users and applications are replaced by numerical codes that are allocated in order of first appearance.

Of course, striving for consistency does not mean that it can always be achieved. An example in point is the very basic data about runtime, typically expressed in logs by the combination of start time and end time. The problem is that the precise semantics of these fields are usually ill-defined. Thus start time may refer to the time that the scheduler decided to start the job, or the time when the first process was started, or the time when the last process was started, or perhaps the time when the logging facility was notified that the job was started. Likewise, end time may refer to the time that the first process terminated, the time that the last one terminated, or the time when this was recorded.

For example, the KTH SP2 log includes a field called uwall giving the used wallclock time, which intuitively seems to correspond to the runtime. However, uwall is actually defined to be the interval from the last node allocation to the first node deallocation. Note that this may be negative if processes fail immediately, and there is no period of time when they are all actually running in parallel. Therefore, in the conversion to the standard format, we elected to use the more commonly used start and end times (even though their precise semantics are unknown). Another problem in the KTH SP2 log is that the system administrators sometimes faked the submit times in order to boost a job's priority. Such cases were identified by comparing the submit time field with the submit time that was encoded in the job ID. A similar problem occurs in the LANL O2K log format, which does not contain an explicit field specifying the job end time. The field specifying the time that the job-termination event was logged was used instead.

Another notoriously problematic field is the job status. In many cases a successful completion status is recorded only if the job terminated with a 0 exit code. While this has been the convention on Unix systems since their inception, there is no guarantee that applications indeed follow it. In cases where jobs do not have a "success" status, we interpret "failed" as jobs that started to run but suffered from some problem or exception condition, and "canceled" as jobs that were killed by the user. In the latter case, a job could have been canceled before it started to run, in which case its runtime and allocated processors may be undefined. However, there is no guarantee that logs indeed use the terminology in the same way we interpret it. Thus it is dangerous to filter jobs based on their recorded status.

The Standard Workload Format was established when the main concerns were the arrivals of jobs and their basic resource requirements, namely processors and compute time. It serendipitously included a field used to specify the partition used to run the job, which has since been found to be useful to represent data about grids including multiple clusters (e.g. SHARCNET and MetaCentrum). However, it cannot handle more complex data requirements. For example, it has been suggested that information about specific job requirements and specific capabilities of different clusters may lead to involved and limiting constraints, which induce significant complexity on the scheduling, and lead to greatly reduced performance [22]. This cannot be expressed using the current version of the standard format. Likewise, it has been suggested that it may be important to follow the dynamics of resource usage during the execution of a job, by sampling them at regular intervals. To store such data, one needs to augment the standard workload data with additional data that includes multiple (and potentially very many) records for each job [5]. This leads to a database-like structure, where one table includes the original general data about all the jobs, and another table includes the dynamic records. The tables can be associated with each other based on the job ID.

Finally, the standard format does not include facilities for distinguishing between nodes, processors, and cores. However, this is believed not to be very important, because allocating a full node to a task rather than just a single core is usually a disguise for allocating all the node's memory to the task. It is better to express this directly as an allocation of memory, which is possible in the standard format.

3. Problems with Log Data

Over the years, the logs available at the Parallel Workloads Archive have been found to contain various problems. This is not unique to this repository — collected data in practically all fields are known to have problems. It also does not detract from the importance and in many cases also not from the usefulness of the data. However, it is definitely desirable to be cognizant of the problems and deal with them when possible. Importantly, most of the problems are not isolated anecdotes but rather are repeated in many logs. We therefore present multiple examples of each one in the following subsections. Cases which are indeed unique are identified as such.

3.1. Incomplete Data

One problem that we sometimes encounter is that the data is incomplete. This means that some important information is simply missing. As a result the usability of the available data is limited. In the following we provide some examples.

The vast majority of parallel supercomputers and clusters dedicate processors to jobs. This means that when a job is scheduled, a certain partition of the machine is carved out for it. The job is then run on the processors in this partition until it terminates. Upon termination, the processors become free and can then be allocated to another job. But it is also possible to use time slicing. The Connection Machine CM-5 was one of the only commercial parallel supercomputers to support gang scheduling [40]. This meant that it could context switch from one parallel job to another. And indeed the LANL CM5 log includes an indication of whether jobs ran on dedicated nodes or not. In those cases where jobs did not run on dedicated nodes, the implication is that they did not run for the full duration from their start time to their end time. However, there is no indication of precisely what fraction of the time was actually used. As a result the real runtimes are actually unknown. Naturally this makes the data practically unusable for simulations of job scheduling and for analyzing utilization. However, it can still be used to study the arrival process, user behavior, memory usage [8], etc.

Another example comes from the SDSC Paragon logs. The data here is given as two separate logs: one for 1995, and the other for 1996. In the interest of preserving privacy, user names were replaced by random numbers in the original logs. Regrettably, this user numbering was inconsistent in 1995 and 1996, and the mapping from the 1995 numbers to the 1996 numbers is not available. Hence the logs cannot be united into a single longer log (unless user information is deemed unimportant), but each can be used in isolation.

A third example is provided by the NASA iPSC log. This log simply does not include submit times at all — only start times and run times. Similarly, the LLNL Atlas and Thunder logs include only start and end times. In the conversion to the standard format we therefore use start times to also represent arrival times. Obviously, this data cannot be used to study the arrival process, as the recorded start times reflect the combined effect of the original arrivals and the wait time. Wait times distort arrival data because they may be influenced by priorities of the scheduling policy. They may also reflect a smoothing out of load [12]. However, the logs can still be used to obtain a lot of useful data, and in fact the NASA iPSC log was the first log to be analyzed in detail [13].

Other fields that are often missing from logs are memory usage, CPU time, and requested resources (in distinction from the resources that were actually used). These are important for studies that need this data, but are not needed for the simplest scheduling studies that consider only processors and runtime.

In addition to fields that are totally absent, it is not uncommon for data to be missing only for a subset of the jobs. Table 2 shows that in most cases submit, start, and end times are missing only for a small fraction of the jobs (except for those logs where submit times are just not available at all). Fields like CPU time or memory used tend to be missing much more often.

Table 2: *Occurrences of incomplete or inconsistent data in the different logs.*

		missing			zero				negative		more than req.			CPU
log	jobs	submit	start	end	proc	run	CPU	mem	wait	run	run	proc	mem	>run
NASA iPSC	42,264	n/a	–	n/a	–	–	n/a	n/a	n/a	–	n/a	n/a	n/a	n/a
LANL CM5	201,387	3	3	–	–	–	37,199	19,517	–	1	36,198	1,212	21,036	17
SDSC Par	115,591	1,608	23	14	–	–	6,181	n/a	27	15	–	–	n/a	3,073
CTC SP2	79,302	–	–	–	–	6	4	n/a	–	–	1,380	–	n/a	155
KTH SP2	28,490	–	–	–	–	–	n/a	n/a	–	–	64	219	n/a	n/a
SDSC SP2	73,496	–	2	–	–	–	1,731	–	–	–	463	–	–	3
LANL O2K	122,233	–	–	–	–	–	21,156	221	–	–	–	–	–	1,886
OSC cluster	80,714	–	1	–	–	–	6,177	n/a	1	–	–	–	n/a	27,596
SDSC Blue	250,440	–	262	–	2	–	4,203	n/a	28	–	8,167	458	n/a	2
Sandia Ross	85,355	–	–	–	1	–	807	1,548	–	–	3,069	–	–	
HPC2N	527,371	–	–	77	–	–	73,483	5,646	12	3	6,784	767	2,548	60,608
SDSC Datastar	96,089	–	4	149	–	–	8,976	n/a	12	87	1,044	–	n/a	149
SHARCNET	1,195,242	–	26	12,389	–	–	78	16,231	–	–	–	–	–	1237
LLNL Atlas	60,332	n/a	–	–	–	–	n/a	n/a	–	–	–	19	n/a	n/a
LLNL Thunder	128,662	n/a	1208	1,208	–	–	n/a	n/a	–	–	–	155	n/a	n/a
MetaCentrum	103,656	–	–	–	–	–	n/a	8	–	–	–	–	–	n/a
ANL Intrepid	68,936	–	–	–	–	–	n/a	n/a	–	–	9,096	30,948	n/a	n/a
PIK IPLEX	742,965	–	10,710	–	1	–	68,191	–	45	83	2,581	2	–	2,632
RICC	447,794	–	–	–	–	–	n/a	n/a	–	–	2,581	–	n/a	n/a
CEA Curie	773,138	–	561	–	1,229	–	n/a	n/a	–	–	4,848	105,019	n/a	n/a

"–" means that there were no such inconsistencies. "n/a" means not applicable, e.g. if the log does not contain such data at all. For runtime and wait time, more than requested or negative is by a margin of 1 minute or more to allow for clock skew or notification delays. Missing start time and 0 processors/CPU/memory are counted only for jobs that had a "success" status (but missing start time with CPU>0 is noted). In the Curie log, the 561 jobs with missing start time actually represent missing run time.

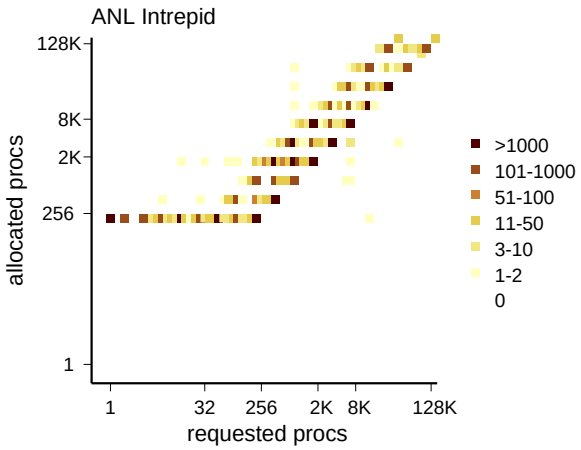

Figure 2: *Allocation of processors on the ANL Intrepid machine. Allocating more than the number requested may result from fragmentation (rounding up to a possible partition size) or from the need to allocate all the memory in a node to a single process, rather than sharing it among processes running on multiple cores.*

3.2. Inconsistent Data

Another type of problem is inconsistent data. This means that the data in the log contradicts itself, and does not pass some simple sanity check (called "integrity constraints" in [6]).

Table 2 lists several circumstances that are easily identified as inconsistent. For example, if a job ran successfully then various resource-usage metrics must be positive or at least non-negative: the wait time, the runtime, the number of processors used, the amount of memory used, etc. Likewise, the average CPU time used per processor cannot be larger than the wallclock running time of the whole job. In some cases it also does not make sense for a job to receive more resources than it had asked for, but such an inconsistency is merely puzzling but not impossible.

It should be noted that timing inconsistencies do not necessarily indicate a real problem. Some cases of zero runtime, for example, could be the result of a resolution problem, e.g. when runtime is measured in seconds and the job's runtime is smaller than half a second. This is unlikely, however, because the distribution of runtimes usually starts at several seconds, and sometimes at tens of seconds. Shorter runtimes cannot be recorded simply due to the delay associated with setting up all the parallel processes and receiving notifications regarding their terminations. (Note that in distinction from measured runtime, *requested* runtime should not be 0, so this is considered an error and not an inconsistency or a resolution problem.)

Negative times may result from clock skew or from notification delays between node daemons and a frontend workstation. Therefore we report only differences of more than 1 minute in Table 2. This filtering may be very meaningful. For example, in the SDSC-SP2 log 4291 jobs got more runtime than they requested, but in only 463 of these the difference was larger than 1 minute. A negative runtime occurred 1 time and negative wait times occurred 183 times, and these were all smaller than 1 minute and therefore considered insignificant.

Inconsistent data is of course not limited to time fields. In the SDSC Blue log, 253 jobs got less processors than they requested. This may look very strange, as it is unclear how a job could run on less processors than it requires. However, parallel jobs are often coded in a style that can use any given number of nodes, and receive the number actually used in a certain run as a parameter.

The opposite may also occur, but often this is not a real problem. On many parallel machines processors are allocated in predefined partitions, and there is a minimal partition size. In some cases this corresponds to the number of processors (or cores) in a node. In other cases the minimal partition may include many nodes. For example, the ANL Intrepid machine consists of 40 racks, housing 40,960 quad-core nodes, and partition sizes are powers of two. Moreover, in 8 racks the minimal partition size is 64 nodes (256 cores), and in the rest the minimal size is 512 nodes (2048 cores). Jobs that require less are nevertheless allocated these sizes, and the extra processors are lost to

Table 3: *Example of possible actions when facing inconsistent timing data.*

action	*submit*	*wait*	*run*
none	unchanged	-55:34m	59:05m
start=submit	unchanged	0	03:35m
submit=start	changed	0	59:05m
start=submit end+=submit-start	unchanged	0	59:05m

fragmentation. Similar rounding up is done on other machines as well. But in many logs we don't know how many are actually used and how many are lost.

In addition to partition size restrictions, over-allocation of processors may be a by-product of allocating memory. Using the Intrepid machine again as an example, each node on that machine has 2 GB of memory, implying 512 MB per core. If a job requires more than that, allocating the required memory will imply that cores will remain unused. Evidence that this happens is shown in Fig. 2. This depicts the correlation between the requested number of processors and the allocated number. The high values (dark shading) on the main diagonal imply that most jobs indeed get what they requested. But note that high values also appear on a second diagonal where allocations are four times higher than requests. This most probably reflects requests where the required memory forces a full node to be allocated to each process, even though it will use only one of the four available cores.

In the above examples examining the value of a single field immediately showed that the data is problematic. Logs may also include redundant data, that allows for sanity checks by comparing the values in several related fields. For example, the HPC2N log uses the Maui scheduler, which records copious data. In particular, the following fields are included:

Field 2: Nodes Requested ($nodesReq$)

Field 3: Tasks Requested ($tasksReq$)

Field 22: Tasks Allocated ($tasksAlloc$)

Field 23: Required Tasks Per Node ($tasksPerNode$)

Field 32: Dedicated Processors per Task ($procPerTask$)

Field 38: Allocated Host List ($nodesList$)

In principle, it may happen that not all requested tasks are actually allocated, so $tasksAlloc \neq tasksReq$. However, in this log this only happens for 767 jobs, which are 0.14%, so in effect we may take these fields as equal. Likewise, we find that $nodesReq = |nodesList|$ for all but one job. This allows for the following checks:

- Calculate number of nodes based on task requirements as $tasksReq/tasksPerNode$. This turns out not to match the actual number of nodes in 6,428 cases. This is worse than it seems because $nodesReq$ is actually specified in only 89,903 cases (in 437,468 jobs $nodesReq$ is 0, so there is nothing to compare with). Also, in 30,357 jobs $tasksPerNode$ is given as 0, so the check is undefined.
- Compare the number of processors in the allocated nodes (each node has 2 processors) with the number calculated based on task requirements, which is $tasksReq * procsPerTask$ (or $tasksReq * procsPerTask + 1$ in case it is odd). These do not match in 6,250 cases.

When inconsistencies are discovered, one has to decide which of the competing data to use. Oftentimes it is unclear what to do. As a simple example, consider the following record from the SDSC Paragon 1995 log, with had a submit time of 05/27/95 13:59:38, a start time of 05/27/95 13:04:08, and an end time of 05/27/95 14:03:13. The problem here is that the start time is before the submit time, so when calculating the wait and run times the wait is negative. The options of how to handle this are listed in Table 3. Setting the start time to the submit time without changing anything else reduces the runtime from nearly an hour to $3\frac{1}{2}$ minutes, which is a big change. We can also do the opposite, and move the submit time back to the start time. An alternative based on using the $\langle$submit, wait, run$\rangle$ triplet is to just

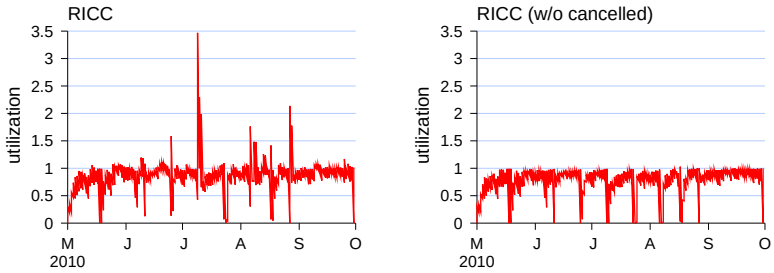

Figure 3: *Strange effect of canceled jobs.*

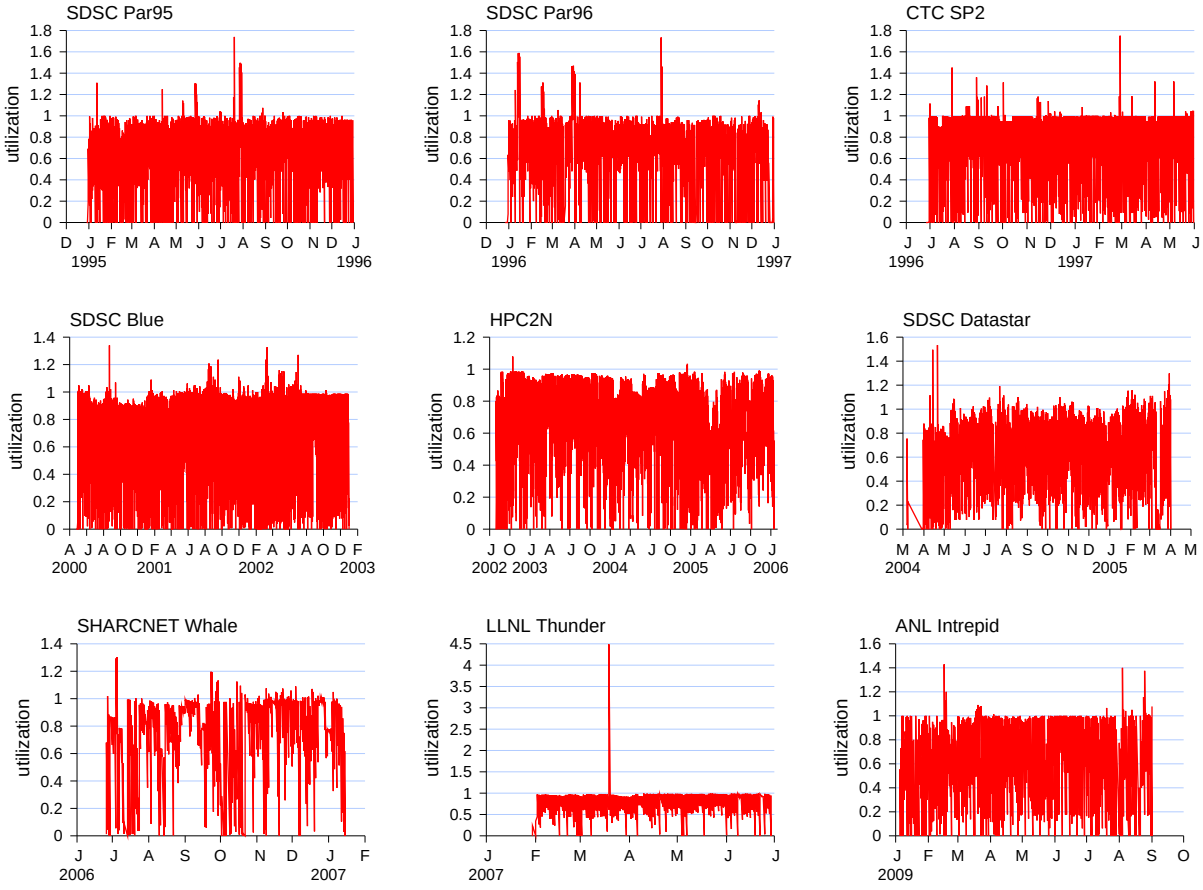

Figure 4: *Examples of utilization exceptions. For each day the range between the minimal and maximal utilizations observed is colored.*

set the wait time to 0. This effectively means setting the start time to the submit time, and changing the end time to maintain the original runtime. Any of these options may or may not reflect what had actually happened in reality.

Another example of such a dilemma is provided by the RICC log. In this log the maximal momentary utilization is erratic, and often surpasses 100%, which should not happen (more on this in the next section). But if we filter out jobs that were marked as canceled, the utilization results are much more reasonable (Fig. 3). This is still troubling, however, because the canceled jobs are in fact recorded as having used time on the processors.

3.3. Erroneous Data

A third type of problem is when the recorded data is downright wrong. For example, the LLNL Atlas and Thunder logs contain jobs with a recorded time limit of 4294967294 seconds. This is most probably the result of mishandling a 32-bit signed value of -2 by placing it in an unsigned variable. In the SDSC Blue log, timestamps are given in human-readable form in the format 2000-12-23–19:52:38. However, when these are tabulated they lead to a daily cycle that peaks from the evening hours to midnight, and achieves a minimum from 10 to 11 AM. This is most probably due to mishandling of UTC timestamps and using the gmtime function rather than the localtime function that corrects for time zones.

In some cases wrong data is the result of intentional misreporting. For example, in the KTH SP2 system the system administrators report that sometimes they have pushed jobs through the FIFO queue by giving them artificially low 'enter-fifo' times. Thus the arrival time and the wait time as recorded in the log are bogus.

A more subtle situation that actually happens in nearly all logs is that the recorded utilization is occasionally greater than 100%, which is technically impossible. To calculate the utilization, one scans the log and simply counts the number of processors in the jobs that are reported as running at each instant (note that if one job terminates and another starts in its place, they should not both be counted). This is then compared to the number of processors in the system to obtain the utilization. The results of performing such calculations are shown in Fig. 4, indicating exceptions that are sometimes large.

There are three possible explanations for such utilization exceptions. The first is timing inconsistency, where one job is recorded as running a few seconds beyond its actual termination, or another is recorded as starting slightly before it actually got hold of the processors. Such situations may be identified and corrected, as shown in Section 4.5 below. The second is that there is no real problem, because a job had released some of its processors before it terminated. While possible in principle, all our logs assume rigid jobs that use all their processors for the duration of the run. As a result we have no data to support this possibility. The third possible explanation is simply a logging error. For example, this is the most likely explanation when the log contains a sequence of several large jobs with very similar parameters that all started on the same second, and together require more processors than are available in the system.

Despite the utilization exceptions, it turns out that all the logs are actually stable in the sense that the service rate is higher than the arrival rate. This is important because it implies that the logs can actually be used for simulations of parallel job scheduling. To verify stability we divided each log into fixed intervals of length T, and counted the fraction of intervals where the work that arrives in the interval cannot be accommodated within the interval. "Accommodation" had two interpretations. The first is just total resource usage by all jobs, meaning that the amount of work arriving within an interval of T was less than the capacity available during this interval. The second is whether they can actually be scheduled by a backfilling scheduler, when jobs are sorted favorably (that is, from longest to shortest or from largest to smallest) and all of them are assumed to arrive at the outset. In both cases, jobs that are longer than T are divided into slices of length T that are assigned to successive intervals.

The results were that in all cases the fraction of unaccommodated intervals went down to zero as T increased (Fig. 5). However, in some cases very long Ts were needed. For example, in the SDSC SP2 log 12% of the weekly intervals were not accommodated, and in the CTC SP2 log 8% of the biweekly intervals suffered the same fate. In both cases this dropped to zero only for intervals of a full month.

A common feature of many utilization graphs is the horizontal upper bound of 100% utilization (albeit sometimes it is breached). A special case of utilization exception is when this upper bound occurs below 100% utilization. This most probably indicates that our information regarding the number of processors is wrong. For example, the size of the CTC SP2 machine was 512 processors, of which 430 were in the batch partition. Assuming this is the number of processors being used leads to the utilization graph shown in Fig. 6, with an upper bound of around 78.38%. This

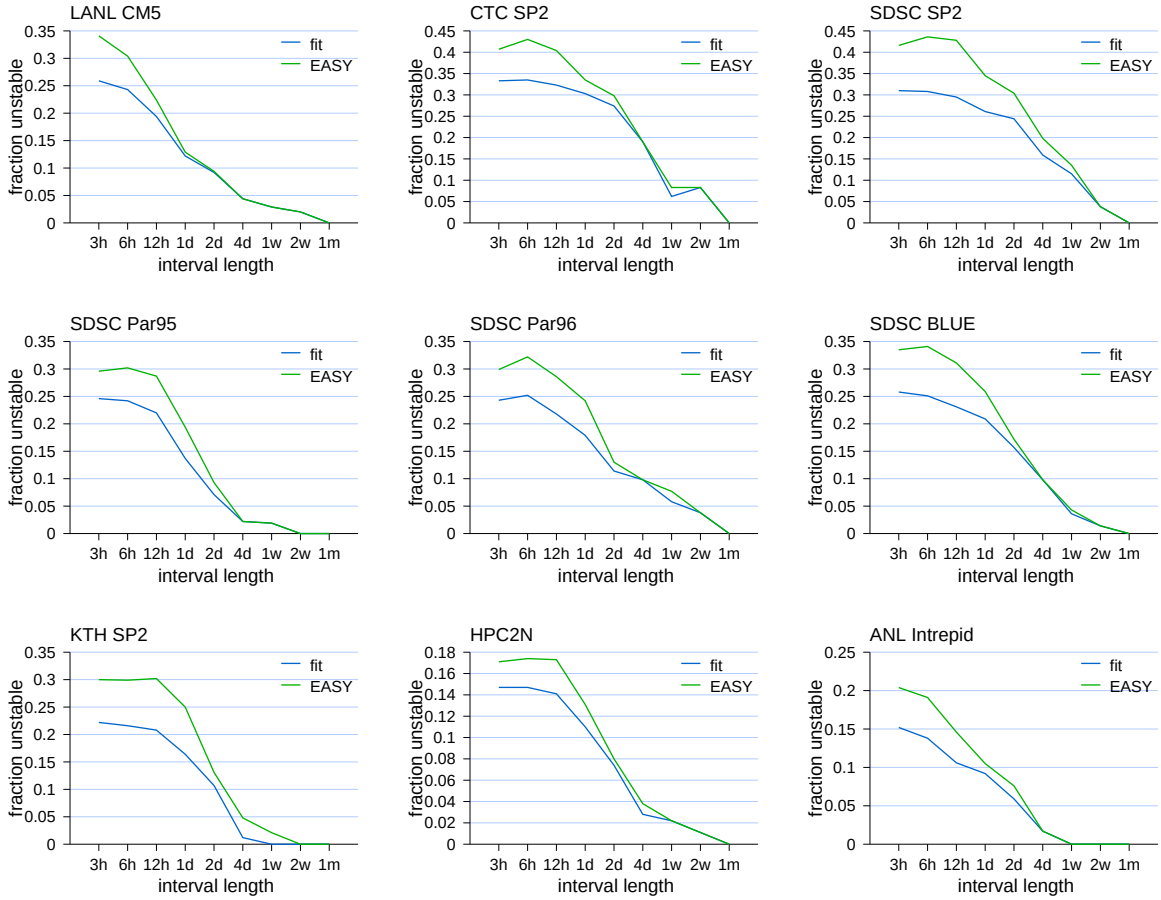

Figure 5: *Stability results of logs that had relatively many unstable intervals. In most other logs only a few percent at most of even the short intervals were unstable.*

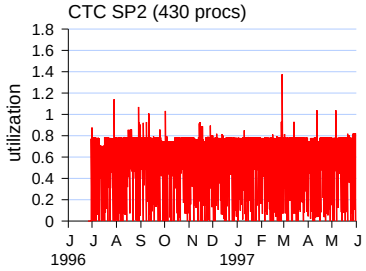

Figure 6: *Effect of assuming the wrong number of processors.*

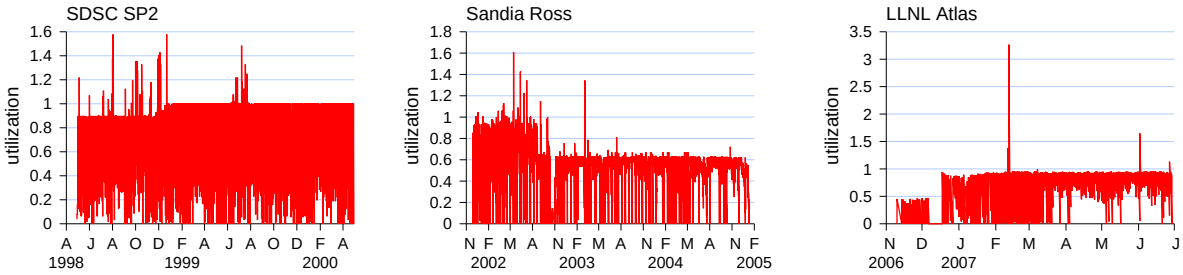

Figure 7: *Examples of utilization variability probably due to configuration change.*

Table 4: *Queues on the SDSC Paragon.*

time	nodes								low
limit	1	4	8	16	32	64	128	256	pri
1 hr		q4s	q8s qf8s	q16s qf16s	q32s qf32s	q64s			
4 hr					q32m qf32m	q64m qf64m	q128m qf128m	q256m qf256m	
12 hr	q1l				q32l qf32l	q64l qf64l	q128l qf128l	q256l qf256l	standby fstandby

indicated that the true size of the batch partition used to capture the log was most probably only 338 processors, and not 430. Subsequent digging in the Internet archive to find old versions of the original web pages describing this machine indicated that the true size may have been 336 processors.

3.4. Environment Variability

A major problem with parallel workload logs is that the configuration of the underlying machine may be heterogeneous and may even change with time. This can be expected to have an effect on the workload, to the point of making it non-stationary. In many cases we do not have information about such effects, but sometimes we can deduce them from the log data.

An important type of variability is apparent changes in system capacity. This is evident from the utilization graphs, as shown in Fig. 7. The SDSC SP2 seems to have grown about a third of the way into the log. In the Sandia Ross machine the available capacity dropped significantly about a quarter of the way into the log. The LLNL Atlas had an initial trial period with half the final capacity. More extreme examples are the LPC cluster, which started with only one node and was later expanded to the full size of 70 nodes, and the CEA Curie machine, which started with one partition containing 11,520 cores and was later expanded with another partition of 80,640 cores. In all these cases, using the whole log consecutively seems to be inappropriate, because it is actually composed of the juxtaposition of two distinct workloads recorded under different conditions.

A more subtle form of variability is the imposition of resource constraints. The scheduling of parallel jobs is often controlled by defining a set of queues with different priorities and resource constraints. Jobs are submitted to the appropriate queue, as a means of specifying their requirements. The scheduler then judiciously selects jobs for execution from the different queues so as to create a "good" job mix that meets the scheduling objectives[2]. This obviously has an effect on the representation of different types of jobs in the log. To confound things, system administrators may change the queue definitions over time.

[2]This is based on the assumption that the jobs are indeed submitted to the "most appropriate" queue, which tightly fits the job's requirements. In retrospect this assumption is naive, and jobs often use only a small fraction of their runtime limit [11, 29].

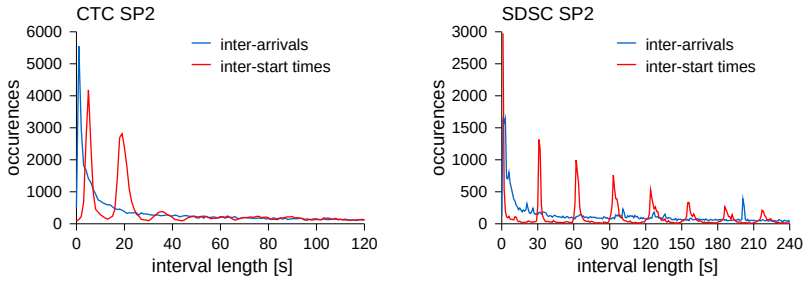

Figure 8: *Examples of modal inter-start-time distributions due to batching by the scheduler.*

For example, the SDSC Paragon system employed the system of queues described in Table 4. The ones with an 'f' indicate use of 32 MB (fat) nodes, while the others are for 16 MB nodes. The scheduler could use different sets of nodes for different queues during prime time and non-prime time (nights and weekends) [44]. Specifically, during prime time it was desirable to provide quick turnaround times for short jobs, so a set of nodes were set aside for such jobs. But despite this richness, the log actually contained quite a few additional queues, including test, interactive, qf32test, q_tmp32, sdsc_test, q1ll, holding, q320m, q4t, and q256s. For some of these we can guess the resource requirements, but for the others we cannot.

A striking example of the effect of such constraints occurred when the scheduler was changed on the LLNL T3D [11] (regrettably, this data is not available on the Archive). When effective gang scheduling was introduced in March 1996 it became much easier to run large jobs. By October the distribution of job sizes had changed, with the fraction of resources devoted to 32-processor jobs dropping by two thirds, while the fraction of resources devoted to 64, 128, and 256-processor jobs more than doubled.

The KTH SP2 system also imposed various limits on job run times (and this was also changed during the period that the log was recorded). In essence jobs were limited to running for up to 4 hours during weekdays, which were defined to be from 7 AM to 4 PM Monday through Friday. At nights they could run for 15 hours, and over the weekend for 60 hours. By tabulating the number of jobs with long requested runtimes that were submitted at different times of the day and the week, one can see that requests to run jobs longer than 4 hour peak every day after 4 PM, and requests to run jobs longer than 15 hours are nearly always submitted on Friday afternoon.

In addition to differences in configuration, schedulers may exhibit idiosyncratic behavior. A small example is the batching of jobs. Some schedulers accumulate jobs across short intervals, rather than immediately scheduling jobs as they arrive. This leads to a modal inter-start-time distribution, as opposed to a smoother inter-arrival distribution, as demonstrated in Fig. 8.

The point of these examples is to demonstrate that the observed workload is not necessarily a "natural" workload that reflects what the users want to run. Rather, users may mold their requirements according to the limitations imposed by each system's administrators and schedulers. And to make matters worse, these limitations may be quite involved, may change unpredictably, and may be unknown to us.

3.5. Non-Representative Behavior

Another source of variability is the users themselves. In quite a few cases we find users whose behavior is different from the behavior of all others, and might be considered to taint the log data.

An early example was the behavior of the system administrators on the NASA iPSC machine. It turns out that these administrators commonly ran the Unix pwd command (print working directory) on a single node of the machine as a means to verify that the system was operational and responsive. All told, no less than 56.8% of the jobs recorded in the log were such pwd commands. Another example comes from the SDSC Paragon log, where an automatic script was executed every day at around 3:45 AM, most probably to perform a sequence of cleanup or maintenance operations. This caused a noticeable perturbation of the normal daily cycle of activity.

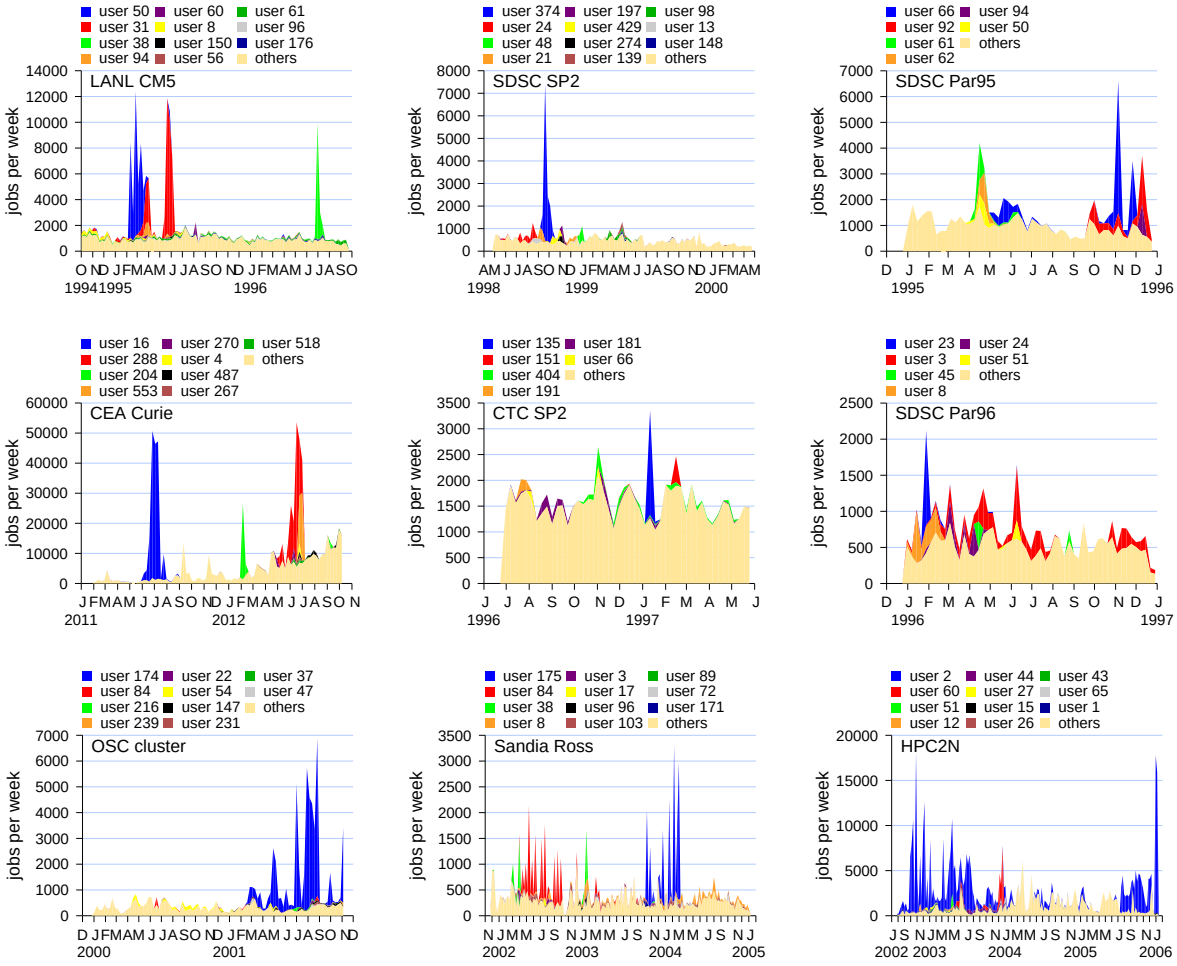

Figure 9: *Examples of large flurries of activity by individual users.*

Another type of non-representative behavior is flurries of activity by individual users, which dominate all other activity in the system [14, 42]. Examples are shown in Fig. 9. To create these graphs, the number of jobs in each week was counted and the weeks with the highest level of activity singled out. Then, the top users in these weeks were identified and their activity color-coded throughout the log. Here we focus on job flurries, but in logs from parallel machines like ours flurry observations can also be based on processes. Importantly, process flurries are not necessarily correlated with job flurries, as they can be created by a relatively small number of jobs that each include a very large number of processes. Examples of logs that contain process flurries that do not correspond to job flurries include SDSC SP2, SDSC Blue, HPC2N, SHARCNET, LLNL Atlas, LLNL Thunder, and RICC.

Flurries can be roughly classified into three types.

- Sporadic large flurries, where the number of jobs produced by a single user is 5–10 times the average weekly total, but this continues only for a short period. A prominent example is the activity of user 374 in the SDSC SP2 log, or the three large flurries in the LANL CM5 log. Note that these are not necessarily the most active users in the log, but their concentration makes them unique.
- Long-range dominance, where the abnormal level of activity by a single user continues for a long time, and

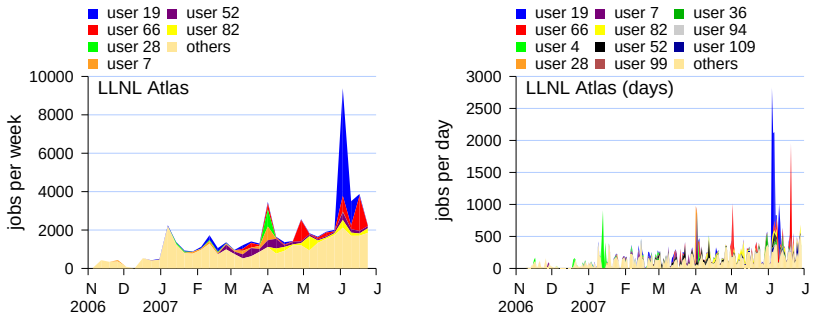

Figure 10: *Flurries observed at different resolutions.*

perhaps even dominates the whole log. A striking example is the activity of user 2 in the HPC2N log, who is responsible for no less than 57.8% of the whole log.

- Small flurries, where some user displays a relatively high level of activity, but not as exceptional as the previous classes. Nevertheless, even such small flurries may cause instabilities in simulations used to evaluate schedulers. An example is the flurry in the CTC SP2 log [14].

While the large-scale flurries pop out and are obviously behavioral outliers, the identification of small flurries is more contentious. There seems to be no precise rule for deciding when a user's activity is abnormal, and when it is just the most active from among a distribution of users. Moreover, the degree that a user's activity appears to be abnormal may depend on the resolution of observation. For example, when using a daily resolution flurries may look more prominent than when using a weekly resolution (Fig. 10). In the Parallel Workloads Archive we attempt to be conservative, and flag only flurries that look prominent on a weekly scale. However, smaller flurries may also be flagged if we know that they lead to problems in simulations.

Other patterns are even more subtle than small flurries, but nevertheless may be important. For example, a study of the interactions between workloads and system schedulers found that the CTC SP2 log is unique in having many serial jobs that are relatively very long [9]. This was attributed to the fact that this machine inherited the workload of an IBM ES/9000 mainframe that was decommissioned at the same site. Importantly, this arcane attribute of the workload actually turned out to influence performance results in the context of simulations of scheduling with backfilling [9]. Thus knowing about it may be a consideration when deciding whether or not to use this workload in an evaluation. In a related vein, most parallel workloads exhibit a weak positive correlation between the parallelism and runtime of jobs, but the LANL O2K log exhibits a weakly negative correlation. This can be important in situations where the correlation between job size and runtime affects performance [27].

Another strange workload attribute is the user residence pattern in the SDSC Blue log[3]. In most logs, many new users are observed in the first few weeks (these are the users who were actively using the system when the logging commenced). Then new user arrivals stabilize at a lower rate. The opposite happens with the users' last appearances in the logs: initially they are randomly distributed, and towards the end of the log one finds a large concentration. But the SDSC Blue log exhibits a different and strange pattern. This log is 32 months long, and includes data about 467 users. Surprisingly, the first user to leave does so only after 248 days (more than 8 months). By this time no less than 307 different users had been observed, and all of them continue to be active. Moreover, only 10 users leave within the first 20 months. Of the remaining 457 users 106 leave during the last month, and the other 351 leave during the period from the 21st month to the 31st month, at an average rate of 32 per month. While we currently do not know of any consequences of this strange pattern, it nevertheless remains highly unusual.

[3]This observation is due to Netanel Zakay.

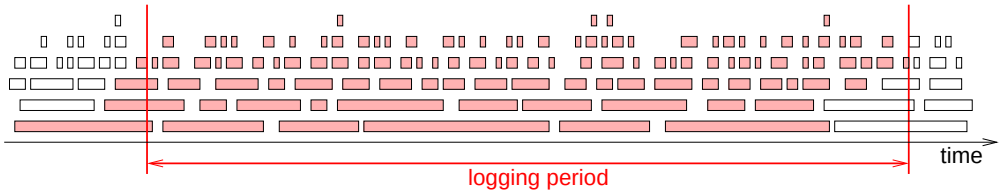

Figure 11: *Example of sampling effects at the ends of the logging period.*

3.6. End Effects

The way that most logs are collected is that the record describing each job is written when the job terminates. If jobs are extremely short this has no appreciable effect. But if jobs can be very long, as is the case for parallel jobs executed on large supercomputers, this can have a marked effect on the observed workload at the log's ends, and on the calculated utilization. This assumes that the machine is in production use, and logging is done for an arbitrary limited duration.

At the beginning of the log we often see a warmup effect. This is because the first timestamp in the log is typically the arrival time of a job that had a very long response time and terminated soon after logging commenced. Jobs that ran in parallel to this job but had shorter response times were not logged, because they terminated before logging commenced. Hence the logged load is smaller than it really was in the initial portion of the log (Fig. 11).

The opposite effect happens near the end of the log, where only short jobs get logged. Jobs with longer response times that start towards the end of the logging period may not terminate within the logging period, and hence are not logged. Again, the effect is of logging only part of the load that was actually present.

To counteract these effects, care must be taken. When calculating a machine's utilization, one usually calculates the total resource usage (processors$\times$time) of all jobs, and divides this by the available resources (totalProcessors$\times$logDuration). To reduce the end effects, it is best to interpret the log duration as the interval from the first termination to the last termination, rather than as the interval from the first timestamp to the last timestamp or the interval from the first arrival to the last arrival.

When performing a simulation using a log, it is important to discard some initial subset of the results in order to allow for warmup. Also, stop measuring when the last arrival occurs, because after that time the simulated jobs will encounter less and less competition, leading to unrealistically good results.

3.7. Missing Downtime Data

The activity on a parallel supercomputer may be interrupted occasionally due to various reasons, such as scheduled maintenance, software failures, and hardware failures. Obviously this affects the logged workload, and creates time intervals where the utilization drops to zero. Jobs may be truncated and re-submitted later. As a side effect, this also distorts overall utilization calculations.

Failure data is also directly important for performance evaluations. Failures may reduce observed performance as jobs need to wait for resources to become available [22]. Their existence also suggests the system-level metric of how many jobs were killed due to failures. Conversely, job data may help in analyzing failures and producing reliable data regarding the severity and effect of failures [47].

Failure data exists for a few of the systems in the Parallel Workloads Archive. Examples include the NASA iPSC, SDSC Paragon, LPC grid, the MetaCentrum grid [22], and ANL Intrepid [47]. However, this data is not integrated into the standard workload format. Note that a separate repository concerning failure data exists at www.usenix.org/cfdr, and in the future it may be beneficial to create some connections between this repository and our archive. Another related repository is the Failure Trace Archive at http://fta.scem.uws.edu.au, which has made inroads toward defining a standard format for recording failure data.

4. Attempting to Improve Log Data Quality

An important goal of the archive is to capture experience with using the logs. This is done by providing improved or specially "cleaned" versions of the logs which reflect our experience. Such versions allow users of the data to benefit from our experience without delving into all the details and cleaning decisions themselves, and also ensure that different users use data that was cleaned in the same way. Needless to say, users are also free to inspect the original data for themselves and make other decisions.

4.1. General Procedure

Importantly, we have found that cleaning operations developed for one log are often applicable to others too. Thus whenever a new log is accepted into the archive it undergoes a battery of tests for data quality.

Some of the data quality improvements are applied automatically as part of our conversion to the standard workload format. These include the following:

- Some attempts are made to recover missing timing data based on redundant or related fields, as described in Section 4.3. Negative wait and run times that are larger than 1 hour and 5 minutes (to allow for possible changes in daylight saving time and for clock drifts) are then changed to -1 (which means the value is unknown), while smaller negative values are simply changed to 0.
- Fields that should be positive (e.g. number of processors, memory usage, and requested runtime) but are recorded as having a 0 value are changed to -1.
- Users, groups, and applications are anonymized by replacing them with serial numbers in order of first appearance.

In other cases visual summaries of the data are prepared, like the arrival graphs of Fig. 9 or the capacity graphs of Fig. 4. These can then be checked by us to determine whether any cleaning is needed to remove data that we feel should not be used because it is erroneous or not representative of normal production use.

The cleaning itself is performed by a script that can handle the following specifications of which jobs to remove from a given log:

- Jobs with a specific field value. For example, this is useful to remove all the jobs submitted by a certain user.
- A specified span of jobs. When combined with a user, this can be used to specify a flurry but leave the user's other non-flurry activity intact.
- Jobs within a specified time span. The span can be one sided, so an initial prefix of the log can be specified.
- Jobs running at specific times each day. When combined with a user this can be used to remove automatic jobs that are fired each day.

These specifications can be combined using Boolean AND, OR, and NEGATION.

4.2. Removing Initial Low-Load Intervals

Some of the logs were started when the machines being logged were very new, before users started to use them for real. As a result they have initial segments that do not reflect real production use but system testing, often interspersed with long idle periods. Such initial segments are typically of no interest for system evaluations (albeit they might be of interest for studies of how the workload evolves [20]). In other logs the initial part of the log reflects a partial configuration, which is later completed to the full configuration of the machine. In the interest of providing data that can be used as-is, we shorten the logs and remove the initial low-load periods from the cleaned versions.

An example is the LLNL Atlas log. As shown in the utilization plot in Fig. 7, this log has an initial segment from 10 November 2006 to 7 December 2006 where the utilization is up to 50%, indicating that most probably the machine was operating at half capacity. Then there is a short interval with no activity, and finally full production work is started on 18 December 2006. In the cleaned version the log is shortened and everything before 18 December 2006 is removed.

Table 5: *Calculation of job timing data based on available input data.*

arr	$start$	end	action
OK	*	*	$ARR = arr$
	OK	*	$WAIT = start - arr$
		OK	$(run)?\ RUN = run : RUN = end - start$
		n/a	$(run)?\ RUN = run :$ $(cpu)?\ RUN = cpu$
	n/a	*	$(run)?\ RUN = run :$ $(cpu)?\ RUN = cpu$
		OK	$(run)?\ WAIT = (end - run) - arr :$ $(cpu)?\ WAIT = (end - cpu) - arr :$ $(succ)?\ RUN = end - arr, WAIT = 0 :$ $WAIT = end - arr$
n/a	OK	*	$ARR = start$
		OK	$(run)?\ RUN = run : RUN = end - start$
		n/a	$(run)?\ RUN = run :$ $(cpu)?\ RUN = cpu$
	n/a	*	$(run)?\ RUN = run :$ $(cpu)?\ RUN = cpu$
		OK	$(run)?\ ARR = end - run :$ $(cpu)?\ ARR = end - cpu : ARR = end$

The notation "(X)? S1 : S2" means that if input X is available or true then action S1 is taken, otherwise action S2 is taken. Note that these may be stringed to form "else if" sequences. $succ$ means the job has a success status. Note that in some combinations of unavailable inputs some of the desired outputs are left undefined.

An extreme case of unrepresentative data is the LLNL uBGL workload log. In this log of 112,611 jobs, 101,331 are recorded as failed, and in fact the vast majority (99,401) did so within 5 seconds. This is attributed to the fact that the machine was new and unstable at the time this log was recorded. As a result, the whole log is actually unrepresentative of production use.

4.3. Reconstructing Missing Data

Among the most important attributes of parallel jobs are their arrival time and running time. Regrettably, in some cases this information or related information (e.g. the start time or end time) are missing. Nevertheless, sometimes missing data can be reconstructed at least partially.

Our conversion scripts accept the following partly redundant fields that all relate to job timing:

- Arrival time (arr)
- Start time ($start$)
- End time (end)
- Running time (wallclock, run)
- CPU time (average per processor, cpu)

If the data is available and consistent, we should have $arr \leq start \leq end$, $run = end - start$, and $cpu \leq run$.

The output of the conversion needs the following three non-redundant fields:

- Arrival time (ARR)
- Wait time ($WAIT$)

- Running time (RUN)

The way these are set based on the available input data is given in Table 5. This reflects various heuristics to automatically recover as much data as possible in those cases that explicit data is erroneous or unavailable. For example, if $start$ is missing, we assume it to be arr. If run and cpu are also not available, we can then estimate the runtime as $end - arr$. However, this should be qualified by job status. If the job was canceled before it was started, it is more appropriate to assign this interval to the wait time, and leave the runtime undefined.

While such heuristics may recover some data and enhance the usability of the log, they may also cause problems. For example, in the SDSC SP2 log, a straightforward analysis revealed that 4291 jobs got more runtime than they requested, and in 463 cases the extra runtime was larger than 1 minute. However, 5831 jobs had undefined start times, so their runtime was not computed. When the missing start times were replaced by the submit times, the number of jobs that got more runtime than they requested jumped up to 6154, and in 2284 of them the difference was larger than 1 minute. As we saw previously, there is no way to know what the correct data was. We need to make a subjective decision based on the data that is available.

4.4. Data Cleaning by Removing Flurries

The anomalous behaviors described in Section 3.5 degrade data quality because they are anomalous and do not represent normal usage. Using logs that contain such anomalies as input to evaluations risks results that are tainted by the anomalies. For example, if a log contains voluminous non-representative activity by a single user, and this is used to evaluate schedulers and suggest operational parameters, we risk making the selection so as to optimize the performance of a non-representative user that was active on a single system some years ago. Flurries may also cause evaluations to be overly sensitive to details. In previous work we have identified and described a situation where the average slowdown of all the jobs in the simulation changed by 8% after a miniscule change to a single job (changing its runtime from 18 hours and 30 seconds to 18 hours flat). The sensitivity was attributed to a flurry of jobs submitted later that were affected en-masse [42].

Of course, removing the abnormal behavior also entails risk. First, maybe we are wrong and the data is not as bad as we think. Second, by removing part of the data we are left with a log that does not give the full picture. In particular, the behavior of other users may have been affected by the load placed on the system by the abnormal user. Therefore the cleaning process deserves careful attention, and we have considered the following cleaning options [41]:

- Avoid the issue by using only short samples of representative jobs. This approach is motivated by computer architecture studies, where the execution of complete benchmarks is often substituted with the execution of representative slices (e.g. [24, 35]). The motivation there is that simulating the billions of instructions in complete benchmarks is extremely time consuming, so settling for slices is worthwhile. In parallel job scheduling we typically do not have that problem, and logs contain no more than several hundreds of thousands of jobs. It is therefore better to use all the data, and avoid the debate over what subsets of jobs are "representative".
- Remove complete days as was suggested in [4]. This removes not only the anomalous data but also contemporaneous data that may have been affected by it, at the cost of leaving a gap in the log. But this may be perfectly OK, because the effect of flurries is typically limited in time. For example, in the sensitivity example noted above, the effect was due to a 10-hour flurry on day 581 of a 730-day log. Removing this flurry affected subsequent simulation behavior for 5 days: simulations with and without the flurry were identical except for days 581 through 585. So removing these days leaves use with 725 days of undisputed valid data. However, identifying exactly how far the effect of the flurry extends is difficult, and may also be context dependent.
- Remove tainted results from the final analysis, rather than removing jobs from the input. Separating the input jobs into classes and reporting performance results for individual classes has been used in various situations (e.g. [33, 9]). Thus we can simply compute our performance indicators based on only the "good" jobs, and exclude the flurry jobs from the average. However, this faces two risks. First, the mere presence of the flurry jobs may affect the performance experienced by other jobs that ran at the same time. Second, we need to develop a methodology to identify the problematic jobs that should be excluded, and trust analysts to incorporate it into their evaluation frameworks and use it correctly and consistently.

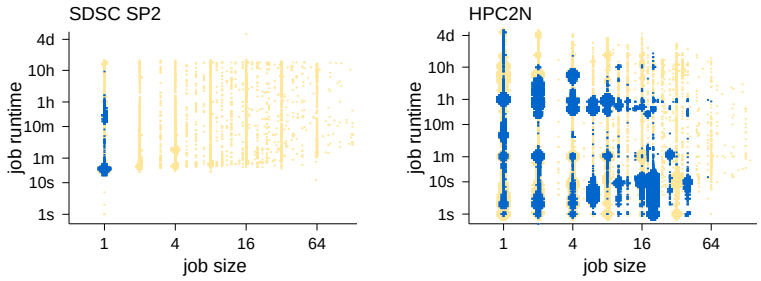

Figure 12: *Scatter plots showing size/runtime data for a whole log, and highlighting jobs of a single highly active user.*

- Remove only the flurry jobs, as we suggested in [14, 42]. In cases we checked, this turns out to have practically the same effect as excluding the flurry jobs from the final averages, but it is safer because it is simpler and cannot be misused.

The policy adopted in the Parallel Workloads Archive is to remove the most prominent dominant users and flurries, but at the same time also provide the original log as is. By using our cleaned logs, analysts can tap into our experience and avoid the need to make cleaning decisions for themselves. This also has the benefit that different analyses will be comparable by virtue of using exactly the same data. On the other hand, if they do indeed want to invest the time in studying anomalous data and deciding what to do about it, this is possible.

About half the logs in the archive have cleaned versions. In further support of cleaning, we note that in most cases the impact on the log is minimal. For example, in the SDSC SP2 log, removing the flurry of activity by user 374 reduced the overall utilization only from 83.7% to 83.5%. The reason is that all the jobs by this user were both small (using only a single processor) and short (typically lasting for less than a minute, but with some lasting up to an hour, as indicated in Fig. 12). The most extreme case was the HPC2N log, where user 2 was responsible for a full 57.8% of the jobs in the log. However, removing them only reduced the load from 70.2% to 60.2%. Again, these jobs tended to be small (up to 8 processors) or short (up to a minute), albeit in some cases they were larger (e.g. 20 processors) or longer (e.g. an hour).

4.5. Enforcing the Capacity Constraint

The errors mentioned in Section 3.3 whereby the utilization exceeds 100% may be reduced by two means. The first is "shaking" the input, namely making small modifications to job start times such that the jobs will fit in [43]. Specifically, we used a linear solver to see whether all jobs could be accommodated if we increase some of the wait times by different amounts. However, this invariably led to either of two outcomes: either a proof that no solution could be found within the specified limits (e.g. only change wait times by up to 1 hour), or failure of the linear solver to terminate within a few hours.

The second option is to simply delete the offending jobs. In order to find which jobs to delete, we first divide the log into cliques of jobs that overlap in time [2]. For those cliques where a utilization exception occurs, we solve a linear program that describes the problem (which jobs were not deleted and the capacity constraint). The optimization criterion is to minimize the number of jobs that are removed, or alternatively the total node-seconds that are removed. This enables a tradeoff between removing a few large jobs or many small jobs. We settle the tradeoff by choosing the approach that leads to the minimal maximal reduction. For example, if removing few large jobs leads to a reduction of LJ percent of the jobs and LU percent of the utilization, while removing many small jobs leads to a reduction of SJ percent of the jobs and SU percent of the utilization, we will choose the first option if $\max(LJ, LU) < \max(SJ, SU)$, and the second otherwise.

Scanning the logs, we find that in some cases very many jobs are involved, and trying to eliminate all the utilization errors would mean removing lots of jobs throughout the log. We therefore decided to leave such logs as they are. But in about half of the logs the utilization errors can be cleaned by removing only a small fraction of the jobs. In these cases using the utilization criterion typically leads to smaller maximal impact. In most cases up to 1 or 2 percent of

Table 6: *Applicability of data quality dimensions to the Parallel Workloads Archive.*

1	accuracy	some problems occur as described in this paper
2	consistency	some internal (among fields in the same log) and external (among similar fields in different logs) inconsistencies occur
3	security	free access is a goal; privacy is maintained by encoding users, groups, and applications
4	timeliness	some logs are dated, but enable research about workload evolution
5	completeness	some desirable data is missing, e.g. job dependencies, memory and I/O requirements, other scheduling constraints
6	conciseness	log files are typically small enough to be easily handled
7	reliability	some problems occur as described in this paper
8	accessibility	freely accessible via the world-wide web
9	availability	freely accessible via the world-wide web
10	objectivity	logs come from different locations and machine types with no biased selection
11	relevancy	extremely relevant as witnessed by extensive use
12	usability	simple format; ASCII files
13	understandability	simple format; documentation of format and background on each log are provided
14	amount of data	seems to be adequate for common usage scenarios
15	believability	data comes from large scale production systems; non-representative behavior is cleaned
16	navigability	table listing logs and their main attributes is provided
17	reputation	data comes from major installations
18	usefulness	witnessed by extensive use
19	efficiency	A year's activity can typically be simulated in seconds
20	value-added	data provides needed grounding in reality

the jobs and utilization need to be removed, and in one case nearly 5 percent. At the time of writing, actually doing this is ongoing work.

5. Conclusions

Even in the age of information overload, good data is a precious and scarce resource. This is especially true in Computer Science, for two reasons. The first is that this field does not have a tradition of experimental research based on empirical observations. The second is the rapid progress in computing technology, which creates the risk that data will be outdated and irrelevant not long after it is collected. Nevertheless, we contend that using real data is still generally preferable over using baseless assumptions. Collecting data and subjecting it to analysis and sanity checks is a crucial part of scientific progress.

Aging is but one aspect of a more general problem, namely the problem of data quality. Thus data should be used intelligently, and experience regarding the cleaning of data and its validity constraints should be recorded and maintained together with the data itself [37]. In the Parallel Workloads Archive, some of the logs have been publicly available for over a decade. Nevertheless, we still occasionally find new and previously unknown artifacts or deficiencies in them. It is unreasonable to expect each user of the data to be able to analyze this independently and achieve comprehensive results. Thus sharing experience is no less important than sharing the data in the first place.

It is interesting to compare our work with work done on data quality in other domains. Knight and Burn have reviewed the commonly cited dimensions of data quality, based on the pioneering work of Wang and Strong and others [23, 39, 45]. Table 6 shows how these dimensions apply to the Parallel Workloads Archive. It turns out that the data itself inherently satisfies some of the dimensions, for example relevance, believability, and value-added. Furthermore, the archive naturally addresses many additional dimensions, for example by making the data available and accessible. The Standard Workload Format that is used also helps, for example by providing privacy and understandability. But

other dimensions are indeed problematic. Specifically, the bulk of this paper was devoted to the description of various accuracy and inconsistency problems. Completeness is another potential problem.

In many cases the decisions regarding how to handle problematic data are subjective in nature. This is of course an undesirable situation. However, it seems to be unavoidable, because the information required in order to make more informed decisions is unavailable. The alternative of leaving the data as is is no better, because the question of how to handle the data arose due to problems in the data itself. Therefore we contend that the best solution is to make the best subjective decision that one can, and document this decision. Doing so in the Parallel Workloads Archive leads to two desirable outcomes. First, users of the data will all be using the same improved version, rather than having multiple competing and inconsistent versions. Second, this can be used as the basis for additional research on methods and implications of handling problematic data.

A further improvement in the usability of workload data may be gained by combining filtering with workload modeling. Specifically, in recent work we considered the concept of workload re-sampling at the user level [46]. This means that the workload log is partitioned into independent job streams by the individual users. These job streams are then combined in randomized ways to generate new workloads for use in performance evaluation. Among other benefits, this approach allows for the removal of users who exhibit non-representative behavior such as the workload flurries of Section 3.5. The reconstructed workloads will also not suffer from underlying configuration changes such as those noted in Section 3.4.

Additional future work concerns data cleaning. One important outstanding issue is how to handle situations where the utilization exceeds 100%, as demonstrated in Section 3.3. As noted in Section 4.5, in about half of the logs we did not find a simple fix to this problem. Another interesting question is to assess the effect of the different problems we found in workload logs. This would enable an identification of the most important problems, which are the ones that cause the biggest effect and therefore justify increased efforts to understand their sources and how to fix them.

Acknowledgments

Many thanks are due to all those who spent their time collecting the data and preparing it for dissemination. In particular, we thank the following for the workload data they graciously provided:

- Bill Nitzberg for the NASA iPSC log
- Curt Canada for the LANL CM5 log
- Reagan Moore and Allen Downey for the SDSC Paragon logs
- Dan Dwyer and Steve Hotovy for the CTC SP2 log
- Lars Malinowsky for the KTH SP2 log
- Victor Hazlewood for the SDSC SP2 and SDSC Datastar logs
- Fabrizio Petrini for the LANL Origin 2000 log
- David Jackson for the OSC cluster log
- Travis Earheart and Nancy Wilkins-Diehr for the SDSC Blue Horizon log
- Jon Stearley for the Sandia Ross log
- Ake Sandgren and Michael Jack for the HPC2N log
- John Morton and Clayton Chrusch for the SHARCNET log
- Moe Jette for the uBGL, Atlas, and Thunder logs from LLNL
- Susan Coghlan, Narayan Desai, and Wei Tang for the ANL Intrepid log
- Dalibor Klusáček and Czech National Grid Infrastructure MetaCentrum for the MetaCentrum log
- Ciaron Linstead for the PIK IPLEX log
- Motoyoshi Kurokawa for the RICC log
- Joseph Emeras for the CEA Curie log

Likewise, many thanks are due to the managers who approved the release of the data. Thanks are also due to students who have helped in converting file formats and maintaining the archive.

Our research on parallel workloads has been supported by the Israel Science Foundation (grants no. 219/99 and 167/03) and by the Ministry of Science and Technology, Israel.

References

[1] 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.

[2] M. Aronsson, M. Bohlin, and P. Kreuger, *Mixed integer-linear formulations of cumulative scheduling constraints - A comparative study*. SICS Report 2399, Swedish Institute of Computer Science, Oct 2007. URL http://soda.swedish-ict.se/2399/.

[3] 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.

[4] 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.

[5] J. Emeras, *Workload Traces Analysis and Replay in Large Scale Distributed Systems*. Ph.D. thesis, Grenoble University, Oct 2013.

[6] W. Fan, F. Geerts, and X. Jia, "*A revival of integrity constraints for data cleaning*". *Proc. VLDB Endowment* **1(2)**, pp. 1522–1523, Aug 2008.

[7] 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.

[8] 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.

[9] 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.

[10] D. G. Feitelson, "*Experimental computer science: The need for a cultural change*". URL http://www.cs.huji.ac.il/~feit/papers/exp05.pdf, 2005.

[11] 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.

[12] 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.

[13] 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.

[14] 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.

[15] 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.

[16] C. Firth, "*Data quality in practice: Experience from the frontline*". In *Intl. Conf. Information Quality*, Oct 1996.

[17] C. W. Fisher and B. R. Kingma, "*Criticality of data quality as exemplified in two disasters*". *Information & Management* **39(2)**, pp. 109–116, Dec 2001, DOI:10.1016/S0378-7206(01)00083-0.

[18] D. Gray, D. Bowes, N. Davey, Y. Sun, and B. Christianson, "*The misuse of the NASA metrics data program data sets for automated software defect prediction*". In 15th *Evaluation & Assessment in Softw. Eng.*, pp. 96–103, Apr 2011, DOI:10.1049/ic.2011.0012.

[19] C. Harger et al., "*The genome sequence database (GSDB): Improving data quality and data access*". *Nucleic Acids Research* **26(1)**, pp. 21–26, Jan 1998, DOI:10.1093/nar/26.1.21.

[20] 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.

[21] 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.

[22] D. Klusáček and H. Rudová, "*The importance of complete data sets for job scheduling simulations*". In *Job Scheduling Strategies for Parallel Processing*, E. Frachtenberg and U. Schwiegelshohn (eds.), pp. 132–153, Springer-Verlag, 2010, DOI:10.1007/978-3-642-16505-4_8. Lect. Notes Comput. Sci. vol. 6253.

[23] S.-a. Knight and J. Burn, "*Developing a framework for assessing information quality on the world wide web*". *Informing Science J.* **8**, pp. 159–172, 2005.

[24] 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.

[25] W. Leinberger, G. Karypis, and V. Kumar, "*Multi-capacity bin packing algorithms with applications to job scheduling under multiple constraints*". In *Intl. Conf. Parallel Processing*, pp. 404–412, Sep 1999.

[26] D. Lichtnow et al., "*Using metadata and web metrics to create a ranking of genomic databases*". In *IADIS Intl. Conf. WWW/Internet*, pp. 253–260, Nov 2011.

[27] 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.

[28] S. E. Madnick, R. Y. Wang, Y. W. Lee, and H. Zhu, "*Overview and framework for data and information quality research*". *ACM J. Data & Inf. Quality* **1(1)**, Jun 2009, DOI:10.1145/1515693.1516680.

[29] 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.

[30] H. Müller, F. Naumann, and J.-C. Freytag, "*Data quality in genome databases*". In 8th *Intl. Conf. Information Quality*, pp. 269–284, Nov 2003.

[31] T. C. Redman, "*The impact of poor data quality on the typical enterprise*". *Comm. ACM* **41(2)**, pp. 79–82, Feb 1998, DOI:10.1145/269012.269025.

[32] 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.*, Oct 2012, DOI:10.1145/2391229.2391236.

[33] 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.

[34] M. Shepperd, Q. Song, Z. Sun, and C. Mair, "*Data quality: Some comments on the NASA software defect datasets*". *IEEE Trans. Softw. Eng.* **39(9)**, pp. 1208–1215, Sep 2013, DOI:10.1109/TSE.2013.11.

[35] 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.

[36] E. Shmueli and D. G. Feitelson, "*Backfilling with lookahead to optimize the packing of parallel jobs*". *J. Parallel & Distributed Comput.* **65(9)**, pp. 1090–1107, Sep 2005, DOI:10.1016/j.jpdc.2005.05.003.

[37] Y. L. Simmhan, B. Plale, and D. Gannon, "*A survey of data provenance in e-science*". *SIGMOD Record* **34(3)**, pp. 31–36, Sep 2005, DOI:10.1145/1084805.1084812.

[38] A. J. Smith, "*Workloads (creation and use)*". *Comm. ACM* **50(11)**, pp. 45–50, Nov 2007, DOI: 10.1145/1297797.1297821.

[39] D. M. Strong, Y. W. Lee, and R. Y. Wang, "*Data quality in context*". *Comm. ACM* **40(5)**, pp. 103–110, May 1997, DOI:10.1145/253769.253804.

[40] Thinking Machines Corp., *Connection Machine CM-5 Technical Summary*. Nov 1992.

[41] D. Tsafrir, *Modeling, Evaluating, and Improving the Performance of Supercomputer Scheduling*. Ph.D. thesis, Hebrew University, Sep 2006.

[42] 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.

[43] D. Tsafrir, K. Ouaknine, and D. G. Feitelson, "*Reducing performance evaluation sensitivity and variability by input shaking*". In 15th *Modeling, Anal. & Simulation of Comput. & Telecomm. Syst.*, pp. 231–237, Oct 2007, DOI:10.1109/MASCOTS.2007.58.

[44] 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.

[45] R. Y. Wang and D. M. Strong, "*Beyond accuracy: What data quality means to data consumers*". *J. Management Inf. syst.* **12(4)**, pp. 5–33, Spring 1996.

[46] N. Zakay and D. G. Feitelson, "*Workload resampling for performance evaluation of parallel job schedulers*". *Concurrency & Computation — Pract. & Exp.* 2014, DOI:10.1002/cpe.3240. To appear.

[47] Z. Zheng, L. Yu, W. Tang, Z. Lan, R. Gupta, N. Desai, S. Coghlan, and D. Buettner, "*Co-analysis of RAS log and job log on Blue Gene/P*". In *Intl. Parallel & Distributed Processing Symp.*, pp. 840–851, May 2011, DOI:10.1109/IPDPS.2011.83.

[48] B. B. Zhou, C. W. Johnson, D. Walsh, and R. P. Brent, "*Job packing and re-packing schemes for enhancing the performance of gang scheduling*". In *Job Scheduling Strategies for Parallel Processing*, pp. 129–143, Springer-Verlag, 1999, DOI:10.1007/3-540-47954-6_7. Lect. Notes Comput. Sci. vol. 1659.