Instability in Parallel Job Scheduling Simulation: The Role of Workload Flurries

Dan Tsafrir Dror G. Feitelson

School of Computer Science and Engineering
The Hebrew University, Jerusalem 91904, Israel
{dants,feit}@cs.huji.ac.il

Abstract

The performance of computer systems depends, among other things, on the workload. This motivates the use of real workloads (as recorded in activity logs) to drive simulations of new designs. Unfortunately, real workloads may contain various anomalies that contaminate the data. A previously unrecognized type of anomaly is workload flurries: rare surges of activity with a repetitive nature, caused by a single user, that dominate the workload for a relatively short period. We find that long workloads often include at least one such event. We show that in the context of parallel job scheduling these events can have a significant effect on performance evaluation results, e.g. a very small perturbation of the simulation conditions might lead to a large and disproportional change in the outcome. This instability is due to jobs in the flurry being effected in unison, a consequence of the flurry's repetitive nature. We therefore advocate that flurries be filtered out before the workload is used, in order to achieve stable and more reliable evaluation results (analogously to the removal of outliers in statistical analysis). At the same time, we note that more research is needed on the possible effects of flurries.

1 Introduction

The performance of a computer system depends not only on its design and implementation, but also on the workload to which it is subjected [9]. Different workloads may lead to different absolute performance numbers, and in some cases to different relative ranking of systems or designs. Using representative workloads is therefore crucial in order to obtain reliable performance evaluation results.

One way to obtain representative workloads is to use real workloads from productions systems. One can record the workload on an existing system, and play back the recording to drive a simulation of a new system. If the existing system has a similar functionality to the new system being evaluated, one can assume that the same workload may apply. Likewise, if a new system design is shown to produce good results when applied to a wide range of such "recorded" workloads, one can claim the results are truly general and representative. Indeed, numerous papers have used this methodology, exploiting the many workload logs that are freely available, for example, in the Parallel Workload Archive [19]. Alternatively, the recorded workload can be used as the basis for constructing a workload model (like [14, 7, 17]), later to be used to generate input for a simulator of a new system. This has the benefit of allowing for more flexible usage, e.g. by modifying model parameters so as to adapt it to different system configurations.

However, using recorded workloads also has its problems. For example, it is well known that workloads at different installations differ, and that workloads evolve with time as users learn to better use the system [13]. This paper deals with a different type of drawback: real workloads may contain abnormal behaviors that, though they do in fact occur on rare occasions, are not representative in general. Workload flurries are such events. They consist of rare, huge surges of repetitive activity by single users that dominate the workload for a limited time.

Workload flurries have two types of effects on performance evaluation [23, 11]. One is in the context of workload modeling, and specifically the fitting of statistical distributions to workload data. The existence of a flurry may alter workload statistics, leading to the use of unrepresentative values by an unwary analyst. The other is an effect on performance evaluation results when using the workload trace to drive a simulation. Flurries may cause a simulation to be very sensitive to fine details of the system configuration or workload, because the whole flurry reacts to a change *en masse* and thereby amplifies its effect. Hence extremely small modifications may lead to large effects that are not reliable predictors of real performance.

In this paper we focus on the second effect, that is, the effect flurries may have on performance evaluation. We start with a detailed example concerning a real workload trace that spans two years and records 67,667 parallel jobs. We show that shortening the runtime of a *single* 18-hour job by a mere 30 seconds results in an 8% change in the average

1-4244-0054-6/06/$20.00

slowdown of *all* the jobs, solely due to the effect it had on a subsequent 375-job flurry that was submitted by a single user over a period of 10 hours. This motivates the study of flurries as unique and important events in computer workloads in Section 3. Due to the rare and unique characteristics of flurries, we suggest that the appropriate way to deal with them is to filter them out of the workload before using it. We argue that this is similar to removing outliers in statistical analysis, and to removing abnormal data from parallel workloads (e.g. by Cirne and Berman in [7]). "Cleaning" the workload by removing the flurries leads to more stable, reliable, and consistent results, as illustrated in Section 4.

2 A Case Study of Instability

In this section we present a case study showing how the presence of a flurry leads to unstable results: very small changes to the workload are amplified by the flurry and lead to an unexpectedly large change in the results. This example uses the SDSC SP2 log: a listing of 67,667 jobs[1] submitted to the 128-node IBM SP2 installed at the San-Diego Supercomputer Center, from May 1998 to April 2000. All logs used in this paper are later described in Table 1.

2.1 Background: EASY Scheduling and Bounded Slowdown

The case study involves the popular EASY backfilling scheme [16], which is the most commonly used method for batch scheduling parallel jobs at the present time [8] and serves as the default setting of prominent schedulers like Maui, Moab, and the IBM Load-Leveler. Specifically, this scheme is employed by many IBM SP2 machines that generated some of the logs that are used in this paper [19].

The basic scheduling scheme is first-come first-serve (FCFS): jobs are kept in a queue in order of arrival, and whenever there are enough free processors for the first queued job, it is allocated the processors it needs. The problem is that this may lead to fragmentation. If the first queued job requires many processors, it may have to wait a long time until enough processors are freed. During this time, processors are left idle as they accumulate.

To solve this problem, the following optimization is employed: Whenever the system status changes (job arrivals or terminations), the scheduler scans the queue of waiting jobs in order of arrival. Upon reaching the first queued job that can't be started immediately (not enough free processors), the scheduler makes a *reservation* on the job's behalf. This is the earliest time in which enough free processors would accumulate and allow the job to run. The scheduler then continues to scan the queue for smaller jobs (require less processors) that have been waiting less, but can be started immediately without interfering with the reservation. The action of selecting smaller jobs for execution before their time is called *backfilling*, and is illustrated in Fig. 1.

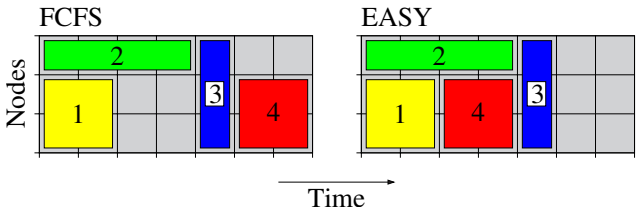


Figure 1: *Left: FCFS scheduling (jobs numbered in order of arrival). Right: FCFS with backfilling. Note that it would be impossible to backfill job 4 had its length been more than 2, as the reservation for job 3 would have been violated.*

Note that backfilling mandates the scheduler to know in advance how long each job will run: to compute the start time of the job that has been waiting the longest (need to know the runtime of job 2 to determine when will job 3 be started) and to know if smaller jobs are short enough to be backfilled (need to make sure job 4 will not delay job 3). The user must therefore provide a runtime estimate upon job submittal. Running jobs that exceed their estimates are killed by the system to make sure waiting jobs will indeed start on time.

One of the most useful metrics used to evaluate job schedulers is the average bounded slowdown. Slowdown is response time normalized by running time, $\frac{T_w+T_r}{T_r}$, where T_r and T_w are the job's runtime and waiting time, respectively. Bounded slowdown eliminates the emphasis on very short jobs due to having the running time in the denominator [10]; a threshold of 10 seconds is commonly used in the context of parallel job scheduling. The definition is

$$bsld = \max\left(1\ ,\ \frac{T_w + T_r}{\max\left(10, T_r\right)}\right)$$

2.2 Example of a Butterfly Effect

The largest user runtime estimate appearing in the SDSC-SP2 trace is 18 hours. This is a limit imposed by the site administrators (different sites may have different limits). Consequently, the longest jobs in this trace are limited to 18 hours (since a job that exceeds its runtime estimate is killed by the system). However, in a real system, it takes some time to propagate the instruction to kill a job to all the nodes[2]. Therefore the trace indicates that some jobs run for a bit more than 18 hours. Of the 67,667 jobs in the trace, only 619 (less than 1%) have runtimes longer than 18 hours.

[1]The full log actually contains 73,496 jobs, but 5829 are recorded as having used zero processors, and having been canceled before they started to run; they are therefore not used in our simulations.

[2]In a simulation this is of course not the case. To maintain the logged runtimes we therefore increase the estimates given in the log to match the actual runtimes.

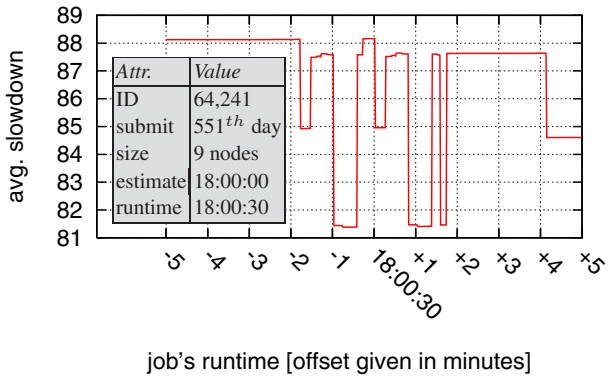


Figure 2: *Average bounded slowdown (obtained by simulating EASY on the SDSC-SP2 trace) as a function of the simulated runtime of the specified job. The job's original runtime is 18:00:30 and so an offset of +1 means the simulated runtime is 18:01:30.*

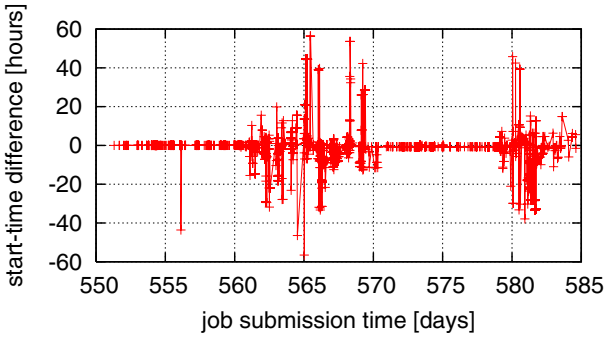


Figure 3: *Shortening job 64,241 by 30 seconds had an effect for more than a simulated month, causing 2,024 subsequent jobs to start earlier or later by up to nearly 60 hours. Only these jobs are shown. The Y-axis indicates the difference between the start time of jobs in the original and truncated simulations (negative values indicate jobs that started earlier due to the truncation).*

In a simulation it is possible to change the irregular runtimes to be *exactly* 18 hours. We found that the average bounded slowdown is surprisingly sensitive to such a change. The following is a striking example that demonstrates this phenomenon: The attributes of job 64,241 are listed in the left of Fig. 2. In our simulation, we have truncated this job's runtime by a mere 30 seconds, and set it to be exactly 18 hours (a modification of 0.0463%). This was the *only* change we've made, that is, we have modified one job out of 67,667 (0.00148%). Remarkably, as a result, the average bounded slowdown of *all the jobs in the trace* changed from 88.16 to 81.38 — that is, by about 8%! Moreover, the effect turned out to be dependant on exactly how much the runtime of this job was changed. Fig. 2 shows the effect of different changes to the runtime of job 64,241 on the overall average. Note that counter intuitively, the average may change by roughly the same amount both by enlarging and by reducing the runtime of the job.

2.3 The Role of a Flurry in Causing the Effect

In a nutshell, the mechanism leading to the above effect has two components. First, the backfilling algorithm propagates the small modification to a single job and influences many other jobs. Second, a whole flurry of similar jobs are affected *en masse*, and their combined weight leads to the observed change in the global average.

In a batch system, a reduction of 30 seconds in the runtime of a job has the obvious immediate (minor) effect of allowing other waiting jobs to obtain the required resources sooner, possibly allowing them to start earlier by up to 30 seconds. But in the context of a backfilling scheduler, a more important effect is that a modification of 30 seconds is enough to make the difference regarding a backfilling decision: by terminating 30 seconds earlier, a slightly larger window is opened, and a job that was previously considered too long to be backfilled may now fit into the available space. This causes a modification of the schedule down the road. Such a chain of modifications allows the effect of one truncated job to accumulate. In our simulation, exactly 2024 jobs were affected by the truncation of the runtime of job 64,241 (in terms of changed start time). The changes in start time are depicted in Fig. 3, where each affected job is represented by a single point. The rest of the schedule remained unchanged.

According to the figure, many of the affected jobs have almost zero difference in start time, and probably reflect the fact that 9 processors became available 30 seconds earlier. The bigger differences between the original and modified schedules are focused in two areas: between days 560-570 (10 days after job 64,241), and between days 580-585 (a month after), and reflect changes in backfilling decisions. Nevertheless, the 8% change in the average bounded slowdown actually stems from start-time differences associated with a group of jobs submitted on the 581st day. This can be seen in Fig. 4, that compares between the running averages of the bounded slowdowns obtained by the two schedules. (The running average at time T is defined to be the average of bounded slowdowns experienced by jobs that were submitted prior to T.) From this figure it is evident that the major difference in overall average performance was due to changes associated with jobs that were submitted at the 581th day, and that all the other changes (e.g. between days 560–570) had a negligible effect.

A closer inspection of the data revels that the perceived change is due to a flurry composed of 375 similar jobs that were submitted sequentially over a period of about 10 hours in the 581st day (exactly one month after the truncated job was submitted). All these jobs were submitted by user 328, required 32 nodes, were estimated to run five minutes, and

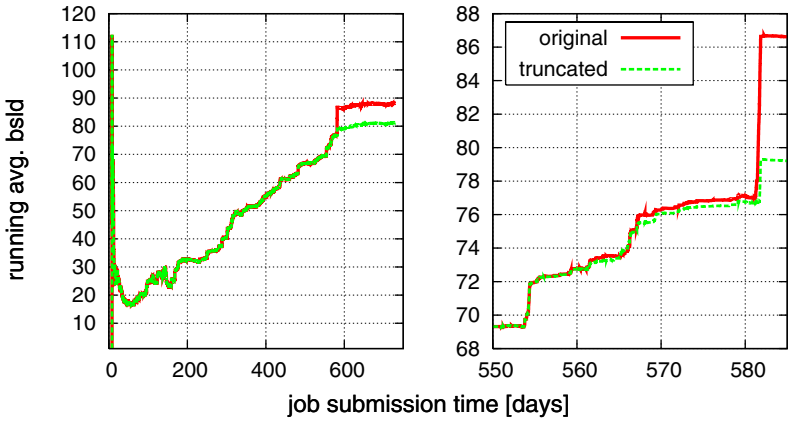


Figure 4: *Running average of the bounded slowdown obtained by the EASY scheduler on the SDSC SP2 trace with/without the 30-second truncation of job 64,241. Left: full trace. Right: zoom in on the part where start-time differences occur.*

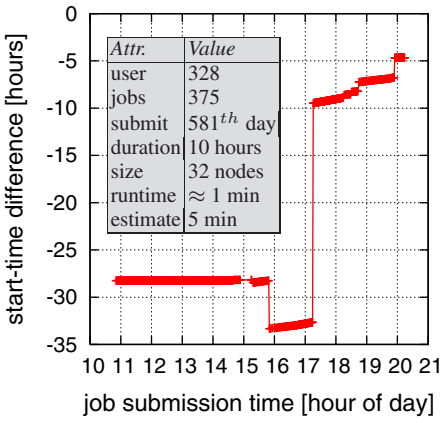


Figure 5: *Start-time differences of the specified flurry jobs by user 328 (X-axis denotes hours on that day).*

ran for about one to two minutes; this is the biggest flurry shown in the right of Fig. 8. The running average of the bounded slowdown of the original and the "truncated" runs were quite similar when the first job of this flurry was submitted (about 1% difference). By the time the last job of the flurry was submitted, the difference was as high as 9%.

Fig. 5 shows the start-time differences associated with the jobs of the flurry (this is a subset of the data displayed in Fig. 3). The jobs' profile similarity along with the fact that they were submitted sequentially, explains their tendency to be effected in the same way by changes to the schedule (in terms of wait time). Note that the effect of shortening the wait time of a job with runtime of around one minute by 30 hours is a reduction of 1800 in its bounded slowdown. This is a huge figure compared to the average bounded slowdown of the entire trace (less than 90), a fact that explains the considerable difference between the slowdown averages of the truncated and original runs.

2.4 Explaining the Sensitivity

Truncating the runtime of job 64,241 (which was submitted 30 days before the flurry) is only one of many trivial modifications we have identified that resulted in a significant change in the average bounded slowdown. These modifications may involve more than one job, and may be applied to jobs with different runtimes, different runtime estimates, and different sizes. However, all these modifications have an effect only when the flurry identified above is scheduled. No other flurry in this log displayed this type of sensitivity. In particular, similar modifications in the neighborhood of the huge flurry identified at the beginning of the log (see Fig. 7) didn't produce similar effects, even though this flurry is an order of magnitude bigger than the flurry above (in terms of the number of jobs composing it).

The reason that the 375-job flurry is so sensitive is that it induces a very high process-load on the system. The process load at time T is defined to be the total number of running or waiting processes (not jobs) that are present in the system at that time instant, divided by the size of the machine. For example, if a machine with 10 nodes is currently running 8 processes (leaving 2 nodes idle), while two jobs of size 6 are waiting in the queue, then its process load is $(8+6+6)/10 = 2$. The left of Fig. 6 displays the evolution of the process load associated with the SDSC SP2 trace. The unequivocal peak in the weekly-average line occurs in the week that contains the 581st day. The right of Fig. 6 shows that this is also reflected in the state of the waiting-queue.

We note in passing that the long-term average process load grows continuously across the trace. This explains the growth trend of the average bounded slowdown (as seen in the left of Fig. 4). It is tolerated by the users because the majority of the jobs still enjoy a fairly reasonable quality of service, as indicated by the bounded slowdown median of the SDSC SP2 trace which is 1.8.

Finally, it should be noted that the effect described above depends on the existence of the flurry, but not only on it. It also depends on the metric being used. When measuring the actual response time, for example, the difference caused by the flurry jobs is not significant enough to change the overall average, because the average response time is dominated by long jobs [9]. By contrast, the average slowdown is dominated by short jobs (that typically have higher slowdowns), so a flurry of short jobs may have a large effect.

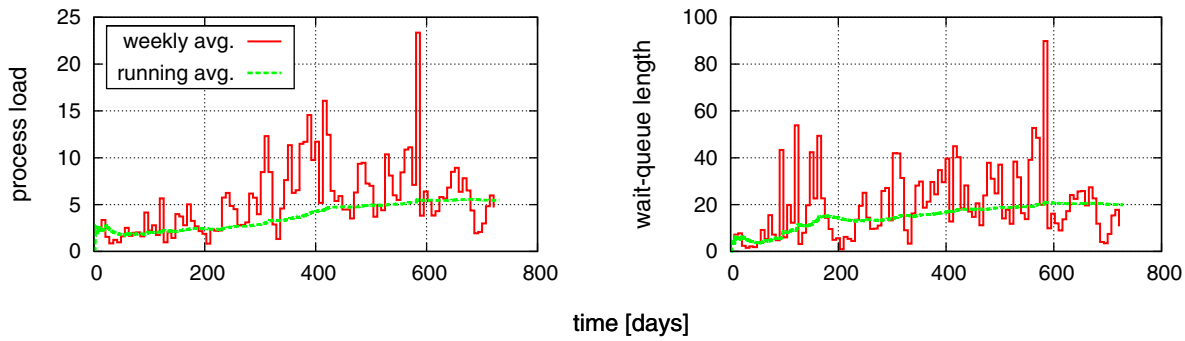


Figure 6: *Evolution of the SDSC SP2 process load and the waiting-queue length when simulating EASY on the original trace.*

Installation	*Abbrev.*	*Machine*	*Start*	*End*	*CPUs*	*Jobs*
Los Alamos National Lab	LANL	CM-5	Oct 94	Sep 96	1,024	201,387
San Diego Supercomp. Ctr	SDSC	Paragon	Jan 95	Dec 96	400	115,595
Cornell Theory Ctr.	CTC	SP2	Jul 96	May 97	512	79,302
San Diego Supercomp. Ctr.	SDSC	SP2	May 98	Apr 00	128	73,496
San Diego Supercomp. Ctr.	SDSC	Blue	Apr 00	Dec 02	1,152	250,440
Utrecht Univ.		DAS	Feb 03	Dec 03	64	33,795

Table 1: *Job logs used in the study.*

3 The Phenomenon of Workload Flurries

Having seen the effect that workload flurries may have on performance evaluations, we now turn to the phenomenon of workload flurries themselves. We define a workload flurry to be a pattern of activity with the following characteristics:

1. it causes a level of activity that is significantly higher than usual, thus dominating the workload,
2. it exists for a limited period of time,
3. it significantly changes the distributions of workload attributes, and
4. it is caused by a single user.

The name "workload flurry" derives from the first and second attributes, and from the fact that the items constituting the flurry are typically lightweight, because otherwise the system would be overwhelmed by their numbers.

The above definition is derived from observations of such phenomena in the long-term workloads experienced by large-scale production parallel supercomputers, as demonstrated now. However, we believe that the phenomenon of workload flurries is much more wide spread, and indeed we have also found such flurries in other system types [23].

Workload data from several large-scale parallel supercomputers is available in the Parallel Workloads Archive [19]. We use data (including job arrivals, runtimes, sizes, etc.) from six different installations as summarized in Table 1. Dozens of research papers have used these and a few other logs as the basis of their evaluations of scheduling mechanisms, oblivious to the fact they contain flurries that might significantly distort the results.

Fig. 7 shows the job arrival rate at the granularity of weeks. In all six logs, large flurries are observed. They range in size from double the average activity to 10 times the average activity, are caused by a single user, and extend from a few days to several weeks. The flurries in the CTC log and the Blue Horizon log seem similar to normal fluctuations, but nevertheless turn out to have an important effect (at least for CTC), as shown in Section 4.

It should be noted that this dataset includes all the long logs in the Parallel Workloads Archive. Flurries were not observed in the shorter logs, including the NASA Ames iPSC/860 (3 months), the KTH SP2 (11 months), the LLNL T3D (4 months), and the LANL Origin 2000 Cluster (5 months). Indeed, periods several months long with no flurry occur also in the six logs that do include flurries.

Fig. 7 shows an especially prominent flurry in the SDSC SP2 data. But this is *not* the flurry that caused the instability described in Section 2. Rather, that flurry is a process flurry, i.e. it includes very many processes but not so many jobs (on parallel supercomputers a job can be composed of dozens of processes). Fig. 8 illustrates the weekly process arrival rate on two of the machines, showing that process flurries do not necessarily correspond to job flurries. In fact, what exactly constitutes a flurry depends on the context in which the question is asked. The "high level of activity" (mentioned as part of the definition of flurries) can in principle also be defined in terms of memory usage, disk operations, or network bandwidth consumed.

The statistical nature of the observed flurries is explored in Fig. 9 (representative for other logs as well). This shows the joint distribution of two major attributes of parallel jobs: the number of processors they use, and their runtime. The flurries tend to correspond to specific locations in these scatter plots, indicating that they are largely composed of jobs with fixed characteristics. In particular, the jobs composing the flurries identified here tend to be small, using few processors and/or running for a relatively short time, as witnessed by the fact that they concentrate near the axes (note that both axes use a logarithmic scale).

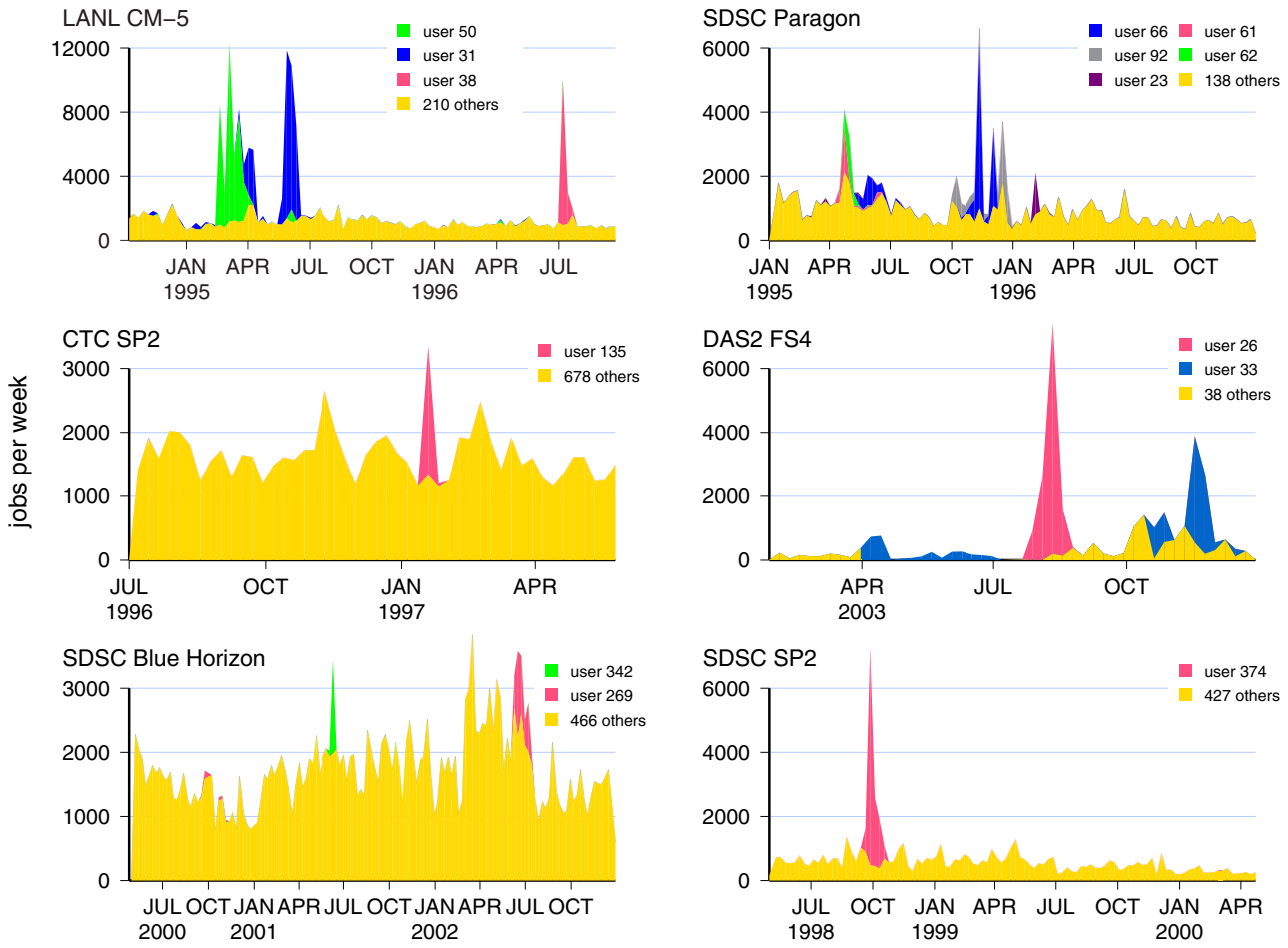


Figure 7: *Job arrivals per week on six large-scale parallel machines. All exhibit flurries of activity due to single users.*

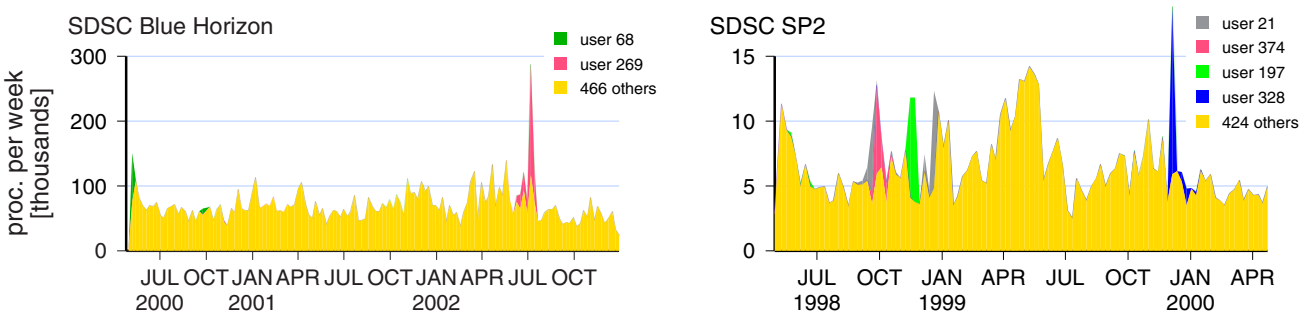


Figure 8: *Process arrivals per week also display flurries, which may be different from the job-arrival flurries.*

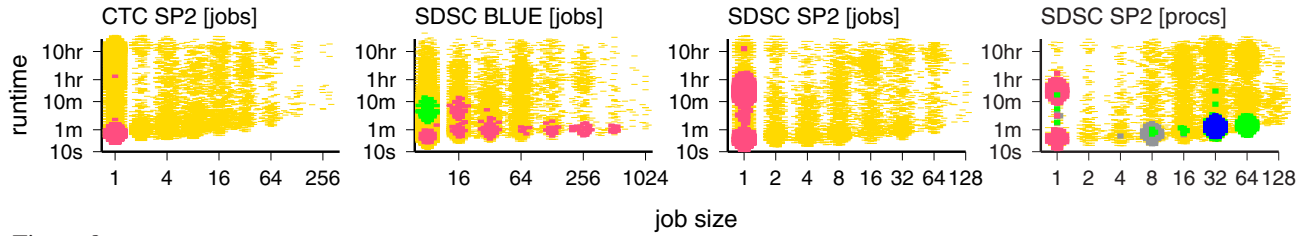


Figure 9: *Scatter plot showing joint distribution of job size and runtime. The single-user flurries typically have rather unique characteristics. Color-coding corresponds to flurries shown in Figs. 7 and 8.*

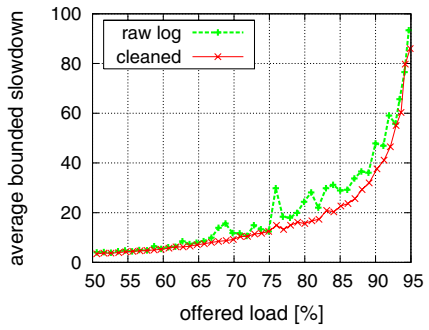


Figure 10: *Simulation of EASY backfilling on CTC. Flurries tend to be sensitive to exact simulation conditions, leading to instabilities. Simulations using a cleaned log are smoother.*

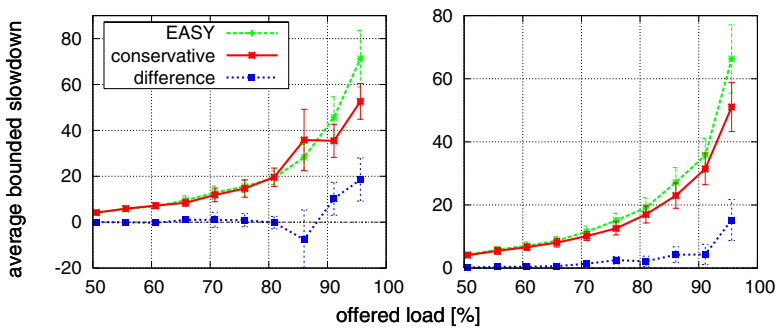


Figure 11: *Comparison of EASY and conservative backfilling, using the CTC log and accurate user runtime estimates. Left: using the complete log leads to inconsistent results. Right: removing the user 135 flurry leads to consistent results showing conservative backfilling is preferable for this scenario.*

4 Impact of Flurries on System Evaluation

As we've seen, simulations of parallel job scheduling can be extremely sensitive to the exact workload conditions. This may also happen in normal evaluations, without any targeted modifications such as the truncation of job 64,241 as described above.

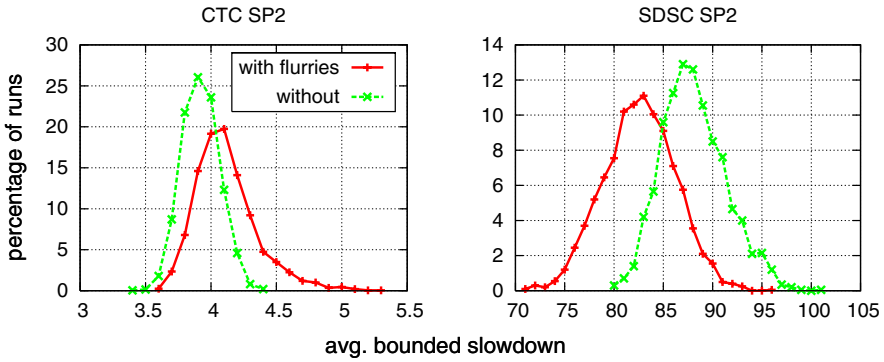
An example is given in Fig. 10, using the CTC workload trace. This is again a simulation of the performance of EASY backfilling, this time showing how it depends on the system's offered load (i.e. the fraction of the machine's capacity that is used). The way to create different offered load conditions is to multiply the job interarrival times by a constant. For example, if the original offered load is ρ, multiplying all arrival times by a factor of $\rho/0.8$ will change the offered load to 0.8. Naturally, this causes the produced schedule to change, as the space available for backfilling is changed. As Fig. 10 shows, such changes to the schedule cause large fluctuations in the bounded slowdown results when using the raw log. It would be ludicrous to take such effects at face value, and claim that, say, the expected performance at a load of 77% is much better than at a load of 76%. In fact, these fluctuations are again examples of amplifications by a flurry: if the 2000-job flurry of activity by user 135 shown in Fig. 7 is removed (this is 2.5% of the total of 78,500 jobs in the log) the result becomes a smooth curve similar to those produced in queueing analysis.

Given that results such as these are hard to predict and correlate with the modifications used to change the offered load, they can also sway the results of evaluations. An example is given in Fig. 11. This shows a study comparing EASY backfilling with conservative backfilling (a version in which reservations are made for all skipped jobs, rather than only for the first one [18]). The study in question dealt with the effect that the accuracy of user runtime estimates have on the performance of the two backfilling schemes [9]. The results shown in Fig. 11 (left) were obtained by simulating the CTC workload using accurate runtime estimates, rather than real user estimates. The results were inconsistent, showing that conservative backfilling produce higher slowdown values for an offered load of 85% but lower values for 90% and 95%. This inconsistency was traced to the same flurry identified above: rerunning the simulations on a modified workload where the flurry was removed led to the cleaner results shown on the right.

However, the results still seem not to be statistically significant as the 90% confidence intervals (computed using the batch means approach) still overlap. We therefore performed a more sophisticated analysis of these results, using the common random numbers[3] variance reduction technique [15]. In this analysis, we first compute the difference in slowdowns between the two schedulers on a per-job basis, which is possible because we are using the same workload trace. We then compute confidence intervals on these differences using the batch means approach. This shows that the flurry indeed makes a big difference in the quality of the results. When it is present, we cannot say anything definite for most offered loads, as the confidence intervals for the difference include 0. When it is removed, the advantage of conservative over EASY is clear across the whole range of offered loads.

A third example is given in Fig. 12. This is again part of the study of the effect of user runtime estimates, this time by randomly shuffling the estimates in the log among the jobs [22]. As a result, the average bounded slowdown is different in each run. The figure shows the histogram of these averages over 2000 runs. When flurries are present, the standard deviation is larger, thereby enlarging the confidence intervals characterizing the result.

[3]The name is somewhat of a misnomer in this case, as we use a logged workload rather than generating it using a random number generator.



stddev	CTC	SDSC
all jobs	0.236	3.616
w/o flurry	0.140	3.235
change	-40.7%	-10.5%

Figure 12: *With randomization, simulation results become non-deterministic. Flurries make them spread out more, reducing the accuracy with which results can be reported.*

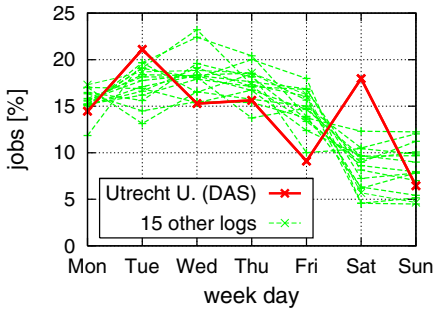


Figure 13: *Weekly arrival pattern on 16 parallel computers, showing abnormal spike on Saturday on the DAS-cluster.*

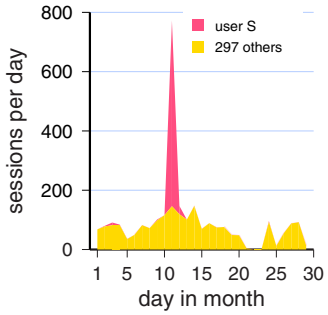


Figure 14: *A flurry in the sessions on a Unix server diverts from users' normal working patterns.*

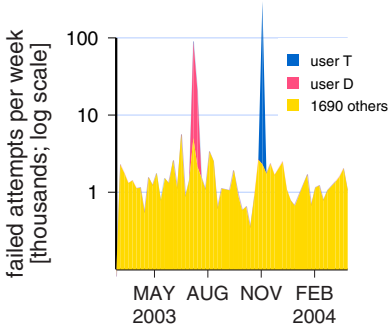


Figure 15: *Flurries of activity on an authentication server are two order of magnitude bigger than the average.*

5 Generalizing

The focus of this paper is the instability induced by flurries on the process of parallel systems evaluation. However, this phenomenon has a broader effect that transcends both the domain of performance evaluation through simulation, and the supercomputers domain.

First, the fact flurries have statistical properties that are different from the "normal" background distributions (Fig. 9) has significant implication on *workload modeling*. A simple example that demonstrates this is given in Fig. 13, which shows the weekly cycle of all 16 logs available through the Parallel Workload Archive [19]. Naturally, more work is being done on weekdays than on weekends. In fact, this is also true within the DAS cluster log, with the single exception of 4,297 jobs submitted by user 26 on Saturday Aug 16 (the major part of the DAS flurry shown in Fig. 7). Indeed, when deleting this flurry, the weekly cycle of the DAS cluster becomes similar to that of the other logs. Obviously, it is erroneous to base a workload model that takes into account the weekly cycle on the DAS-cluster raw log, as in all Saturdays but one the load is relatively low. Many example and further discussion about the dire impact of flurries on workload modeling can be found in [23, 11].

Second, flurries are not unique to parallel supercomputers. Once we became aware of the phenomenon and began to look for it, it was rather easy to find it in other systems. For example, a large flurry was observed in the session log for March 2004 of a Unix server used by students (Fig. 14). This turned out to be the result of ftp'ing a large directory structure by a certain student one afternoon; the (MS Windows) implementation automatically opened a new ftp session for each directory, and this was logged as a distinct user session. Obviously, this data does not represent normal user sessions, and would cause misleading results if used as the basis of an attempt to optimize for interactive user sessions. Another example is the activity on our departmental authentication server (Fig. 15). In this case data covering a long period was available, and two distinct flurries were observed. These were traced to a bug in Windows, where an authentication failure led to an infinite loop of retries.[4].

An important generalization of flurries replaces the source component of their definition: instead of being work

[4]Indeed, it is possible that some of the flurries on supercomputers are also the result of runaway scripts rather than being intentional. This does not detract from the importance of the phenomenon. On the contrary, situations in which flurries are unintentional add motivation to the need to identify them before using the workload as representative of normal work.

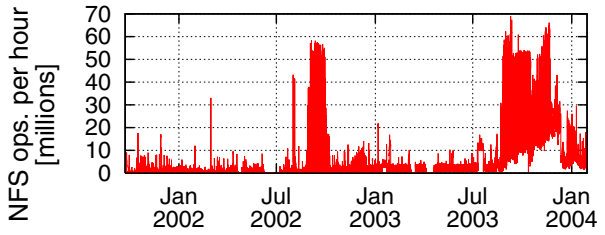


Figure 16: *Activity on a departmental NetApp filer.*

generated by a single user, we can consider work generated by a singular event. Two examples are shown in the file server data of Fig. 16. The first high-load event, in September 2002, is attributed to a massive copying due to a hardware upgrade. The second, during September to December 2003, is attributed to a bug in a new release of the GNU C library[5] [21]. Installing the new version is the event that triggered this flurry of activity, and fixing it ended the flurry.

There are many accounts of flurry-like events on the Internet, provided we generalize the notion of source from a single user to some singular event that attracts many users (but still a small subset of all Internet users, and for a limited time). For example, new releases of software by Microsoft have caused the so called "midnight madness" phenomenon, where users flocking to download the new version (typically released at midnight) saturate the network and overwhelm the servers [20]. Other examples include the surge of activity on CNN's servers on September 11, 2001, and the usage of sites set up especially to cover sporting events such as the Olympic games or the World Cup finals [3]. All of these events are singular, and lead to unique traffic patterns. We claim that it would be wrong to use workload data including such singular events to analyze the performance of web servers under normal conditions, just as it would be wrong to use normal data for an evaluation of how systems would behave under unique conditions. Of course, in these particular cases, high-load conditions may be more important and meaningful than normal conditions; if this is the case, they should be the focus of study rather than being eliminated as suggested below. For example, Ari et al. model such activity, which they call "flash crowds", with the aim of evaluating schemes to survive them [2].

Targeted attacks on specific servers also qualify as flurries. In many cases, the nature of the attack is to flood the server and overwhelm it with a load that is much higher than its capacity. This load is generated by a small group of machines (relative to the whole Internet), and lasts for a limited, well-defined time. In this case, an analysis of the attack workload patterns is not only useful for evaluation of servers, but also as a tool in identifying such attacks [4].

[5]The bug is that the d_off field in the dirent structure isn't maintained correctly by the auto-mount daemon. Specifically, the 64-bit offset is either 0 or a garbage value. When using a 32-bit file system interface (like the libc readdir routine), getdents verifies that only 32 bits are actually used, and therefore fails if the garbage contains more bits. In trying to handle this error it attempts to seek to the beginning of the erroneous entry, identified using the offset of the previous one. But this is also a garbage value. And if it is 0, we end up with an endless loop of repeatedly reading the first entry, which is what caused the surge of activity seen in Fig. 16.

6 Discussion and Conclusions

Workload characterization and modeling have been advocated and practiced for many years [12, 1, 5]. This is typically done by collecting workload traces, and creating a statistical model based on fitting the distributions of workload attributes [15]. But such an approach is questionable if the data is not stationary, as seems to be the case in the context of parallel supercomputers. For example, Chiang et al. analyze six months of data from the NCSA Origin 2000 machine, and find that their load conditions may be quite different from each other [6]. We identify flurries as a specific type of deviations from stationarity that have to be taken into account when creating a workload model.

We suggest that the flurries phenomenon cannot be ignored, as they might lead to large effects on workload modeling and on performance evaluations using real workloads. Instead, the workload needs to be separated into "normal" workload and "flurries". Modeling and evaluation of normal workloads can then be performed using current methodologies. This may be expected to lead to reliable and consistent results that are applicable most of the time (during which flurries are not present). Comparing evaluation results using the cleaned log against those based on the raw log will identify whether the removed flurries actually have a significant effect in the specific case being studied.

The main justification for removing flurries steams from the fact they are rare and unique: Using a workload with a flurry in effect emphasizes the rare and unique event at the expense of normal conditions. Thus *leaving the flurry in* is actually the unjustifiable approach. To argue for evaluations based on workloads with flurries, one must argue that the activity of a specific user during a short time should indeed dominate the evaluation results. Also, one must be satisfied with results that may change considerably if the span of time covered by the evaluation is shifted such that the flurry happens to be excluded.

The question is then how to identify and remove the flurries. The methodology we have used is to plot activity levels as a function of time. In the case of parallel jobs, this means job or process arrivals per unit time. In other contexts, other workload attributes would be appropriate. For example, when analyzing Internet traffic one can tabulate packets and flows; for storage systems, one can look at I/O operations and at bytes transferred.

Once a period of time with exceptionally heavy load is identified, this load should be checked for uniformity and source. The flurries we have identified were all composed of

numerous repetitions of the same type of work. Identifying this is the key for removing the flurry from the workload, as the combination of the time frame and the flurry's specific attributes often provide an effective filter. As finding flurries is not trivial, this information should be shared together with the original data. In other words, when workload data is made available, it should be accompanied by all the accumulated knowledge regarding problems with its use, and specifically, with information regarding flurries that occur in it. As a first step, we have added our data to the Parallel Workloads Archive [19], from which our original data comes, and which is used by many researchers for numerous studies of parallel job scheduling.

We view this as a first step because, somewhat surprisingly, computer systems analysts rarely verify the integrity of the data on which they rely for their analysis, and the overwhelmingly common case is to use the data "as is" (e.g. consider all the papers that fit distribution against log files without even considering whether some sanitation is in order [11]). This is in disagreement with what is routinely done in every statistical analysis, where data is throughly validated, outliers are removed when necessary, etc. Rare studies that do attempt to sanitize, tend to have a "local" or "specific" nature, targeting a single attribute or concept instead of providing a generalization like we do in this paper. For example, in an attempt to model the daily cycle of the jobs submittal process, Cirne and Berman clustered days, and excluded clusters populated by only one day from participating in the evaluation [7] . The flurries phenomenon suggests this approach is problematic because (1) "normal" jobs are also needlessly excluded, and (2) flurries may span more than one day and thus be erroneously included.

Of course, just eliminating flurries is also not a good solution, as flurries do in fact occur. An open question is how to model or evaluate the effect of the flurries on a system designed and optimized for the more common non-flurry workload. An obvious first step is to use specific flurries that occur in recorded workloads and study their effect. But it is doubtful whether this can predict the effect of other potential flurries. Important future work is therefore to develop methods to extend and generalize the results obtained with specific flurries, and try to derive bounds on the effects of other potential flurries.

To summarize, it is extremely important to use real data regarding the workload on computer systems. But it is equally important to ensure that this is high-quality and representative data. Using measured workloads indiscriminately risks the introduction of unknown anomalies that may lead to unknown effects. Workload flurries are such an anomaly, and should be handled with care.

Acknowledgment This research was supported in part by the Israel Science Foundation (grant no. 167/03).

References

[1] A. K. Agrawala, J. M. Mohr, and R. M. Bryant, "*An approach to the workload characterization problem*". *IEEE Computer* **9(6)**, pp. 18–32, Jun 1976.

[2] I. Ari, B. Hong, E. L. Miller, S. A. Brandt, and D. D. E. Long, "*Managing flash crowds on the Internet*". In 11th *IEEE Intl. Symp. on Modeling, Analysis, & Simulation of Comput. & Telecommunication Syst. (MASCOTS)*, pp. 246–249, Oct 2003.

[3] M. Arlitt and T. Jin, "*A workload characterization study of the 1998 world cup web site*". *IEEE Network* **14(3)**, pp. 30–37, May/Jun 2000.

[4] M. Burgess, H. Haugerud, S. Straumsnes, and T. Reitan, "*Measuring system normality*". *ACM Trans. on Comput. Syst. (TOCS)* **20(2)**, pp. 125–160, May 2002.

[5] M. Calzarossa and G. Serazzi, "*Workload characterization: a survey*". *Proc. IEEE* **81(8)**, pp. 1136–1150, Aug 1993.

[6] S-H. Chiang and M. K. Vernon, "*Characteristics of a large shared memory production workload*". In *Job Scheduling Strategies for Parallel Processing*, pp. 159–187, Springer Verlag, 2001. Lect. Notes Comput. Sci. vol. 2221.

[7] W. Cirne and F. Berman, "*A comprehensive model of the supercomputer workload*". In 4th *Workshop on Workload Characterization*, Dec 2001.

[8] Y. Etsion and D. Tsafrir, *A Short Survey of Commercial Cluster Batch Schedulers*. Technical Report 2005-13, Hebrew Univ., May 2005.

[9] D. G. Feitelson, "*Metric and workload effects on computer systems evaluation*". *IEEE Computer* **36(9)**, pp. 18–25, Sep 2003.

[10] D. G. Feitelson, L. Rudolph, U. Schwiegelshohn, K. C. Sevcik, and P. Wong, "*Theory and practice in parallel job scheduling*". In *Job Scheduling Strategies for Parallel Processing*, pp. 1–34, Springer Verlag, 1997. Lect. Notes Comput. Sci. vol. 1291.

[11] D. G. Feitelson and D. Tsafrir, "*Workload sanitation for performance evaluation*". In *IEEE Intl. Symp. Performance Analysis of Syst. and Software (ISPASS)*, Mar 2006.

[12] D. Ferrari, "*Workload characterization and selection in computer performance measurement*". *IEEE Computer* **5(4)**, pp. 18–24, Jul/Aug 1972.

[13] S. Hotovy, "*Workload evolution on the Cornell Theory Center IBM SP2*". In *Job Scheduling Strategies for Parallel Processing*, pp. 27–40, Springer-Verlag, 1996. Lect. Notes Comput. Sci. vol. 1162.

[14] J. Jann, P. Pattnaik, H. Franke, F. Wang, J. Skovira, and J. Riodan, "*Modeling of workload in MPPs*". In *Job Scheduling Strategies for Parallel Processing*, pp. 95–116, Springer Verlag, 1997. Lect. Notes Comput. Sci. vol. 1291.

[15] A. M. Law and W. D. Kelton, *Simulation Modeling and Analysis*. McGraw Hill, 3 ed., 2000.

[16] D. Lifka, "*The ANL/IBM SP scheduling system*". In *Job Scheduling Strategies for Parallel Processing*, pp. 295–303, Springer-Verlag, 1995. Lect. Notes Comput. Sci. vol. 949.

[17] U. Lublin and D. G. Feitelson, "*The workload on parallel supercomputers: modeling the characteristics of rigid jobs*". *J. Parallel & Distributed Comput.* **63(11)**, pp. 1105–1122, Nov 2003.

[18] 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.

[19] "*Parallel workloads archive*". www.cs.huji.ac.il/labs/parallel/workload/.

[20] E. Schooler and J. Gemmel, *Using multicast FEC to solve the midnight madness problem*. Technical Report MS-TR-97-25, Microsoft Research, Sep 1997.

[21] D. Tsafrir, "*Bug (+fix) in getdents() [glibc-2.3.2/linux-2.4.22/i686]*". URL http://sources.redhat.com/ml/bug-glibc/2003-12/msg00028.html, Dec 2003.

[22] D. Tsafrir, Y. Etsion, and D. G. Feitelson, "*Modeling user runtime estimates*". In *Job Scheduling Strategies for Parallel Processing*, pp. 1–35, Springer Verlag, Jun 2005. Lect. Notes Comput. Sci. vol. 3834.

[23] D. Tsafrir and D. G. Feitelson, *Workload Flurries*. Technical Report 2003-85, Hebrew Univ., Nov 2003.