Using Site-Level Modeling to Evaluate the Performance of Parallel System Schedulers

Edi Shmueli*† Dror G. Feitelson*
*School of Computer Science & Engineering, Hebrew University, Jerusalem 91904, Israel
†IBM Haifa Research Lab, Mount Carmel, Haifa 31905, Israel

Abstract

The conventional performance evaluation methodology for parallel system schedulers uses an open model to generate the workloads used in simulations. In many cases recorded workload traces are simply played back, assuming that they are reliable representatives of real workloads, and leading to the expectation that the simulation results actually predict the scheduler's true performance. We show that the lack of feedback in these workloads results in performance prediction errors, which may reach hundreds of percents. We also show that load scaling, as currently performed, further ruins the representativeness of the workload, by generating conditions which cannot exist in a real environment. As an alternative, we suggest a novel site-level modeling evaluation methodology, in which we model not only the actions of the scheduler but also the activity of users who generate the workload dynamically. This advances the simulation in a manner that reliably mimics feedback effects found in real sites. In particular, saturation is avoided because the generation of additional work is throttled when the system is overloaded. While our experiments were conducted in the context of parallel scheduling, the idea of site-level simulation is applicable to many other types of systems.

1. Introduction

In the conventional performance evaluation methodology for parallel systems schedulers, a model of the scheduler is exercised in a simulation using a workload made of a stream of incoming job submission requests. The source for that stream is usually a trace that was recorded on a real system. At the end of the simulation, performance metrics collected for the scheduler model are used to predict the scheduler's performance in a real environment.

To generate the workload from the trace, the conventional methodology uses an open model, where the trace is simply replayed according to the timestamps of the submission requests, and there is no feedback between the completion of jobs and the submission of subsequent jobs. To evaluate the scheduler's performance under different load conditions, the timestamps in the trace are modified before the simulation begins; inter-submission times are reduced or expanded to increase or decrease the load, respectively.

Whether the load is modified or not, an underlying premise is that the generated workloads are reliable representatives of workloads that would be observed by the scheduler in a real environment. We argue that this is not the case because these workloads lack the feedback effects that naturally exist between users and the scheduler, and show this lack of feedback may result in inaccurate performance predictions of hundreds of percents. We also argue that load scaling as currently performed further ruins the representativeness of the workload, by violating precedence relations that existed in the real environment.

To get accurate performance predictions and allow for safe load scaling, we suggest a novel *site-level modeling* evaluation methodology, in which the workload for the simulation is *not* generated by replaying a trace, but dynamically, in a manner that reliably mimics feedback effects found in reality. A site-level model includes not only the scheduler but also the users who generate the workload. When the users wait for their jobs to complete, they introduce feedback to the workload generation process, because the completion of jobs depends on the load in the system and on the scheduler's ability to cope with that load.

To study these feedback effects, we analyzed recorded system traces in an attempt to understand the way users submit jobs to the scheduler. To our best knowledge, this is the first attempt to extract such information from traces.

We found that users *job submission behavior* can be modeled using *batches*: groups of jobs submitted asynchronously, i.e. without waiting for one job to complete before submitting the next, and with short inter-submission times between them. Furthermore, the job submission model is independent of the characteristics of the jobs themselves. The latter can be derived using a separate model we named the *workpool* model. Together, the two models dynamically generate the stream of jobs to be scheduled.

We implemented all this in the *SiteSim* framework for site-level simulations. SiteSim enables the easy development of new job submission and workpool models, combining them in various ways to change the characteristics of the generated workload, and evaluating different sched-

uler models using these workloads in a reliable manner. It also generates a trace of each simulation, which can be used for conventional simulations. We use these traces to demonstrate the differences between the conventional and site-level approaches.

This paper is organized as follows: Section 2 describes the conventional performance evaluation methodology, motivates a new methodology, and describes our novel site-level modeling evaluation methodology in detail. Sections 3 and 4 describe the job submission behavior and workpool models. SiteSim is discussed in Section 5. Section 6 describes the experiments we performed to demonstrate the effect of feedback on evaluations. Section 7 outlines related work, and Section 8 concludes and discusses future work.

2. Site-Level Modeling

2.1. Background: The Conventional Methodology

Scheduling policies for parallel systems have been the subject of intensive research for many years. This research is often based on simulations, due to the impracticality of performing evaluations on real production systems, and the reduced level of detail possible with mathematical analysis. In a simulation, a model of the scheduler is exercised using a workload made of a stream of incoming job submission requests. Such a stream is often generated by *replaying* a trace containing a list of submission request records, that is, records of jobs that were actually submitted to and executed on a production-use parallel machine.

Within the trace, each submission request has a *timestamp* which specifies when the job was submitted, and also several attributes that specify the resources used by the job. For a space-sharing parallel machine executing rigid jobs, typical attributes are the job's *size* — the number of processors used by job, and the job's *runtime* — the interval of time during which these processors were used, and thus were unavailable for other jobs. When the trace is used for simulation, these attributes are treated as the resource *requirements* of the job. In addition, jobs may have a *runtime estimate* — a rough estimation provided by the user and used by the scheduler to plan ahead.

Replaying a trace directly implies an *open model*, in which the jobs' submission rate is dictated by the timestamps from the trace. Sometimes these timestamps are scaled by a certain factor, so as to increase or decrease the load. In either case, there is no feedback between the completion of jobs and the submission of subsequent jobs.

The alternative is a *closed model*, having an unconditional feedback between the completion of a job and the submission of the the next job. For this model, the timestamps in the trace are ignored and submission requests are issued only after a previous job completes. The problem is that this leads to extreme regularity: there are no bursts of activity — severely limiting the optimizations that can

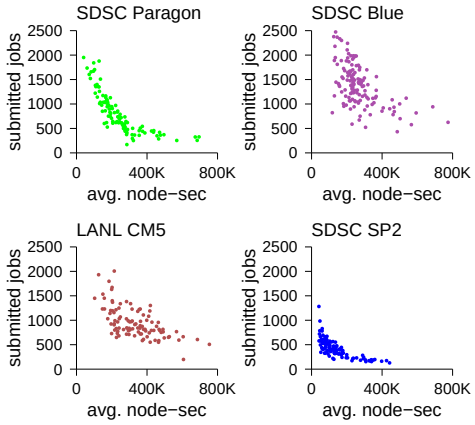


Figure 1. The effect of feedback in real workloads: in weeks where many jobs are submitted they tend to be small, and vice versa.

be performed by the scheduler, and there is no way to manipulate the load for the simulation. For these reasons, the conventional performance evaluation methodology adopted the open model in generating the workload.

During the simulation statistics are recorded for each individual job. These typically includes the job's *wait-time* — the time the job spent in the scheduler's queue waiting for processors to become available, the *response-time* — the time between submission to completion (wait time + running time), and the *slowdown* — the response time normalized by the actual runtime, which shows how much slower the job ran due to the load on the system. At the end of the simulation, the means for these metrics are calculated and used as performance metrics.

2.2. Motivating a New Methodology

We argue that workloads generated by replaying traces are not reliable representatives of real workloads, because the traces contain a *signature* of the feedback effects which existed between the users and their scheduler when the trace was recorded, and that replaying that signature during simulation leads to inaccurate performance predictions.

Consider for example a loaded system where jobs wait for a long time in the scheduler's queue for processors to become available. Because users often wait for their jobs to complete before submitting more jobs, such a high load will actually cause the submission rate to decrease, eventually leading to a decrease in the load. As the load decreases, jobs wait less time in the queue and respond faster, causing the submission rate to increase again, eventually leading to a higher load, etc.

Figure 1 illustrates that such *self regulation* by users (avoid submitting additional jobs if the system is already overloaded) indeed exists in real workload traces. The data is from extensive logs of jobs executed on large scale paral-

lel machines (the original data and additional information is available in the Parallel Workloads Archive [14]). In these scatter plots, each log is partitioned into weekly slices. For each slice, the number of jobs submitted is counted. In addition, the average node-seconds needed by these jobs is tabulated. Plotting one against the other shows that when there are many jobs, they tend to be smaller; when jobs are heavy, there tend to be fewer of them.

Such feedback effects leave their signature in the trace in the form of timestamps for each submission request record. When replaying the trace according to these timestamps, the generated workload matches the scheduling policy that was in effect on the traced system, instead of adapting itself to the scheduling policy being evaluated. This means that the rate of submissions will not decrease if the scheduler model fails to handle the load, nor will it increase if the model handles the load easily, causing performance prediction to be underestimated or overestimated.

We also argue that load scaling, as currently performed, further ruins the representativeness of the workload, because it generates conditions which cannot exist in a real environment. One example is the violation of dependencies between job completions and subsequent submissions. As noted above, users often wait for their jobs to complete before submitting more jobs, which means that some jobs simply will not reside together in the scheduler's queue. When scaling the load by modifying the jobs' submission timestamps, submission requests may be issued before the jobs they depend on complete.

Finally, the conventional performance evaluation methodology uses metrics that are conjectured to be good proxies for user satisfaction, such as the wait time or response time. It does not support a metric that directly quantifies productivity, such as throughput. This is an inherent problem with open system models, because in such models the throughput is dictated by the workload and is not affected by the scheduler (at least as long as the system is not overloaded).

The methodology we suggest below incorporates feedback into the workload generation processes to get accurate performance predictions, but unlike the pure closed model discussed above, it postulates bursts of jobs, allowing the scheduler to perform optimizations. Load scaling is performed in a safe manner that preserves the workload representativeness, and throughput (and hence productivity) becomes a metric that can be measured.

2.3. Site-Level Modeling Details

We suggest a novel *site-level modeling* evaluation methodology to accurately predict the true performance of parallel system schedulers. The essence of this methodology is that workloads are generated dynamically during the simulation in a manner that reliably mimics feedback effects found in real sites. This implies that we simulate not only the evaluated scheduler, but also the users, who generate the workload for the scheduler. During their activity periods, known as *sessions*, users may wait for their jobs to complete; when they do, they introduce feedback into the workload generation process.

Category	Conventional	Site-Level
Evaluation tool	Simulation	Simulation
Workload source	Trace replay	Users sessions
Job submissions	Trace timestamps	Submission behavior model
Job characteristics	Trace based	Workpools model
Load scaling	Trace (de)-compression	Number of sessions Submission model

Table 1. Methodology Comparison.

In addition to the scheduler and user models, a complete site-level simulation may also include a machine model. This may be important because the performance of specific applications may be affected by the machine's architecture [17], or by interference from other jobs [13]. However, such detailed simulations require much more information about applications and take much longer to run. In this paper we wish to focus on the feedback effects related to the workload generated by the users. We therefore assume that job runtimes are not affected by the system state. This assumption is often made in conventional simulations.

The workload observed by the modeled scheduler at any given time during the simulation is a combination of workloads generated by all user sessions that are active at that time (Figure 2). A session model has two components: a job submission behavior model and a workpool model. The job submission behavior defines the *structure* of the session, i.e. when the user submits more jobs and when he waits for jobs to complete. The workpool model specifies the *characteristics* of the jobs. Importantly, we found that these two models are independent of each other. This contributes to the flexibility of the simulation, allowing to experiment with different models and modify the workload generated by each session.

Table 1 summarized the difference between the conventional and site-level evaluation methodologies.

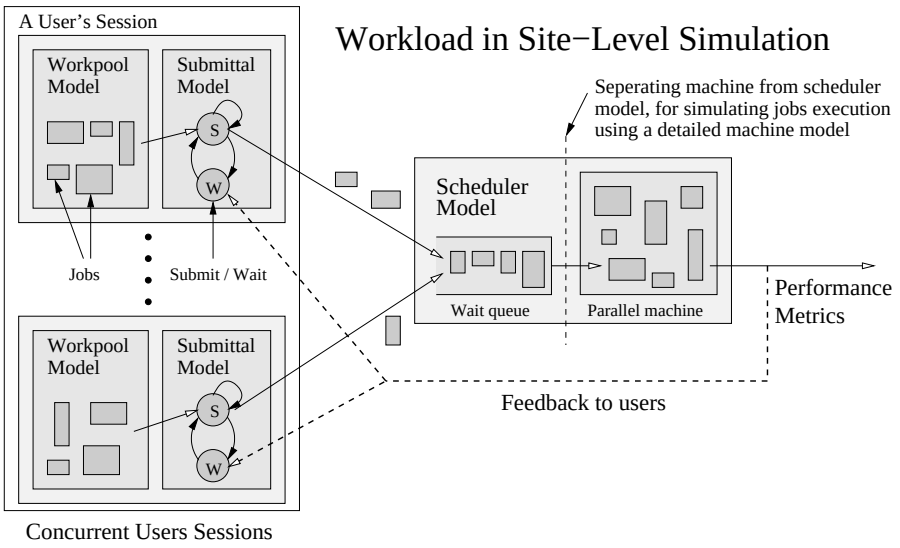
3. Modeling Users Job Submission Behavior

3.1. The Structure of Sessions

Users interact with computer systems in periods of continuous activity known as sessions [8, 1, 18]. For parallel systems schedulers, a session is made of one or more job submissions.

Zilber et al. [18] analyzed several parallel system scheduler traces and classified user sessions. A preliminary step to extracting sessions data was to determine the session

Figure 2. In site-level simulation the workload observed by the scheduler model is dynamically generated by several concurrent user sessions. Each session has a workpool model that defines the characteristics of the submitted jobs, and a job submission model that defines when they are submitted and introduces feedback to the workload generation process.



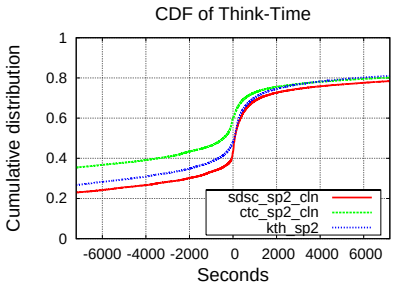


Figure 3. CDF of "think times". Negative values indicate that one job started before the previous one completed.

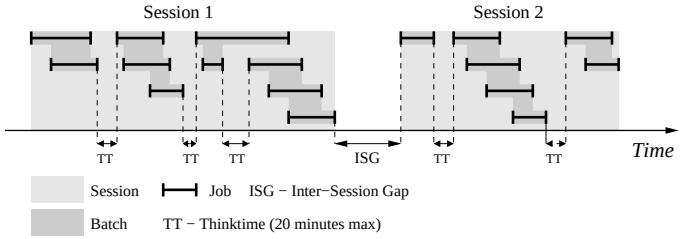


Figure 4. Sessions and batches.

boundaries. This was done by setting a threshold on the think-time distribution: shorter think times are considered to be think times within a session, while longer ones are periods of inactivity after which a subsequent submission starts a new session.

The CDF of think-times for the different traces is shown in Figure 3 (the data is again from the Parallel Workloads Archive [14]). The plots show that at about twenty minutes the CDF stops its steep climb, which means that a large portion of the jobs are submitted within twenty minutes of the completion of a previous job — indicating continuous activity periods by the users. Furthermore, beyond twenty minutes and for rest of the time scale the think-times are evenly distributed, without any features indicating a natural threshold. Zilber et al. therefore defined sessions to be sets of jobs submitted within twenty minutes from the completion of the previous job. In our work we adopt this definition.

Another feature of the think time distribution, which has little importance for session classification, but is highly important for understanding users' job submission behavior, is the fact that a major fraction of the think-times (over 50% for some traces) is below zero. These negative values result from the definition of think-time as *the time between the completion of the previous job and the submission of the current job*; they indicate that jobs were submitted before the previous job completed.

With respect to users job submission behavior this means that within sessions, users submit jobs either *synchronously* or *asynchronously*. Synchronous submissions are those that may depend on the completion of previous jobs, as identified by a positive think time. These submissions thus effectively depend on the load on the system and the scheduler's ability to handle that load, and provide the desired feedback. Asynchronous submissions are those that do not wait for the previous job to complete. These submissions occur regardless of the state of the system.

For the purpose of modeling the users' job submission behavior, we define a *batch* to be a set of jobs submitted asynchronously to one another, and the term *batch-width* to denote the number of jobs in the batch. Using this definition, a single job submitted synchronously is simply a special case batch that has a width of one.

Batches provide a convenient way to model the way users submit their jobs: a session is made of a series of one or more batches, where each batch contains one or more

jobs. The time between the termination of the last job in a batch and the submission of the first job in the next batch must not exceed twenty minutes — the session's think-time boundary. Within a batch, all jobs except the first are submitted before the previous job completes. All this is illustrated in Figure 4.

3.2. Simple Job Submission Behavior Model

To model the users job submission behavior we thus need three sets of data

- The distribution of batch-widths
- The distribution of job inter-submission times within batches
- The distribution of (positive, inter-batch) think-times of up-to twenty minutes

Data for these distributions can naturally be obtained by analyzing workload traces from different parallel machines. Given the data, one can model it by fitting appropriate probability distribution. Alternatively, one can use the empirical data directly. As fitting distributions is secondary to our primary goal of demonstrating the importance and effect of feedback, we use empirical distributions from the SDSC-SP2, CTC-SP2 and KTH-SP2 traces in the simulations reported in this paper.

Figure 5 shows the distribution of batch-widths for the three workloads. Obviously the distributions are quite similar in all the traces, indicating that this data is representative of user job submission behavior in general. The dominating fraction of batches are of width one. Batches of width 2 are the second most common, accounting for about 10% in each trace. Larger batches are progressively rarer.

The distributions of inter-submission times for asynchronous job submissions within a batch is also shown. Note that this refers to the time from *one submittal to the next*, and is therefore non-negative (as opposed to the think time, which is the interval from a termination to a submittal). These and the distributions of think-times between batches favor short times, and are also consistent across the three traces.

4. Modeling Workpools

We claim that the users job submission behavior is largely independent of the characteristics of the jobs that are submitted. Modeling flexibility is enhanced by associating each session with distinct job submission and workpool models, which define the characteristics of the batches and jobs submitted during that session. In principle, the models should be statistically different, e.g. one model for light day jobs and another for heavy night jobs. However, in our current implementation, they all draw from the same empirical distributions.

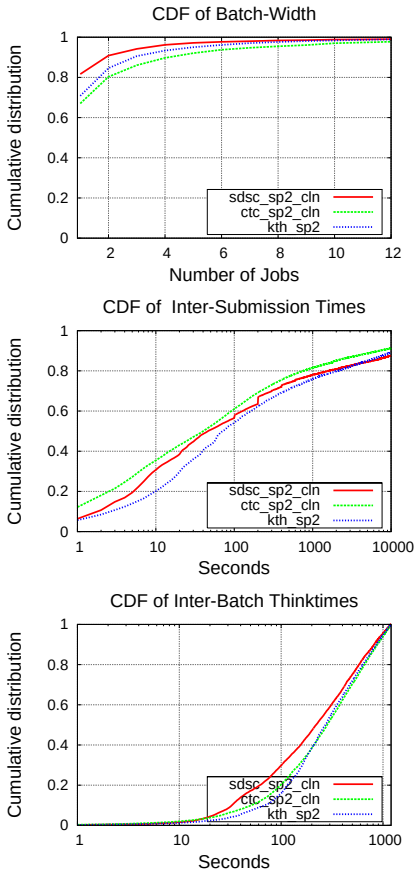


Figure 5. Distributions of batch-widths, jobs' inter-submission times within batches, and think times between batches.

A basic workpool model is essentially composed of two distributions, corresponding to the two main attributes of parallel jobs:

- Size: the number of processors required for the job to execute, assuming pure space slicing.
- Runtime: the actual time it will execute once all processors have been allocated

Analyzing the traces also indicates that jobs display a "locality of sampling": successive jobs tend to be very similar to each other. This may be because users actually submit the same jobs repeatedly. To capture this effect, we also tabulate the distribution of such repetitions.

Just like for the job submission behavior model, we model workpools using empirical data drawn from the three traces. The distribution of the jobs sizes for the three traces is shown in Figure 6. As has been observed before, this is a modal distribution with most jobs using power-of-two nodes [5]. The distributions of runtimes and repetitions are also shown. Again we see that they are reasonably similar across the three traces.

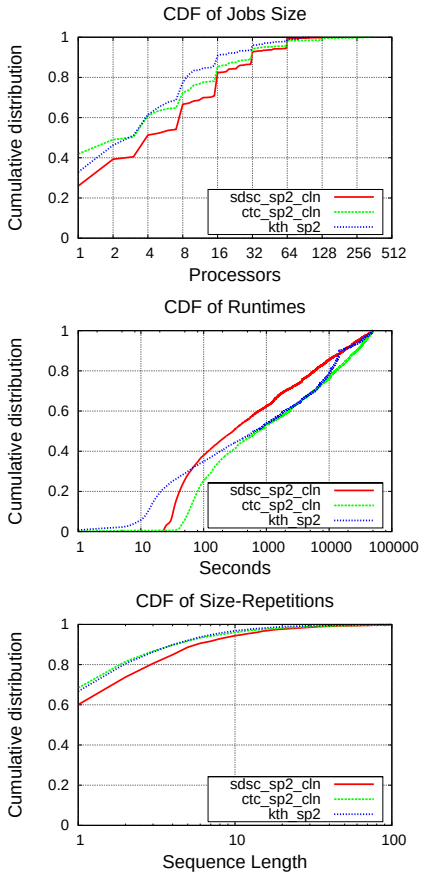


Figure 6. Distributions of job sizes, runtimes, and repetitions that cause locality effects.

5. Simulation Framework

Our site-level simulation framework, *SiteSim*, enables easy development and combination of job submission and workpool models, and reliable evaluation of different scheduler models using dynamically generated workloads.

SiteSim defines two types of entities: *users* and *schedulers*. Users generate the workload in periods of activity called sessions. At present, only a static set of sessions is supported, and there are no user arrivals or departures. As explained in Section 2.3, simulating a detailed machine model is not required, and therefore the machine model is embedded in the scheduler model. SiteSim supports multiple schedulers (and machines) in the same simulation, to allow the modeling of machines with multiple partitions.

SiteSim can also run conventional simulations, simply by replaying standard workload format traces (See http://www.cs.huji.ac.il/labs/parallel/workload/swf.html). For these simulations only one user is defined, with a job submission model that uses the jobs' submission timestamps from the trace, and a workpool model that takes job characteristics from the trace.

In the simulations reported here, we typically define 10 users, who all use the same job submission and workpool models. At the end of a simulation, SiteSim generates per-user and system-wide statistics, and a standard format trace containing full data for each job. This can be used for post-mortem analysis or for other simulations.

As noted above, the submission behavior and workpool models we use are based on empirical data from real traces. We used distributions generated by combining the data from all three traces. In the simulations, we randomly generated job attributes from these distributions, and repeated jobs according to the distribution of repetitions. To validate this approach we plotted the resulting distributions of workload attributes, which were indeed found to be very similar to the original distributions.

6. The Effect of Feedback on Evaluations

6.1. Inaccurate Performance Predictions

Users often wait for their jobs to complete before submitting more jobs. If they use a low-end scheduler, that fails to optimize the machine's resource usage, their jobs will spend a long time in its queue waiting for resources to become available. As a result, new job submissions will be delayed. On the other hand, if the machine's resources are managed by a high-end scheduler, queuing times shorten and jobs complete earlier, causing new submissions to be issued earlier.

When recording job submissions in a trace, the submission timestamps form a signature of the feedback effects between the users and the scheduler, and as explained above, different schedulers would result in different signatures. Later, when the trace is replayed during a simulation, it is the original signature that determines the rate of submissions. A trace from a high-end scheduler will contain a signature that, when replayed, will generate higher loads compared to a trace from a low-end scheduler.

To quantify how inaccurate performance predication using the conventional methodology may be, we designed an experiment in which a low-end scheduler is evaluated using a trace from a high-end scheduler, and vice-versa (Figure 7). The idea is that the trace of the high-end scheduler will generate a load that will be too much for the low-end scheduler to handle; due to the lack of feedback the submission rate will not decrease, and the simulation results will indicate extremely poor performance for the low-end scheduler, underestimating its true performance. Similarly, the high-end scheduler will handle the low-load trace of the low-end scheduler easily, but because the submission rate will not increase as happens in a real environment, the simulation will indicate very good performance for this scheduler, overestimating it's true performance.

To generate the two traces we used SiteSim to run a site-level simulation of 10 concurrently active users (ses-

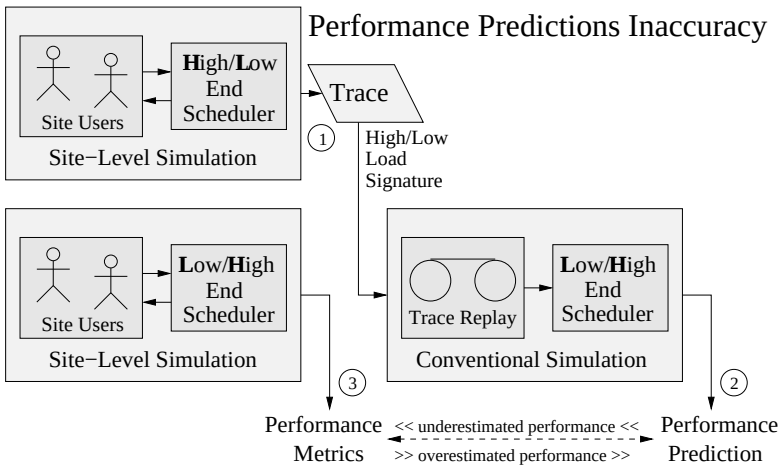


Figure 7. Experiment illustration: (1) Site level simulation generates a trace with a signature of one scheduler. (2) Conventional simulation using this trace predicts another scheduler's performance. (3) Site-level simulation of this other scheduler produces performance metrics used to quantify the prediction inaccuracy.

sions). For the high-end scheduler we used the EASY scheduler [11], which employs backfilling (executing jobs from the back of the queue) to reduce fragmentation and improve responsiveness. For the low-end scheduler we used FCFS (first-come-first-served). For the runtime estimates (required by EASY) we used the actual jobs runtime, that is, our estimates were perfectly accurate. The machine we simulated had 128 processors.

Underestimated Performance We ran a site-level simulation using the EASY scheduler as the high-end scheduler. We then ran a conventional simulation using the resulting trace, but this time we used the FCFS scheduler. The results of the simulation indicate very poor performance of the FCFS scheduler: over 24 hours on average for the jobs to respond, and 22.5 hours of waiting in the queue. Obviously, given these performance predictions, one would never consider using an FCFS scheduler, especially when comparing with EASY that achieves an average response of less than $2\frac{1}{2}$ hours.

Next, we repeated the site-level simulation for the same user population, this time using FCFS. The results indicate that FCFS actually performs reasonably considering its limitations; the jobs respond in $3\frac{1}{3}$ hours (just 39% more than EASY), and their mean wait is about $1\frac{1}{2}$ hours. The results for FCFS are still worse than those of EASY, but not as poor as predicted using the trace with the EASY signature. In fact, FCFS mean response time was overestimated by 634%, mean wait time by 1345%, and mean slowdown by 1332%! These results are summarized in Table 2.

Note that the comparison of EASY to FCFS when using a site-level simulation is no longer based on serving the same *jobs* (as in conventional simulations), but on serving the same *users*. As a result of the feedback FCFS actually served fewer jobs, but the difference was less than 10% for a load of 10 sessions. Throughput is further discussed in Section 6.3.

Metric (average)	EASY Site-lev.	FCFS Conv.	FCFS Site-lev.	Prediction inaccuracy
Response [s]	8571	87370	11897	634%
Wait [s]	2283	81082	5611	1345%
Slowdown	21.4	1127	78.7	1332%

Table 2. Underestimated performance.

Metric (average)	FCFS Site-lev.	EASY Conv.	EASY Site-lev.	Prediction inaccuracy
Response [s]	11897	7695	8571	-10%
Wait [s]	5611	1409	2283	-38%
Slowdown	78.7	17.3	21.4	-19%

Table 3. Overestimated performance.

Overestimated Performance Repeating the above experiment methodology in the opposite direction, we ran a 10 users, site-level simulation with the FCFS scheduler managing the machine, and then ran a conventional simulation on the generated trace, using the EASY scheduler. The results indicate excellent performance for EASY: two hours on average for the jobs to respond, and just 23 minutes of waiting in the queue. However, these performance predictions are actually far too good. A site-level simulation of EASY for the same user population indicates its mean response time was underestimated by 10%, the mean wait by 38%, and the mean job slowdown by 19%. These results are summarized in Table 3.

In summary, underestimated performance is much larger than overestimated performance. Also, the response time seems to be the least sensitive to performance prediction inaccuracy due to lack of feedback, and the waiting time the most sensitive; the slowdown is in between. This is because the wait time is the most direct measure of the system's effect on job performance.

6.2. Safe Load Scaling

One of the important features of a performance evaluation methodology is the ability to examine performance at different load levels. In the conventional methodology, load scaling is typically done by modifying the jobs' submission timestamps before replaying the traces in the simulation. By multiplying job submission timestamps from the original traces by a constant load scaling factor, the time between subsequent submissions either increases or decreases, depending on whether the factor is greater or smaller than one, respectively. This either decreases or increases the submission rate, and hence the load observed by the scheduler.

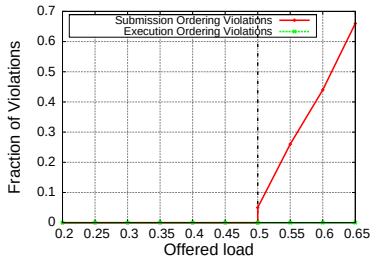
Modifying the original traces may effect the representativeness of the workload, by generating conditions which cannot exist in a real environment. One example is the violation of dependencies between job submissions. As explained above, users may wait for their jobs to complete before submitting more jobs, which means that there is a dependency between submissions; jobs that are dependent cannot reside together in the scheduler's queue.

When scaling the load by modifying jobs' submission timestamps, it may well occur that when replaying the trace, submission requests will be issued before the jobs they depend on have completed, and even worst — before the jobs they depend on even start executing. This raises the risk that the scheduler being evaluated will choose to execute jobs that depend on the completion of other jobs which still reside in the queue, totally violating the original job order.

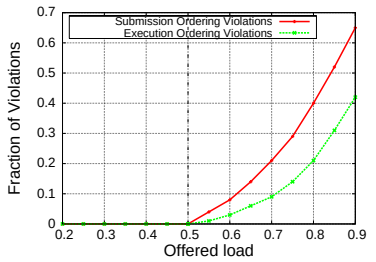
To quantify these effects, we used SiteSim to monitor submission dependency and execution ordering violations which occur during a conventional simulation. We ran two *site-level* simulations, both with 10 concurrently active user sessions. In the first simulation we used the FCFS scheduler, and in the second we used EASY. For each scheduler SiteSim generated a trace containing all job submission requests, their time-stamps, and dependency information. These traces are different because they include the signatures of the different schedulers.

We then ran conventional simulations using these traces. For each trace we simulated both the FCFS and EASY schedulers, at varying load levels ranging from 0.2 to 0.7 for FCFS and 0.2 to 0.9 for EASY (as EASY can sustain a higher load). We instrumented SiteSim to count the number of *submission dependency violations* — the number of times a job is submitted to the scheduler, but actually depends on the completion of a job that has not completed yet. We also count the number of times the scheduler chooses to start executing a job that depends on the completion of a job that still remains in the queue. We call the latter *execution ordering violations.*

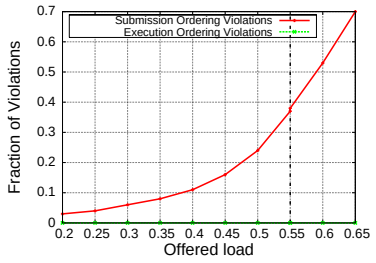
Figure 8 shows the fraction of jobs whose submission or execution involved violations. In all sub-figures, the dashed vertical line shows the original load in the trace, without any load scaling.



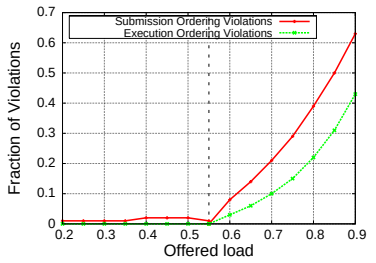
(a) FCFS trace / FCFS scheduler



(b) FCFS trace / EASY scheduler



(c) EASY trace / FCFS scheduler

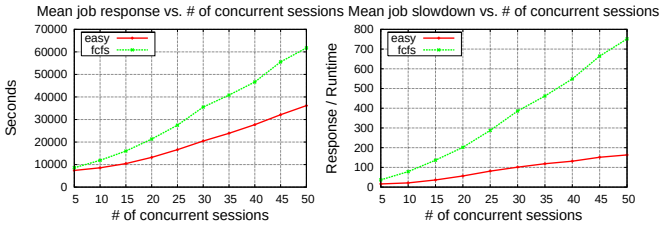


(d) EASY trace / EASY scheduler

Figure 8. Submission-dependency and Execution-ordering violations.

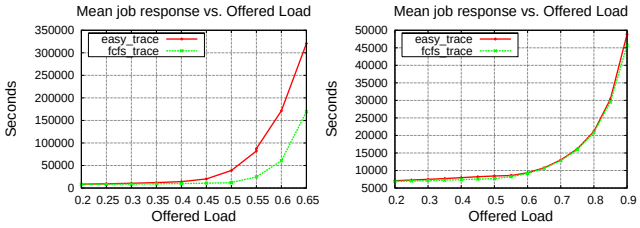
For the FCFS trace in sub-figures (a) and (b), we see that for both schedulers, the percentage of submission dependency violations starts to increase at 0.5 offered load[1] — the original load in the trace. For the FCFS scheduler the percentage of submission violations increases almost linearly, reaching 100% at 0.7 offered load — a load at which the simulated system is saturated. Obviously, because FCFS executes jobs according to queue order, there are no execu-

[1]The offered load is the load imposed on the system in an open model. The accepted load is what the system manages to handle, and may be lower than the offered load if it is saturated or requests are dropped.



(a) Mean job response (b) Mean job slowdown

Figure 9. Load scaling in Site-level simulations. The results for wait time are like response time, shifted down.



(a) FCFS scheduler (b) EASY scheduler

Figure 10. Load scaling in conventional simulations. The graphs for wait time and slowdown are very similar.

tion ordering violations.

For the EASY trace in sub-figures (c) and (d), we see that when using the FCFS scheduler the percentage of submission dependency violations starts to increase far before reaching the original load in the trace. The reason is that the EASY trace contains such a high-load signature, which is too much for the FCFS scheduler to handle, even if the load is scaled below the original load in the trace. In fact, some submission dependency violations occur even for the EASY scheduler under reduced load. There are no violations only at an offered load of 0.55 (the original load from the EASY trace), because at this point the lack of feedback of the conventional simulation has no effect on the representativeness of the workload.

For site-level simulations, load scaling is performed by simulating different numbers of users, which effectively changes the number of concurrently active sessions — increasing or decreasing the load. There is no problem with violating any dependency because the workload is dynamically generated which means that a submission which depends on the completion of a previous job will only commence after a that job has completed and following a period of think-time.

Figure 9 shows how the value of performance metrics changes, when the load is scaled for site-level simulations. As can be expected, the performance of EASY is always better than that of FCFS, but the more interesting phenomenon is the shape of the curves; instead of the curves

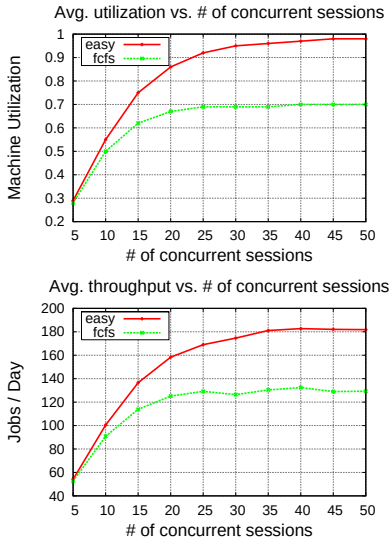


Figure 11. Utilization and throughput: implicit and explicit productivity measures.

often seen in open-system models, which tend to infinity when the load approaches the saturation point, see Figure 10, here the degradation in performance is much milder, due to the feedback that curbs the generation of additional work.

6.3. Quantifying Productivity

Increasing overall site productivity is a primary goal of any parallel system scheduler, but the conventional evaluation methodology lacks a metric for quantifying productivity. The only metric that correlates with productivity is the *average machine utilization* — the fraction of the machine that got utilized over its activity period. Intuitively, the larger this fraction is, the more work that was performed, implicitly indicating a higher productivity. However, in conventional simulations this is dictated by the rate new jobs are submitted, so it does not really reflect on the performance of the scheduler.

In contradistinction, our site-level evaluation methodology provides a metric that quantifies productivity directly: the system *throughout*, which is defined as the number of jobs processed in a given time frame. Figure 11 shows the throughput, measured as the average number of jobs executed in a 24 hours timeframe, for the FCFS and the EASY schedulers. For comparison, it also shows the utilization. The results indicate that the two metrics are highly correlated. They also show how the throughput levels out when the system becomes saturated. Beyond this point adding user sessions does not contribute to the throughput, but only increases the average response time. Also, the onset of saturation is gradual rather than being sharp as in conventional simulations.

7. Related Work

Most of the work on workload modeling for parallel supercomputers has been based on the open model, where jobs arrive at a given rate irrespective of how the scheduler handled previous jobs [10, 5, 4, 3, 12]. However, there has been some workload modeling work in other contexts that did involve feedback. One example is the study of gaming traffic [2]. Ganger and Patt observe the neither the open nor the closed model are satisfactory in their pure form, because real workloads are a mix with only some items being critical for progress [6]. This led to work by Hsu and Smith who added feedback to I/O traces, similarly to our own work [9]. It has also been suggested to try to use feedback in network design to avoid congestion [15].

Our model of sessions built of batches of jobs is related to other generative hierarchical workload models, which use several layers to try and mimic the process that generates the workload. Hlavacs et al. [8] presented a framework for modeling user behavior in interactive computer systems, using sessions, applications, and commands, which are initiated synchronously. An implementation of the framework for generating workload for network traffic simulation was presented in [7]. Arlitt [1] analyzed user sessions for the 1998 World Cup Web server. Zilber et al. [18] analyzed parallel systems traces and classified users and sessions based on their characteristics. Their work can be incorporated in ours by defining diverse submission behavior and workpool models.

8. Conclusions

We have shown that user sessions on parallel supercomputers can be modeled as a sequence of batches of jobs, where the jobs within each batch are submitted asynchronously, but each new batch is only started a certain time (the think time) after the last job in the previous batch completed. This imparts a measure of feedback on the workload generation process, leading to a better match between the workload and the scheduler's capabilities. Ignoring this feedback effect leads to exaggerated evaluations, that mix performance results related to the evaluated scheduler with results that are due to the scheduler that was used when the workload data was traced.

The simulations presented in this paper are limited to using a static number of active sessions. In a real site, users arrive and depart at different times, so the number of active sessions changes dynamically. A natural extension to our work would be to model the users population [16] and use that model in the simulation, to dynamically change the number of user sessions that are active at different times. Furthermore, not all user sessions generate similar workloads for the scheduler. Our next challenge is therefore to develop a library of workpools and submittal models, and combine them in various ways to produce sessions with different characteristics. Finally, our new methodology calls for new performance metrics. One such metric, the scheduler's throughput as a quantifier for the site's productivity, was introduced in Section 6.3. An interesting research direction is to find new metrics that measure the users' satisfaction of the scheduler performance even more directly.

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

Many thanks are due to all those who provided workload data to the Parallel Workloads Archive. In particular, we mainly used logs from the San Diego Supercomputer Center (SDSC), from the Cornell theory Center (CTC), and from the Royal Institute of Technology in Stokholm (KTH).

References

[1] M. F. Arlitt. Characterizing web user sessions. *SIGMETRICS Perform. Eval. Rev.*, 28(2):50–63, 2000.

[2] M. S. Borella. Source models of network game traffic. *Comput. Commun.*, 23(4):403–410, Feb 2000.

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

[4] A. B. Downey. A parallel workload model and its implications for processor allocation. *Cluster Computing*, 1(1):133–145, 1998.

[5] D. G. Feitelson. Packing schemes for gang scheduling. In *Job Scheduling Strategies for Parallel Processing*, pages 89–110. Springer-Verlag, 1996. LNCS vol. 1162.

[6] G. R. Ganger and Y. N. Patt. Using system-level models to evaluate I/O subsystem designs. *IEEE T. Comput.*, 47(6):667–678, Jun 1998.

[7] H. Hlavacs, E. Hotop, and G. Kotsis. Workload generation by modeling user behavior, 2000.

[8] H. Hlavacs and G. Kotsis. Modeling user behavior: A layered approach. In *Conf. Measurement & Simulation of Comput. & Telecommunication Syst.*, pages 218–225, 1999.

[9] W. W. Hsu and A. J. Smith. The performance impact of I/O optimizations and disk improvements. *IBM J. Res. Dev.*, 48(2):255–289, Mar 2004.

[10] 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*, pages 95–116. Springer Verlag, 1997. LNCS vol. 1291.

[11] D. Lifka. The ANL/IBM SP scheduling system. In *Job Scheduling Strategies for Parallel Processing*, pages 295–303. Springer-Verlag, 1995. LNCS vol. 949.

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

[13] J. Mache, V. Lo, and S. Garg. Job scheduling that minimizes network contention due to both communication and I/O. In *14th Intl. Parallel & Distrib. Proc. Symp.*, pages 457–463, May 2000.

[14] Parallel workloads archive. URL http://www.cs.huji.ac.il/labs/parallel/workload/.

[15] S. L. Scott and G. S. Sohi. The use of feedback in multiprocessors and its application to tree saturation control. *IEEE T. Parallel & Distributed Syst.*, 1(4):385–398, Oct 1990.

[16] D. Talby. *A User-Based Model of Parallel Workloads*. PhD thesis, Department of Computer Science, Hebrew Univeraity, in prep.

[17] A. Wong, L. Oliker, W. Kramer, T. Kaltz, and D. Bailey. System utilization benchmark on the Cray T3E and IBM SP2. In *Job Scheduling Strategies for Parallel Processing*, pages 56–67. Springer Verlag, 2000. LNCS vol. 1911.

[18] J. Zilber, O. Amit, and D. Talby. What is worth learning from parallel workloads?: a user and session based analysis. In *Proc. 19th Intl. Conf. Supercomputing*, pages 377–386, Jun 2005.