This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication.

On Simulation and Design of Parallel-Systems Schedulers: Are We Doing the Right Thing ?

Edi Shmueli and Dror G. Feitelson

***Abstract*— It is customary to use open-system, trace-driven simulations to evaluate the performance of parallel-system schedulers. As a consequence, all schedulers have evolved to optimize the packing of jobs in the schedule, as a mean to improve a number of performance metrics that are conjectured to be correlated with user satisfaction, with the premise that this will result in a higher productivity in reality. We argue that these simulations suffer from severe limitations that lead to sub-optimal scheduler designs, and to even dismissing potentially good design alternatives. We propose an alternative simulation methodology called site-level simulation, in which the workload for the evaluation is generated dynamically by user-models that interact with the system. We present a novel scheduler called CREASY that exploits knowledge on user behavior to directly improve user satisfaction, and compare its performance to the original, packing-based EASY scheduler. We show that user productivity improves by up to 50% under the user-aware design, while according to the conventional metrics, performance may actually degrade.**

***Index Terms*— Parallel job scheduling, trace-driven simulations, open-system model, user behavior, feedback.**

I. Introduction

An important goal of any parallel-system scheduler is to promote the productivity of its users. To achieve high productivity the scheduler has to keep its users satisfied and motivate them to submit more jobs. Due to the high costs involved in deploying a new scheduler, it is uncommon to experiment with new designs in reality for the first time. Instead, whenever a new scheduler is proposed, it is first evaluated in simulation, and only if it demonstrates significant improvements in performance, it then becomes a candidate for an actual deployment. The role of simulations is thus critical for the choices made in reality.

The conventional simulations presently used to evaluate the schedulers are trace-driven, and use an open-system model to play-back the trace and generate the workload for the evaluation. This means that new requests get issued during simulation solely according to the timestamps from the trace, irrespective of the system state, and implies that as long as the system is not saturated, the *throughput* of the scheduler being evaluated also gets dictated by the timestamps, instead of being affected by the actual performance of the scheduler.

- E. Shmueli is with IBM R&D Labs in Israel, Haifa University Campus, Mount Carmel, Haifa 31905, Israel, and with the School of Engineering and Computer Science, The Hebrew University, Jerusalem 91904, Israel. Email: edi@il.ibm.com.
- D. G. Feitelson is with the School of Engineering and Computer Science, The Hebrew University, Jerusalem 91904, Israel. Email: feit@cs.huji.ac.il.

The inability to influence throughput is an inherent problem in open models. In our case, job throughput is probably the best indicator for user productivity, but the metric simply cannot be used in the evaluation. The common solution is to use an alternative set of metrics which on one hand can be affected by the scheduler, and on the other be conjectured to correlate with user satisfaction. More specifically, the jobs' average *response-time* and *slowdown* are frequently used in evaluations, with the premise that improving them in simulation will result in a higher productivity in reality.

We argue that these simulations suffer from severe limitations, and that they lead the schedulers to focus on the packing of jobs in the schedule as a mean to improve the average values in simulation, which results in sub-optimal scheduler designs. We also argue that the conventional performance metrics do not necessarily correlate with user productivity, which may even result in dismissing potentially good design alternatives as poor.

As an alternative, we propose a novel simulation methodology named *site-level simulation*, in which the workload is generated *not* from traces, but dynamically by *user-models* that interact with the system, and whose behavior in simulation is similar to the behavior of users in reality. We claim that such behavior can be extracted directly from traces of systems, and with a level of detail that is sufficient to enable us to develop reliable models of the users to be used in the simulations.

Site-level simulations reproduce the fine-grained interaction that naturally exists between the users and the system in reality. This means that schedulers capable of motivating their users to submit more jobs will actually cause the throughput of the jobs in the simulation to increase, and implies that schedulers can be designed to improve user satisfaction directly, since their effect on productivity will be reliably evaluated.

We present such a scheduler and name it *CREASY*. Our scheduler inherits its backfilling algorithm from the original, packing-based EASY scheduler, but uses a novel prioritization scheme that exploits knowledge on user behavior to improve user satisfaction. It uses the fact that some jobs are more *critical* to the users than others (hence "CR" stands for CRiticality) in the sense that delaying them too much may cause their owners to leave the system. It assigns higher priorities to these jobs to reduce the likelihood for session aborts, and to motivate the users to submit more jobs.

We compare the performance of our scheduler, in simulation, to the performance of EASY, and show that user productivity improves by more than 50% under the user-aware design. We investigate the reason for this exceptional improvement and show that it stems from CREASY's ability to maintain long user sessions under high loads.

Digital Object Indentifier 10.1109/TPDS.2008.152

We also compare the two schedulers according to the conventional performance metrics and show inconsistent results: the average job response-time under CREASY is 27% higher compared to EASY, while the average slowdown is 66% lower. We show that the increase in response time is the outcome CREASY's tendency to prioritize short jobs at the expense of longer ones that dominate the average, and that the decrease in slowdown is the result of the exact same trade-off, and the fact that slowdown is affected mostly by the shorter jobs.

This paper is organized as follows. Section II provides the background: it gives examples of common schedulers designs, describes the conventional simulations and how they lead to these designs. Section III introduces site-level simulations: it describes our findings regarding the behavior of users in parallel systems, and the user models we use in our simulations which are based on these findings. Section IV presents CREASY and describes the simulation results. Section V surveys related work, and Section VI concludes the paper and suggests future research directions.

II. Shortcomings of Conventional Simulations

There are different types of parallel systems, and each requires a scheduler that is tailored to its own specific architecture. Though all schedulers are evaluated in simulation in a similar way, we chose to focus, without loss in generality, on a specific type of system that is both common and easy to describe.

Our system has a distributed memory model, in which every processor in the system is associated with a private memory, and the processors are connected to each other using a fast network. A parallel job in such a system is a unit of work that is composed of multiple processes that need to execute in parallel and communicate over the network.

There is no time-sharing nor preemption support in our system. This means that processors need to be allocated to the jobs using a one-to-one mapping — one processor for every process of the job, and once allocated they remain dedicated to the job until it terminates. This scheme is often referred to as space-slicing.

The role of the scheduler in such a system is to accept the jobs from the users, to allocate processors and to execute the jobs on the selected processors. For simplicity, we ignore issues like network contention, heterogeneous node configurations, and security.

The system users submit their jobs by providing job descriptions to the scheduler. For our type of system this typically includes two important attributes: the number of processors the job requires in order to execute, which is often referred to as the job's *size*, and an estimated upper bound on the runtime of the job, to enable the scheduler to plan ahead.

A. Common Scheduler Designs

The behavior of the schedulers upon job arrival differ greatly. Most schedulers maintain a queue where the jobs wait for processors to become available [1], [2]. Whenever the state of the system changes, either due to an arrival of a new job, or a termination of a running job, they scan the queue and select jobs for execution. Some schedulers maintain a number of queues and use, for example, the job's runtime estimates to select the right queue for the job [3]. Other schedulers maintain futuristic execution profile for the jobs; when a new job arrives, they insert it into the profile in a location where it either does not conflict with any of the already existing jobs [4], or in a place where it delays some of these jobs by a small factor [5].

It is difficult to determine which approach is the best, and in fact some studies have indicated that the relative performance of schedulers may actually depend on the workload [4]. On the other hand, there is one thing that *all schedulers* share in common: they all focus on the packing of jobs in the schedule, which as we demonstrate below, may not be optimal for productivity.

Consider for example a loaded system, and three users numbered 1, 2 and 3, who submitted three jobs to their scheduler at 11:00am, 11:10am, and 11:55am, respectively. Assume that the time is 12:00pm and that neither of these jobs had started executing yet. By this time, there is a high probability that users 1 and 2 have given up waiting for their jobs and that they have left the system already. On the other hand, there is a good chance that user 3 who had just submitted his job is still active at the system, and is excepting a fast response.

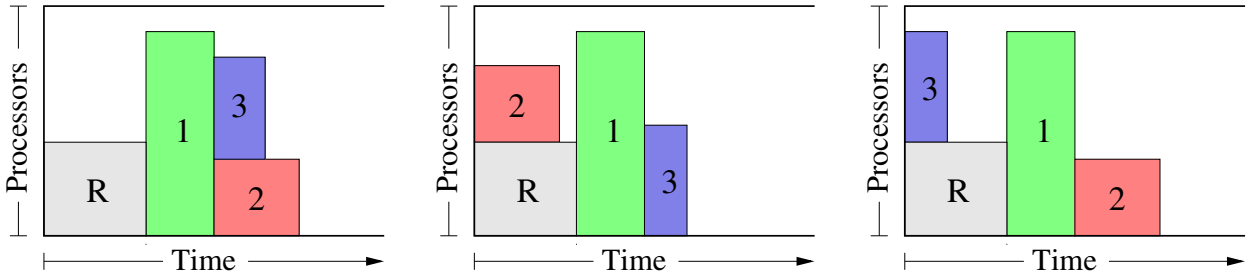
Figure 1 illustrates how three different schedulers would have treated these jobs. In all sub-figures, the system processors are laid out vertically, and time is running from left to right, starting at 12:00pm. Our three jobs are labeled 1, 2 and 3, after their users. There is also one more job that is labeled R and is currently running, and enough free space beside that job to accommodate job 2 or 3, but not job 1.

The simplest scheduler, First-Come-First-Served (FCFS) in Figure 1(a), would simply execute the jobs in their arrival order. Since job 1 must wait for job R to terminate before it can start executing, a large space at the beginning of the schedule remains un-utilized. Jobs 2 and 3 will start executing together under FCFS, but only after job 1 terminates.

The problem with FCFS is of course the poor system utilization. This led to the development of a new class of schedulers that relax the strict execution order of the jobs to improve utilization. When the jobs reside in a wait queue in their arrival order, such schedulers pick small jobs from the *back* of the queue, and execute them before larger jobs that arrived earlier, to *fill* holes in the schedule. This behavior was given the name *backfilling*.

Backfilling can be implemented in different ways. Figure 1(b) illustrates the schedule under the EASY scheduler — a classic backfilling scheduler that was originally developed for the IBM SP parallel system, and is used ever since as a reference for performance comparison in virtually any job scheduling research [1].

EASY prioritizes the waiting jobs according to their arrival order, and uses the jobs' runtime estimates to calculate when the highest priority job — the earliest arriving job — will be able to execute in the future. It then examines the remaining jobs in descending priority order, and backfills any job that fits into the currently free processors, as long as it will not conflict with the projected execution of the highest priority job.



(a) FCFS Schedule (b) EASY Schedule (c) User-aware Schedule

Fig. 1. Three different schedules for the jobs: (a) Poor system utilization under FCFS, (b) Improved utilization but not user-optimal schedule under EASY, and (c) User-aware schedule to motivate user 3 to submit more jobs.

Concentrating on the highest priority job is done to guarantee the execution of all jobs: once this job starts executing, the next earliest-arriving job will become the highest priority job, and it also will no longer be delayed.

In our example, job 1 is the earliest arriving job, so EASY determines that it will be able to execute only after job R terminates. It then examines job 2 that has the second-highest priority, and backfills that job since it will not conflict with the execution of job 1. Finally it examines job 3 and determines that there are not enough free processors to backfill that job too. Job 3 will therefore be delayed to a later time, and execute only after job 1 terminates.

At first glance it seems that EASY's schedule is optimal: the space beside job R has been utilized by job 2, and job 1 will execute without delay — but this is just an impression that is based on a static view of the system. The problem is that by the time job 3 will terminate, there is a high probability that user 3 will give up waiting for it and leave the system. In other words, EASY backfilling may be apparently good for utilization, but it is not optimal for the users.

Figure 1(c) illustrate a user-aware schedule in which job 3 is backfilled before job 2, although it has arrived last. The idea is to get job 3 to respond while its owner is still active at the system, to motivate user 3 to continue the interaction and submit more jobs. Though initially it seems less intuitive, this schedule is in fact based on the anticipated dynamics of the system and speculating about future user behavior, and should result in a higher productivity.

B. Simulations Effect on Design

Though it is clear from the above example that scheduling jobs without considering the users might not be optimal, virtually all schedulers would backfill, similar to EASY, job 2 ahead of job 3. We argue that the reason they do not explicitly consider the users is rooted in the way the conventional simulations are carried out to evaluate the performance of the schedulers.

In these simulations, the workload is usually generated from *traces* that contain records of jobs that were submitted to real, production-use parallel systems over long periods of time. Each record in the trace contains several attributes that describe a job, and includes a *timestamp* that indicates when the job was originally submitted.

There are two models for actually generating the workload from the trace, the *closed-system* model, and the *open-system* model. The closed model ignores the timestamps and issues new requests only after a previous job completes. The problem is that it leads to extreme regularity: there are no bursts of activity in the workload which severely limits the optimizations that can be performed by the scheduler, and there is no easy way to manipulate the load for the evaluation.

The open model on the other hand plays-back the trace solely according to the timestamps, and issues new requests irrespectively of the system state. It supports bursts as imposed by the timestamps, and the load can be easily manipulated by modifying the timestamps in the trace before the simulation begins. Since real workloads often exhibit bursts and varying load conditions, the conventional simulations adopted this model in generating the workload, but the choice is more of a compromise than an optimal selection, and it even seems to have affected the way schedulers are designed.

In open-system simulations, as long as the system is not saturated, the *throughput* of the scheduler that is being evaluated gets dictated solely by the timestamps from the trace, and it is *not* affected by actual performance of the scheduler. A scheduler capable of motivating its users to submit more jobs will not cause more jobs to be submitted, and an inefficient scheduler that ignores its users and causes them to leave the system will not decelerated the creation of additional work.

This inability to influence throughput is an inherent problem in open models in general. In our case, job throughput is probably the best indicator for user productivity, and improving it should therefore be an important goal for any parallel-system scheduler, but the metric simply cannot be used in the evaluation. The common solution is to use an alternative set of metrics which on one hand can be affected by the scheduler, and on the other be conjectured to correlate with user satisfaction. More specifically, the jobs' average *response-time* which is the time the jobs spent in the system from submission to termination, and their *slowdown* which is the response time normalized by the actual runtime of the job, are frequently used in evaluations. The premise is that improving them in simulation will result in a higher productivity in reality.

Consequently, all schedulers evaluated using the conventional simulations have evolved to consider the user of the system only implicitly by trying to improve these metrics. They often try to optimize the packing of the jobs in the schedule, since tighter packing usually leads to lower average values. We are not aware of any parallel-system scheduler

TABLE I

METHODOLOGICAL DIFFERENCE BETWEEN THE TWO TYPES OF SIMULATION.

Category	Conventional Simulations	Site-Level Simulations
Workload source	System traces	User models
Workload generation	Open-system model	Users-scheduler interaction
Load scaling	Trace (de)-compression	Number of users
Performance metrics	Response time, slowdown	Throughput, session length

that considers its users explicitly, nor of any investigation as to whether these seemingly "user-friendly" metrics indeed correlate with higher productivity.

III. SITE-LEVEL SIMULATIONS

As described above, the conventional simulations lead to the design of schedulers that consider the system users only implicitly. To enable the design of truly user-aware schedulers, we propose to change the way simulations are carried out: instead of using traces to generate the workload, we suggest to model the users of the system, and use these models in simulation to *dynamically* generate the workload for the evaluation. We name these simulations, naturally, *site-level simulations*.

Site-level simulations reproduce the fine-grained interaction that naturally exists between the users and their system in reality. This means that schedulers capable of motivating the users to submit more jobs will actually cause the throughput of the jobs to increase, and implies that schedulers can be designed to improve user satisfaction directly, since their effect on productivity will be reliably evaluated. Table I summarizes the methodological difference between the two types of simulations.

The basic and most important elements in a site-level simulation are the *user-models*, and in fact our entire methodology depends on the ability to understand the behavior of users, and to capture this behavior in a model that can be used in simulation. One of the major contributions of our work is the analysis methodology we developed, that enabled us to uncover the users' behavior patterns directly from systems traces, without conducting live experiments with real users. In the following section we briefly describe our findings and focus only on those that directly pertain to our user models. The complete methodology is described in [6].

These findings form the basis for the *session dynamics model* described below, which is the first of three models that together comprise the complete user model we use in our simulations. The dynamics model handles the dynamic aspects in the user behavior — the starting and the ending of user sessions as a reaction to the performance of their jobs. The other two models are the *job submission model* that handles the actual submission of jobs during the sessions, and the *activity cycles model* that incorporates daily and weekly cycles into the simulation.

Our user model is described in Section III-B. We implemented and integrated it into *Site-Sim* — a framework we developed for site-level simulations to enable the reliable evaluation of user-aware schedulers. We used Site-Sim extensively to explore design alternatives as we developed CREASY — the first truly user-aware parallel-system scheduler described in Section IV. The simulation results of our scheduler reported in that same section were also obtained using Site-Sim.

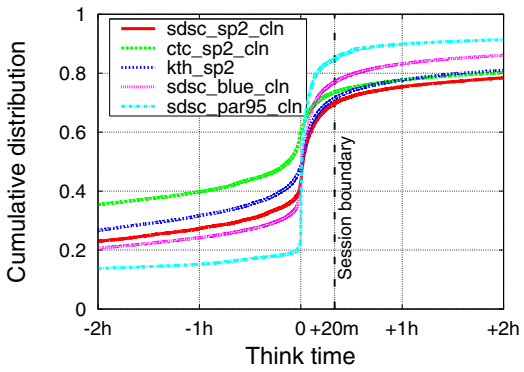


Fig. 2. CDF of think times in the five traces: negative values indicate that sometimes users submit jobs without waiting for their previous jobs to terminate. The steep climb in all curves which levels-off at about twenty minutes lead to defining the sessions think time threshold to be twenty-minutes.

TABLE II

THE FIVE TRACES WE USED FOR OUR ANALYSIS: TOGETHER, THEY REPRESENT MANY YEARS OF ACTIVITY BY HUNDREDS OF USERS.

Trace	Duration	Users	Jobs
SDSC-Par-1995-2.1-cln	1/1995–12/1995	98	53,970
CTC-SP2-1996-2.1-cln	6/1996–5/1997	679	77,222
KTH-SP2-1996-2	9/1996–8/1997	214	28,489
SDSC-SP2-1998-3.1-cln	4/1998–4/2000	437	59,725
SDSC-BLUE-2000-3.1-cln	4/2000–1/2003	468	243,314

A. User Behavior Patterns

In reality, users tend to submit several jobs one after the other in periods of activity that are known as *sessions*. The time between the termination of a job and the submission of the next is globally known as the *think time*, but the fact is that if the think time is too long, it may actually indicate a break which is not part of the session. The question is therefore what is the think time threshold that separates jobs that belong to the same session from those that belong to the next.

Zilber et al. answered the question by simply observing the distribution of the think times in different traces of parallel systems [7]. Figure 2 shows the CDF of the think times in five of these traces[1], which are listed in Table II. Two important observation can be made on this figure. The first is that think times can be negative, which means that sometimes users submit jobs without actually waiting for their previous job to terminate.

[1]All traces are available from the Parallel Workloads Archive at URL http://www.cs.huji.ac.il/labs/parallel/workload/.

The second observation is the steep climb in all curves at zero, which starts to level off at around twenty minutes. This means that a large portion of the jobs are submitted within twenty minutes from the completion of a previous job, and that beyond twenty minutes the think times are evenly distributed, without any features indicating a natural threshold. Zilber et al. therefore defined the threshold to be *twenty minutes*; above twenty minutes the think times are considered breaks, and the jobs that follow them are considered to belong to the next session.

In our work we adopted this definition, but also tried to understand what may cause the users to continue their sessions or to take breaks. We found that in all traces, there is a strong correlation between the response times of the jobs and the think times: the longer the response, the higher the think times that follow the jobs. This led us to speculate that user behavior is affected by the response times of their jobs — that short response times encourage the users to quickly submit more jobs, and that longer ones may cause them to abort their sessions. Similar observations regarding the relation between job response and user behavior were reported through the use of live-experiments in [8].

Due to the large variance that naturally exists in the traces, we divided the jobs into classes according to their response times, and for each class we calculated the percentage of jobs that were submitted below the twenty minutes think time threshold. The result was a mapping between the response times of the jobs and the probability for the users to continue their sessions, which indicates that the longer the response the lower the probability for the users to continue and submit more jobs. The mapping is illustrated in Figure 3(a) and it forms the basis for session dynamics model described in Section III-B.1.

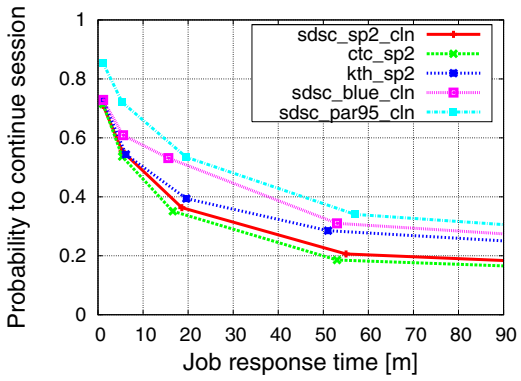
It is important to note that the response times of jobs is only one of many factors that affect the users, and that user behavior in reality is far more involved than what our current model depicts. However, for the purpose of demonstrating the effect of simulations on the design of the schedulers, our simple models suffice. In the conclusions section we provide suggestions as to how to enhance the models to further improve the accuracy of the evaluation.

B. Complete User Model

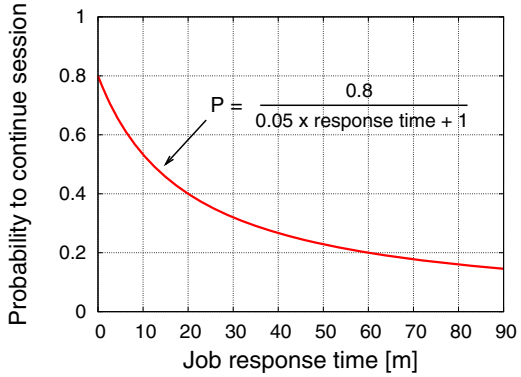
Our user model is composed of three sub-models that interact with each other during simulation to simulate a realistic user behavior. The *session dynamics model*, the *job submission model*, and the *activity cycles model* are described in detail in the following sections. In section III-B.4 we provide examples as to how these models interact during simulation.

1) Session Dynamics Model: As described above, one of the important factors that affect user behavior is the response times of their jobs: the longer the response, the lower the probability for the users to continue their sessions. This means that response times in effect, affect the users' decision to continue or abort their interactive sessions with the system.

There are two reasons why it is extremely important to accurately model this decision. First, it is an integral part in the behavior of users, representing their satisfaction with the performance of the system. Second, since the length of the sessions directly affects the throughput metric, schedulers can try to influence this decision as a mean to improve productivity. In other words, the accurate modeling of this decision is essential for both the evaluation and the design of user-aware schedulers.



(a) Original trace data



(b) Data combined and curve smoothened

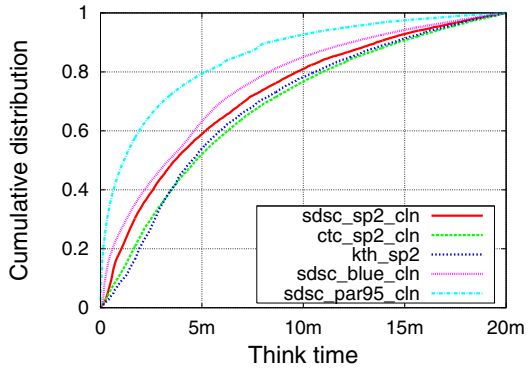
Fig. 3. Jobs response time effect on user behavior: (a) In all traces the longer the response, the lower the probability for the users to continue their sessions, and (b) Trace data combined and resulting curve smoothened.

The session dynamics model is responsible for taking these decisions for the user models during simulation, and based on the outcome to determine when will they submit more jobs to the system. In its essence, the dynamics model handles the dynamic starting and the ending of user sessions during simulation.

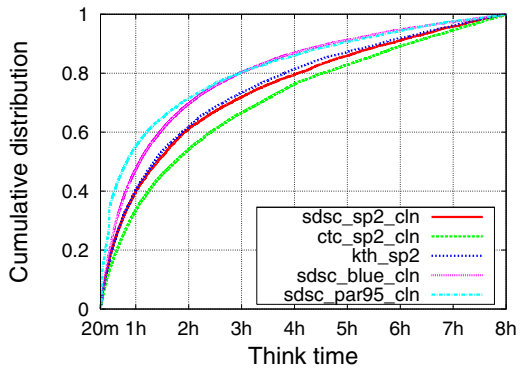
To model the decision, we first combined the data from all five traces of Figure 3(a) and smoothened the resulting curve, as shown in Figure 3(b). We found that the curve can be roughly described by Equation 1. Next, during simulation whenever job j terminates, we calculate the response time of the job, and use Equation 1 to determine the probability $p\_cont(j)$ that the user who submitted the job will continue his session with the system.

$$p\_cont(j) = \frac{0.8}{0.05 \times resp\_time(j) + 1} \quad (1)$$

To make the final call we perform a single Bernoulli trial, with probability $p\_cont(j)$ for success and $1 - p\_cont(j)$ for failure. If the trial ends in a success, the user will continue his session with the system, otherwise he will take a break. The trial is summarized in Equation 2.

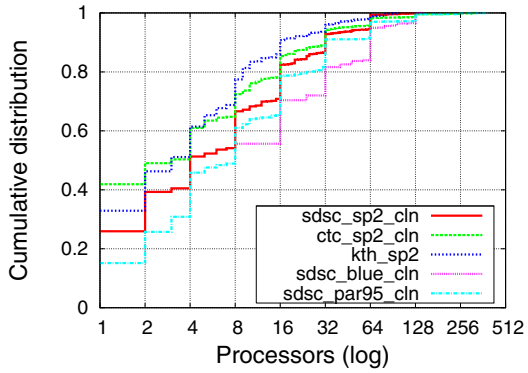


(a) Think times $< 20m$

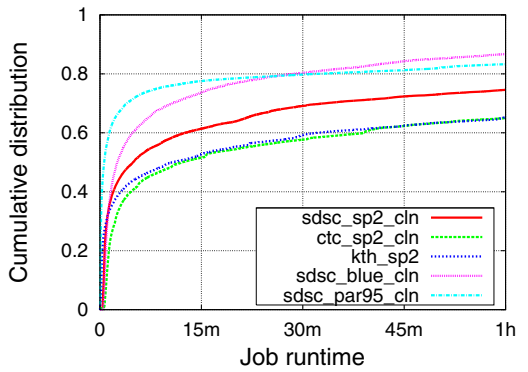


(b) $20m <$ Think times $< 8h$

Fig. 4. The two think time distributions in the traces: (a) short think times are used for sessions that continue, and (b) longer think times are used for breaks.



(a) CDF of jobs' sizes



(b) CDF of jobs' runtimes

Fig. 5. CDF of job sizes and runtimes in the traces: (a) sizes is a modal distribution with most jobs using power-of-two processors, and (b) runtimes is rather skewed distribution, dominated by small runtime values.

$$decision = \begin{cases} \text{continue session} & \text{with probability } p\_cont(j) \\ \text{abort session} & \text{with probability } 1 - p\_cont(j) \end{cases} \quad (2)$$

Once we know whether its session continues or not, the next step is to determine when will the user submit his next job. As described above, jobs within the same session are submitted with up to twenty-minutes of think time from the completion of a previous job, whereas between sessions the think times are longer and are considered breaks. We therefore need two distributions: one with short think times to be used for sessions that continue, and the other with longer think times to be used for breaks.

We used distributions that are based on empirical data we extracted from the same five traces of Table II. In these traces, breaks may sometimes be as long as several months, since real users do not necessarily use the system continuously throughout the year. To avoid such long pauses in user activity during simulation, we limited the breaks to a maximum of eight hours by filtering-out longer think times during trace analysis. The two distributions as they appear in the traces are shown in their CDF format in Figure 4. For the simulations, we combined the data from all five traces into a single representative distribution.

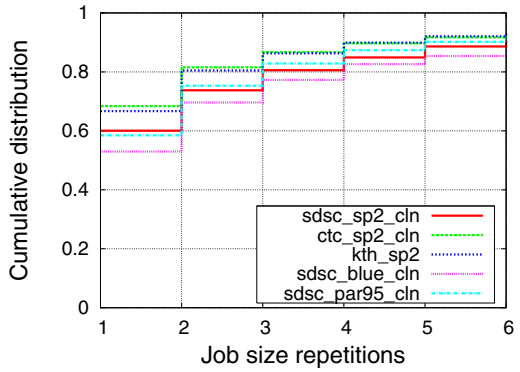
2) Job Submission Model: The session dynamics model described above does not handle the actual submission of jobs. This is the role of the job submission model: it generates the attributes for the jobs, and submits the jobs to the scheduler in a realistic manner.

To generate the attributes, we once again used distributions that are based on empirical data from the traces. The CDFs of the job sizes and runtimes are shown in Figure 5. The first is a modal distribution with most jobs using power-of-two processors, and the second is a rather skewed distribution dominated by small runtime values, usually in the order of a minute or less. Similar observation regarding size and runtimes were reported in several studies [9].

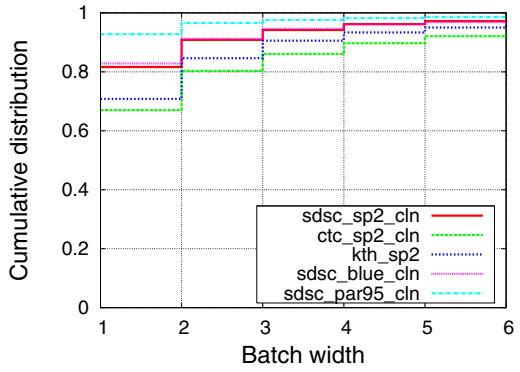
Though the above distributions are based on empirical data, using them "as-is" will still not generate a truly realistic workload. The reason is that in reality, users tend to submit the same jobs over and over again, which means that successive jobs by the same user tend to be similar to each other. This temporal locality in the workload will therefore be lost if we simply sample these distributions in the course of simulation.

The solution is to use a two-level sampling process, with the top level generating the attributes for the jobs[2], and the bottom level repeating them to generate effects of locality [10]. For the bottom level, we chose the jobs' *sizes* to be the leading distribution, and extracted the number of times jobs of the same size appear successively in the traces. The CDF of size repetitions in the different traces is shown in Figure 6(a). Again, we combined all traces into a single representative distribution for use in the simulation.

[2]Further accuracy can be achieve be considering the correlation between size and runtime.



(a) Size repetitions



(b) Batch widths

Fig. 6. CDF of job size repetitions and batch widths in the traces: for the simulations, we combined the data from all five traces into a single representative distribution.

To actually submit the jobs, we closely examine Figure 2 and observe that a large fraction of the think times in the traces — more than 50% in some cases — are in fact negative. This stems from the definition of think time as the time from the termination of a job to the submission of the next, and indicates that sometimes users submit jobs without waiting for their previous jobs to terminate.

An effective way to model this behavior is to use *batches* which are groups of jobs submitted asynchronously to one another, without being affected by the performance of previous jobs. Sessions will thus consist of series of one or more batches, each containing one or more jobs, and the session dynamics model described above will only be used to derive the think time from the last job in a batch, to the first job in the following batch. The relationship between sessions, batches and think times is illustrated in Figure 7. The jobs marked with an *X* are those used to derive the think time.

The CDF of the *width* of the batches — the number of jobs submitted asynchronously within batches — is shown in Figure 6(b). As can be seen, the distributions are reasonably similar in all traces which indicates that our data is representative of job submission behavior in general.

3) Activity Cycles Model: Daily and weekly cycles are universal human traits. Most users arrive to work in the morning and leave for home in the evening. Normally, they work during week-days, and rest over weekends. Incorporating these cycles of activity in the simulation is important, not just because they constitute a fundamental characteristic of real workloads, but also since they introduce periodic intervals of low loads that enable the scheduler to stabilize the state of the system and prepare for the next interval of high load [11].

Figure 8(a) shows the distribution of job submissions during the 24-hours *daily* cycle in the traces. Not surprisingly, all traces indicate higher levels of activity during the daytime compared to the nighttime. What is interesting though is the high level of similarity among the traces, which in fact enables us to roughly define a boundary between day and night. Accordingly, we defined daytime to be from 7:30am to 17:30pm, and nighttime from 17:30pm to 7:30am the next morning. Our analysis indicates that approximately 70% of all job are submitted during the 10 hours of daytime, and the reset during the nighttime.

Similarly, Figure 8(b) shows the distribution of submissions during the *weekly* cycle. As expected, weekdays Monday to Friday are busier than weekends, accounting for 80% of all submissions. The remaining 20% occur during the weekends, Saturday and Sunday.

The role of the activity cycles model is to incorporate these daily and weekly cycles into the simulation. At simulation start, it performs two Bernoulli trials for each user model: the first to determine whether the user will be active during the day or the nighttime, and the second to determine its days of activity — weekdays or weekends. The probabilities we used in these trials are 70% and 80%, respectively. This effectively divides the user population into four classes: (a) daytime-weekdays, (b) daytime-weekends, (c) nighttime-weekdays, and (d) nighttime-weekends, and guarantees that the levels of activity in the simulated workload will be similar to those found in reality.

The model then continuously monitors the time of day and the day of week during the simulation, and determines for each user model, based on its class, whether it should continue to be active or be temporarily suspended. For the *daytime-weekday* users for example, if a job terminates after 17:30pm, the model will determine it is sleep time for these users, and suspend their activity until the next morning, or even until the next weekday, if it is already a weekend.

To prevent bursts of activity at shift transition, the cycles model also attaches a random number between -60 and 60 to each user model, and uses this number to personalize the user's window of activity. For example, if the number 20 was attached to a certain daytime user, the cycles model will shift its window of activity by 20 minutes from the "official" daytime window. This means that the user will submit his first job at 7:50am and be suspended at 17:50pm.

4) Models Interaction During Simulation: The three models described above interact with each other in order to simulate a realistic user behavior. We provide two examples for this interaction, both for the *daytime-weekday* users. The first happens entirely during the day, and demonstrate how sessions start and end dynamically during simulation. In the second example, the cycles model intervenes, and suspends the user until the next morning. In both cases, we assume the user only submits a single job at a time.

The first example is illustrated on the left side of Figure 9. Our user arrives to work at 7:30am sharp, and the job

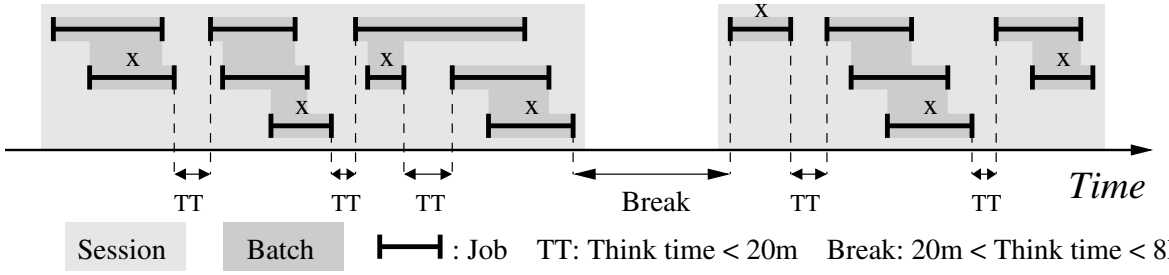
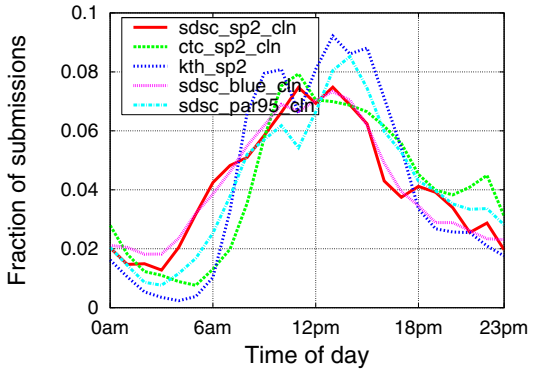
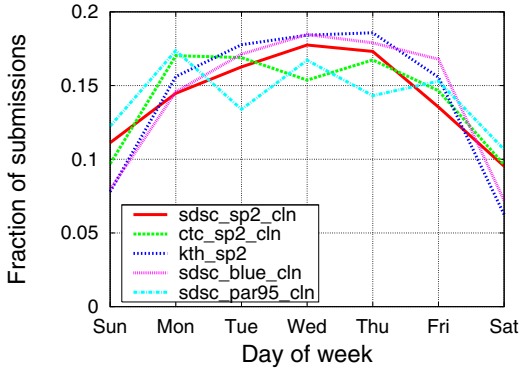


Fig. 7. Sessions, batches, and think times: the jobs marked with an *X* are those used by the session dynamics model to derive the think time until the next job.



(a) Daily Cycles



(b) Weekly Cycles

Fig. 8. Daily and weekly cycles in the traces: (a) 70% of the jobs are submitted during daytime, 7:30am to 17:30pm, and (b) 80% of them are are submitted during week-days, Monday to Friday.

submission model is immediately called to submit the first job to the scheduler. When the job responds five minutes later, the activity cycles model is called to determine whether the user is still active at work. Since 7:35am is just the beginning of the workday for our daytime-weekday user, the session dynamics model is called to determine if its session should continue or not.

The dynamics model determines that five minuets of response are satisfactory, and decides on a short think time of 10 minutes following this job. Ten minutes later at 7:45am, the activity cycles model is called once again to verify the time, and the submission model is called to submit the second job by the user. When this job responds 10 minutes later, the cycles model verifies the time again, and the dynamics models decides on a 15 minutes think time until the next job.

At 8:10am our user submits the third job to the scheduler. This time, the job responds after a whole hour, so the dynamics model determines that the session should *not* continue, and decides on a long, three hours break for the user. Three hours later, at 12:10pm, the cycles model verifies the time once again, and our user submits the fourth job, and so forth.

The second example is illustrated on the right side of the figure. Our user submit a job at 17:10pm that responds five minutes later. The time is verified, and the dynamics model decides on a think time of 10 minutes until the next job. At 17:25pm our user submits one more job that responds at 17:35pm. This time the cycles model determines that it is late for the user, and send him on a long sleep of 13 hours and 55 minutes, until 7:30am the next morning.

IV. User-Aware Scheduling

Site-level simulations allow user-aware schedulers to be reliably evaluated and effectively designed. We developed such a scheduler and compared its performance, in simulation, to the original EASY scheduler which is not user-aware. We present our scheduler and its simulation results in the following sections.

A. Criticality of Jobs

Our scheduler is similar to the EASY scheduler from Section II-A in the sense that they both use backfilling to improve performance. Furthermore, our scheduler actually *inherits* its backfilling algorithm from the EASY scheduler. In fact, the only difference between the two schedulers is in the way they prioritize the waiting jobs: while EASY accounts only for the jobs' arrival order in the interest of fairness [12], our scheduler tries to assess the *criticality* of the jobs for the users, and assigns its priorities accordingly. We therefore named our scheduler CREASY, with "CR" standing for CRiticality, and "EASY" to denote the backfilling algorithm internally used.

The criticality of a job is determined by the way it affects the behavior of its owner. We already know that user behavior is affected by the response times of the jobs. A closer look at Figure 3 also reveals that the mapping between the response times and user behavior is non-linear: the probability for users to continue their sessions drops rapidly as response times increase for short response times, and continues to drop more slowly for higher response times.

This means that jobs with short response times are *much more critical* to the users in the sense that any delay incurred by these jobs, even the smallest one, dramatically increases the

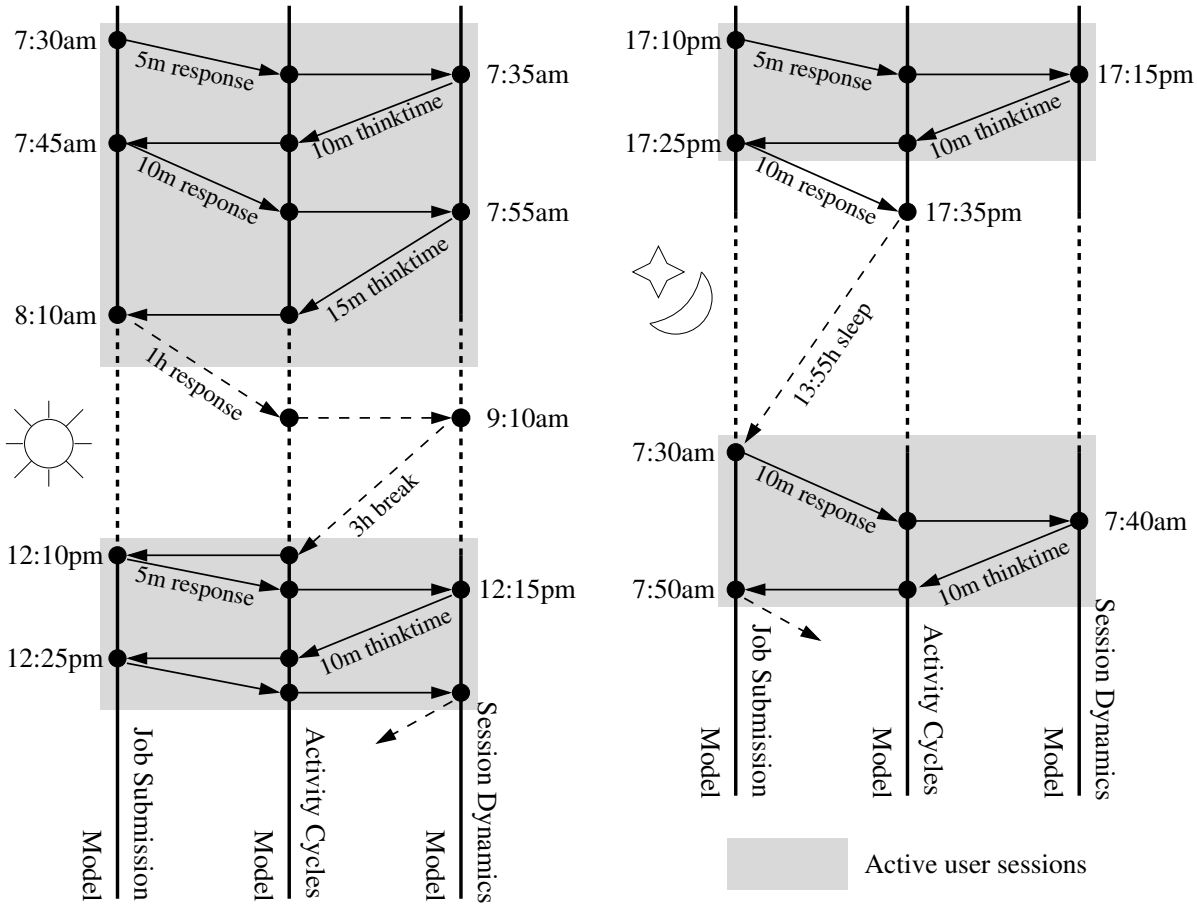


Fig. 9. Two examples for the models interaction during simulation: active user sessions are shown in the dark gray.

chances for a session abort. We therefore defined the criticality of jobs using Equation 3, which is the *derivative* of Equation 1 in absolute values, and hence accurately accounts for these differences in criticality: it assigns high values to jobs with short response times, and near-zero values to those whose effect on user behavior is marginal. This differs from plain shortest-job-first scheduling in the sense that short jobs are given high priorities only provided that they have not been delayed too much.

$$criticality(j) = \frac{0.04}{(0.05 \times estimated\_response\_time(j) + 1)^2} \quad (3)$$

Note that in the denominator of Equation 3 we only use an *estimate* for the response time of the job, since exact response times can only be determined after the jobs terminate. For the estimate we sum the time the job had already spent waiting in the scheduler's queue, and the time it is expected to run, which is based on the user estimate. Together, the two values represent the total time the job is expected to spend in the system, from submission to termination.

If Equation 3 will be used to prioritize the jobs, it will increase the chances for critical jobs to execute before other jobs, which should reduce the likelihood for sessions abort, and motivate the users to submit more jobs. The problem is that this is *not* enough, because the EASY algorithm internally used to backfill the jobs can guarantee the execution of all jobs *only* if *every* waiting job will eventually become the highest priority job.

While this is true under EASY's original prioritization scheme, it is not guaranteed in our case since according to Equation 3, senior jobs whose response time is already long will never become more critical than short executing jobs that keep getting submitted. In other words, the combination of criticality-based prioritization and EASY backfilling may lead to starvation.

The solution is to combine a *seniority* factor in the priority calculation, as shown in Equation 4: the criticality part on the left is taken directly from Equation 3, and the seniority factor is simply the time, in minutes, that the job is waiting for execution in the scheduler's queue. Finally, the weight α is used to set the relative importance of the two factors in the calculation, and at the same time to adjust the different units used.

$$priority(j) = \alpha \times criticality(j) + seniority(j) \quad (4)$$

If $\alpha = 0$, jobs will be prioritized solely according to their seniority, resulting in a prioritization scheme which is effectively *identical* to EASY's original scheme. Non-zero α values on the other hand will cause the criticality factor to take an increasing effect, and performance to improve as we demonstrate below. However, since the largest possible value of the criticality factor according to Equation 3 is 0.04, and the seniority of jobs steadily increases with time, it is guaranteed that for any α value that we choose, senior jobs will eventually

reach higher priorities, and their execution will be guaranteed by the EASY algorithm.

B. Simulation Results

We used Site-Sim to run site-level simulations of CREASY, and compared its performance to the performance of the original EASY scheduler. As described above, setting α to 0 in Equation 4 results in a prioritization scheme which is effectively identical to EASY's original scheme, which means that the behavior of the two schedulers becomes identical. We therefore didn't even need to explicitly implement EASY — we simply simulated it using CREASY.

We found that α values of 1500, 3000, 4500 and 6000 have a noticeable and a progressive effect on the performance of our scheduler. When $\alpha = 1500$ its performance is closest to EASY's, and beyond 6000 changes in performance are marginal. In total, we experimented with five schedulers: the original EASY scheduler (simulated using CREASY with $\alpha = 0$), and the four variants of CREASY, each with one of the above non-zero α values.

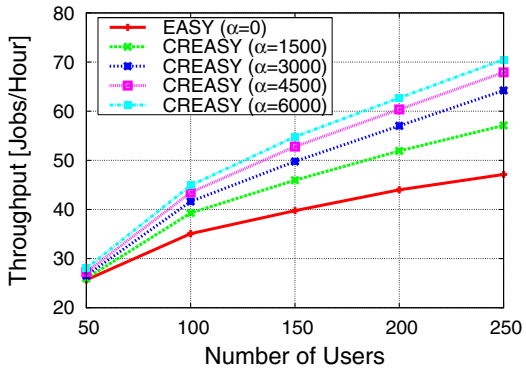
To compare the performance of the schedulers under different loads we ran five simulations of each scheduler, using a different number of users models in each run. We used 50 users to simulate low loads and gradually increased the size of the site by adding 50 users each time, until we reached 250 user models. In each run we simulated six months of user activity, which produces enough data to allow us to base our conclusions on statistically significant results. We also compared key attributes of the resulting data to their original distributions from the traces to validate the correctness of our simulations.

1) User Productivity: Improving productivity is an important goal for any parallel system scheduler, and the best indicator for user productivity is the throughput — the number of jobs the users submit to the system over a period of time.

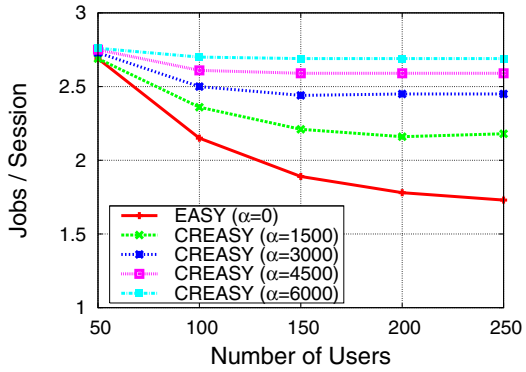
Figure 10(a) shows the average job throughput under the five schedulers as a function of the size of the site. As seen in the figure, for the smallest site of 50 users, all schedulers perform similarly since the load is too low for any optimization to take effect. Only when the load begins to increase, the differences in performance become noticeable.

For the largest site we simulated, of 250 users, the throughput under the EASY scheduler is 47 jobs/hour, while under CREASY with $\alpha = 6000$ it is 71 jobs/hour which is an exceptional improvement of more than 50%. Improvement is milder but is still very significant for CREASY with lower α values: 21%, 36%, and 44% for αs of 1500, 3000, and 4500, respectively. A similar improvement is seen in the measured system utilization, which increased from 57% to 85% for the largest α value. This happens since the two metrics are strongly correlated: as long as the system is not saturated, increasing throughput directly leads to an increase in utilization.

To understand this exceptional improvements in throughput we need to examine the behavior of the users under the five schedulers. We chose to focus on the users sessions and in particular on the length of their sessions with the system.



(a) Average job throughput



(b) Average session length

Fig. 10. Average job throughput and session length: the higher the value of α, the higher the throughput (a), and the milder the drop in session length (b).

Session length is defined as the number of jobs the users submit during their sessions of activity with the system, and hence it serves as a good indicator for user satisfaction.

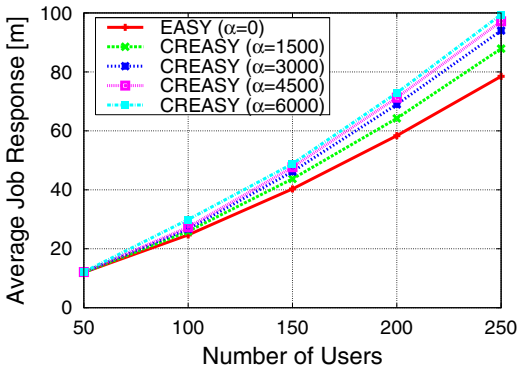
Figure 10(b) shows the average session length under the five schedulers, as a function of the size of the site. Under the EASY scheduler, session length drops significantly from 2.69 jobs/session on average for the small 50 users site, to 1.73 jobs/session for the 250 users site — a 36% drop. The drop becomes milder with higher α values, and it is hardly noticeable when $\alpha = 6000$.

The reason session length drops is rooted in the core design of the schedulers. The original EASY scheduler does *not* consider the critically of the jobs. As a consequence, when the load begins to increase more and more users under EASY abort their sessions as a result of their jobs being delayed by the scheduler. This causes average session length to decrease, and explains the poor throughput of the scheduler in Figure 10(a).

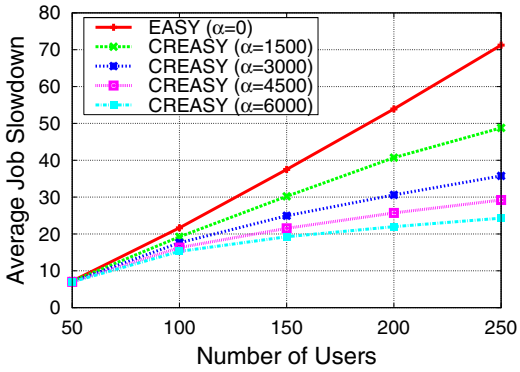
As we increase the value of α, the chances for critical jobs to execute before other jobs also increase: the higher the value of α, the higher the priority of critical jobs, and the more critical jobs that respond in time, which causes more users to continue their sessions with the system, and the overall job throughput to improve.

Finally, when $\alpha = 6000$, the drop in session length is hardly noticeable even under the highest loads. This means that CREASY was capable of virtually isolating the interactive users from the load conditions that exist in the system, suc-

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication.

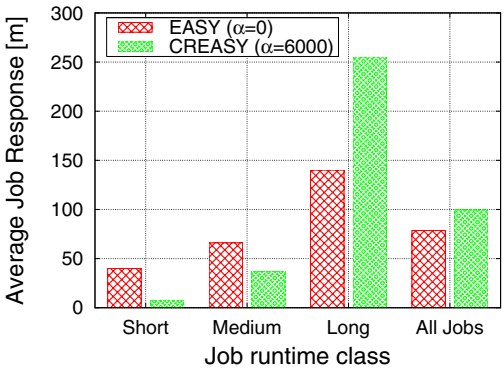


(a) Average response time

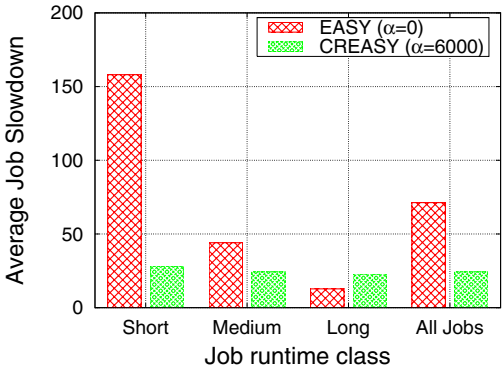


(b) Average slowdown

Fig. 11. Inconsistency according to the conventional metrics: the schedulers with the *lower* α values outperform the ones with the higher values according to the response-time metric (a), but according to the slowdown, the ones with the *higher* values have significantly better performance (b).



(a) Per-class average job response time



(b) Per-class average job slowdown

Fig. 12. Per-class performance comparison of the two schedulers: (a) the 27% increase in the average response is the outcome CREASY's tendency to prioritize short jobs at the expense of longer ones that dominate the average, and (b) the 66% decrease in the average slowdown is the result of the exact same trade-off, and the fact that the metric is affected mostly by the shorter jobs.

cessfully providing them with a highly responsive environment even under the most extreme load conditions.

2) Conventional Performance Metrics: While throughput remains the best indicator for user productivity, there are other, more conventional metrics that can be measured in site-level simulations as well. Figure 11 shows the performance of our five schedulers according to two of the most commonly used metrics: the average job response time and the slowdown.

Similar to Figure 10, we see that the differences in performance between the schedulers become significant only when the load begins to increase. In contrast from Figure 10 though, it is the schedulers with the *lower* α values that outperform the ones with the higher values, but only according to the response-time metric of Figure 11(a). The ones with the *higher* values still have significantly better performance according to the slowdown.

Under the highest 250 users load for example, the average job response time under the EASY scheduler is 78 minutes, while under CREASY with $\alpha = 6000$ it is 99 minutes, which is a 27% degradation in performance for CREASY. On the other hand, the average job slowdown under EASY is 71, while under CREASY it is *only* 24, which is a 66% improvement.

The above results are surprising. We would expect performance to improve with higher α values according to both metrics, since the metrics are conjectured to be correlated with user satisfaction, and thus should improve along with the throughput metric of Figure 10(a). Obviously, this is not the case, and the response-time metric is in fact inversely correlated with productivity.

To understand the reason for this inconsistency, we divided the jobs into classes according to their runtimes, and examined the average response-time and slowdown in each class. We chose to use three classes: one for short jobs of up to 1 minute of runtime, the second for medium jobs whose runtime is between 1 and 10 minutes, and the third for longer jobs that execute for more than 10 minutes. We chose these boundaries based on the distribution of runtimes from Figure 5(b), in order to create classes of approximately the same size.

Figure 12 compares the performance of the EASY scheduler with CREASY using $\alpha = 6000$, under the highest, 250-user load, on a per-class basis. For the response time metric in Figure 12(a), we see that under both schedulers the response times of the jobs is correlated with their runtimes: the higher the runtimes, the higher the average response time in the class.

The difference, though, is that under EASY the increase in the average response time is rather moderate, while under CREASY it is more extreme. Furthermore, in the class with the short jobs the average under CREASY is 83% *lower* than the average under EASY, in the class with the medium runtimes it is only 44% lower, while in the third class, the average response under CREASY is 81% *higher* compared to EASY.

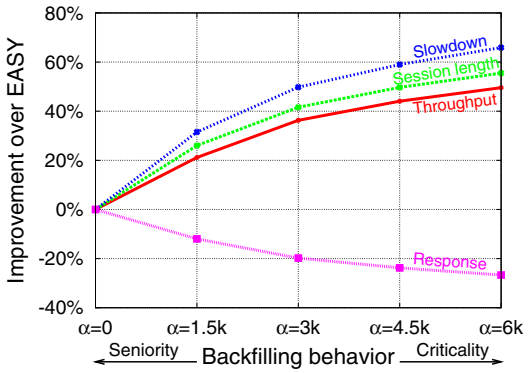


Fig. 13. Improvements in performance for CREASY relative to the original EASY scheduler: performance improves with higher α values according to the throughput, session-length, and the slowdown metrics, but degrades according the response-time metric.

These differences are, once again, rooted in the core design of the schedulers. Short jobs have naturally more backfilling opportunities than jobs with longer runtimes. While this is true under both schedulers, the effect is intensified under CREASY as it further prioritizes the short jobs which are also much more critical to the users. The outcome is a large reduction in the response of the short jobs, at the expense of an increase in response for the longer jobs — a trade-off resulting in a 27% higher average response-time for CREASY, because the long jobs dominate the average.

The slowdown metric in Figure 12(b) behaves exactly the opposite: the average slowdown is inversely correlated with the runtimes, decreasing under both schedulers as the runtimes increase. But this time, the decrease is steep under EASY, and very small under CREASY.

Slowdown is the response time normalized by the actual runtime, which causes the metric to be affected mostly by the shorter jobs. This means that although the relative differences in the average slowdown between EASY and CREASY in each class are similar to the differences in the average response time, the absolute values of the metric are intensified in the class of the short jobs, and lessened in the class of the longer jobs. This changes the relative contribution of each class of jobs to the overall average, and results in a 66% lower absolute average slowdown for CREASY.

Figure 13 summarizes the improvements in performance under CREASY for all four metrics: the average job throughput, session length, the average job slowdown, and the average response times. The results were measured under the highest 250-users load, and are all relative to the performance of the original EASY scheduler.

When $\alpha = 0$, there are no gains or losses in performance under CREASY since its behavior is identical to the behavior of the EASY scheduler. When the value of α increases, performance improves under CREASY but only for the first three metrics; for the response time metric performance degrades with higher α values. In either case, both improvements or degradations are not linear, and the curves begin to level-off at the right side of the scale.

V. Related work

Traditionally, parallel-systems schedulers are evaluated using trace-driven simulations [5], [4], [13], [14], [3], [2]. The alternative but less common approach is to use models to generate the workload for the simulations, but the early models were too simplistic and failed to capture important proprieties of the workload such as self-similarity, locality, and cycles of activity [9], [15], [16], [17], [18].

Zilber et al. were the first to present a comprehensive study of the traces based on users and sessions [7]. Similarly but in a different context, Arlitt presented a session-based analysis of web server logs [19]. Both studies can be used to develop more realistic models of the workload based on users and sessions, but they lack a description of how the users react to the performance observed from the system.

While the mechanics of using traces during simulation are straightforward, there is more than one way to model the users that generate the workload. Haugerud and Straumsnes used user models that have different characteristics and whose behavior is affected solely by the time of the day, to simulate the workload of an interactive computer system [20]. Hlavacs et al. suggested a layered user model made of sessions, application, and commands, and demonstrated its use in driving network simulations [21]. Shmueli and Feitelson used two interdependent models: a job-submission model and a job characteristics model, that together generate the workload for job-scheduling simulations [22]. In these cases also, user behavior was independent of the performance of the system.

Until recently, it was assumed that studying user reaction to performance requires live experiments with real users. Bouch et al. for example, used live experiments to investigate the tolerance of users to web server delays [23]. Similarly, Lee and Snavely examined user satisfaction live, at the San Diego Supercomputer Center [8]. Recent studies on the other hand have shown that it is possible to uncover user behavior directly from traces of the system. Tran et al. developed a model of web surfers reaction to network congestion just by analyzing HTTP packet-traces [24]. Similarly, Chen et al. developed a model of Skype's users satisfaction purely from their VoIP traces [25].

In the context of parallel job scheduling, we were the first to characterize user behavior. Using a novel trace analysis methodology that we developed we have found that user behavior is correlated with the response times of their jobs [6]. This enabled us to develop the session-dynamics model in Section III-B.1 that depicts the users reaction to their jobs.

The SJF scheduling algorithm is well-known to produce optimal average response times, and variants have been incorporated in several parallel schedulers [26], [27]. However, our prioritization differs from SJF in that we consider the response time rather than just the job's execution time. Thus our scheduler is not equivalent, or even similar, to those other parallel schedulers.

VI. Conclusions

For more than two decades parallel-systems schedulers are being evaluated using simulations that suffer from severe limitations. In particular, the inability of these simulations

to reproduce the fine-grained interaction that naturally exists between the users and the system, led to the design of schedulers that focus solely on the packing of jobs as a mean to improve a number of performance metrics that are only conjectured to be correlated with user satisfaction.

In this paper we have shown that the conventional, packing-based approach to scheduling is far from optimal, and demonstrated the potential in user-aware designs. We have also shown that the conventional metrics do not necessarily correlate with user productivity, which means that it is even difficult to identify good designs under these simulations.

Site-level simulations allow user-aware designs to be explored and their performance to be reliably evaluated, but they rely on user-models to generate the workload, which, as opposed to traces, will always be open for interpretation. Though we feel we have clearly demonstrated their advantage, we are also aware of the fact that we relied on major simplifications, and that user behavior in reality is far more involved than what our current model depicts.

It is known for example that real users are influenced by contextual factors such as the type of task they perform, their experience, and the cumulative time they interact with the system. It is also known that users are sensitive to fairness in the system, and might consider fairness to be more important than productivity. It is important to understand these factors and the way they affect the users, in order to enhance the models and improve the accuracy of the evaluation.

Furthermore, the jobs the users submit during the day are known to be different from those submitted during the night: interactive jobs usually require much less resources and are much more critical to the users than the batch jobs that execute over nights and weekends. Users also use scripts to automate the submission of their jobs, which may result in tens and hundreds of jobs being submitted simultaneously. Such factors should also be taken into consideration.

Finally, CREASY will need to be revised to consider the aggregate effect of all these factors on the users, and its performance will need to be evaluated again to demonstrate that it can still significantly improve user productivity. lThis task alone is extremely challenging, but it is a necessary step toward the actual deployment of the first truly user-aware parallel-system scheduler.

Acknowledgments

This research was supported in part by the Israel Science Foundation (grant no. 167/03). Many thanks are due to the people and organizations who deposited their workload logs in the Parallel Workloads Archive, and made this research possible.

REFERENCES

[1] D. Lifka, "The ANL/IBM SP scheduling system," in *Job Scheduling Strategies for Parallel Processing*, D. G. Feitelson and L. Rudolph, Eds. Springer-Verlag, 1995, pp. 295–303, lect. Notes Comput. Sci. vol. 949.

[2] E. Shmueli and D. G. Feitelson, "Backfilling with lookahead to optimize the packing of parallel jobs," *J. Parallel Distrib. Comput.*, vol. 65, no. 9, pp. 1090–1107, 2005.

[3] B. G. Lawson and E. Smirni, "Multiple-queue backfilling scheduling with priorities and reservations for parallel systems," *SIGMETRICS Perform. Eval. Rev.*, vol. 29, no. 4, pp. 40–47, 2002.

[4] 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 Distrib. Syst.*, vol. 12, no. 6, pp. 529–543, 2001.

[5] D. Talby and D. G. Feitelson, "Supporting priorities and improving utilization of the IBM SP scheduler using slack-based backfilling," in *IPPS '99/SPDP '99: Proceedings of the 13th International Symposium on Parallel Processing and the 10th Symposium on Parallel and Distributed Processing*. Washington, DC, USA: IEEE Computer Society, 1999, p. 513.

[6] E. Shmueli and D. G. Feitelson, "Uncovering the effect of system performance on user behavior from traces of parallel systems," in *15th Modeling, Anal. & Simulation of Comput. & Telecomm. Syst. (MASCOTS)*, Oct 2007, pp. 274–280.

[7] 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*, Jun 2005, pp. 377–386.

[8] C. B. Lee and A. Snavely, "On the user-scheduler dialogue: Studies of user-provided runtime estimates and utility functions," *Int. J. High Perform. Comput. Appl.*, vol. 20, no. 4, pp. 495–506, 2006.

[9] D. G. Feitelson, "Packing schemes for gang scheduling," in *Job Scheduling Strategies for Parallel Processing*, D. G. Feitelson and L. Rudolph, Eds. Springer-Verlag, 1996, pp. 89–110, lect. Notes Comput. Sci. vol. 1162.

[10] D. G. Feitelson, "Locality of sampling and diversity in parallel system workloads," in *21st Intl. Conf. Supercomputing (ICS)*, Jun 2007, pp. 53–63.

[11] V. Lo and J. Mache, "Job scheduling for prime time vs. non-prime time," in *Intl. Conf. Cluster Comput.*, no. 4, Sep 2002, pp. 488–493.

[12] G. Sabin and P. Sadayappan, "Unfairness metrics for space-sharing parallel job schedulers," in *Job Scheduling Strategies for Parallel Processing*, D. G. Feitelson, E. Frachtenberg, L. Rudolph, and U. Schwiegelshohn, Eds. Springer Verlag, 2005, pp. 238–256, lect. Notes Comput. Sci. vol. 3834.

[13] S. Srinivasan, R. Kettimuthu, V. Subramani, and P. Sadayappan, "Selective reservation strategies for backfill job scheduling," in *JSSPP '02: Revised Papers from the 8th International Workshop on Job Scheduling Strategies for Parallel Processing*. London, UK: Springer-Verlag, 2002, pp. 55–71.

[14] J. William A. Ward, C. L. Mahood, and J. E. West, "Scheduling jobs on parallel systems using a relaxed backfill strategy," in *JSSPP '02: Revised Papers from the 8th International Workshop on Job Scheduling Strategies for Parallel Processing*. London, UK: Springer-Verlag, 2002, pp. 88–102.

[15] A. B. Downey, "A parallel workload model and its implications for processor allocation," *Cluster Computing*, vol. 1, no. 1, pp. 133–145, 1998.

[16] 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*, D. G. Feitelson and L. Rudolph, Eds. Springer Verlag, 1997, pp. 95–116, lect. Notes Comput. Sci. vol. 1291.

[17] U. Lublin and D. G. Feitelson, "The workload on parallel supercomputers: Modeling the characteristics of rigid jobs," *J. Parallel & Distrib. Comput.*, vol. 63, no. 11, pp. 1105–1122, Nov 2003.

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

[19] M. F. Arlitt, "Characterizing web user sessions." *SIGMETRICS Perform. Eval. Rev.*, vol. 28, no. 2, pp. 50–63, 2000.

[20] H. Haugerud and S. Straumsnes, "Simulation of user-driven computer behaviour," in *LISA '01: Proceedings of the 15th USENIX conference on System administration*. Berkeley, CA, USA: USENIX Association, 2001, pp. 101–108.

[21] H. Hlavacs and G. Kotsis, "Modeling user behavior: A layered approach," in *MASCOTS '99: Proceedings of the 7th International Symposium on Modeling, Analysis and Simulation of Computer and Telecommunication Systems*. Washington, DC, USA: IEEE Computer Society, 1999, p. 218.

[22] E. Shmueli and D. G. Feitelson, "Using site-level modeling to evaluate the performance of parallel system schedulers," in *MASCOTS '06: Proceedings of the 14th IEEE International Symposium on Modeling, Analysis, and Simulation*. Washington, DC, USA: IEEE Computer Society, 2006, pp. 167–178.

[23] A. Bouch, A. Kuchinsky, and N. Bhatti, "Quality is in the eye of the beholder: meeting users' requirements for internet quality of service,"

in *CHI '00: Proceedings of the SIGCHI conference on Human factors in computing systems*. New York, NY, USA: ACM Press, 2000, pp. 297–304.

[24] Y. T. Dinh Nguyen Tran, Wei Tsang Ooi, "Sax: A tool for studying congestion-induced surfer behavior," in *In Proceedings of Passive and Active Measurement Conference, Adelaide, Australia*, March 30-31 2006.

[25] K.-T. Chen, C.-Y. Huang, P. Huang, and C.-L. Lei, "Quantifying skype user satisfaction," in *SIGCOMM '06: Proceedings of the 2006 conference on Applications, technologies, architectures, and protocols for computer communications*. New York, NY, USA: ACM Press, 2006, pp. 399–410.

[26] D. Tsafrir, Y. Etsion, and D. G. Feitelson, "Backfilling using system-generated predictions rather than user runtime estimates," *IEEE Trans. Parallel Distrib. Syst.*, vol. 18, no. 6, pp. 789–803, June 2007.

[27] S.-H. Chiang, A. Arpaci-Dusseau, and M. K. Vernon, "The impact of more accurate requested runtimes on production job scheduling performance," in *Job Scheduling Strategies for Parallel Processing*. Springer-Verlag, Jul 2002, no. 8, pp. 103–127, lect. Notes Comput. Sci. vol. 2537.

Edi Shmueli is a PhD student at the Hebrew University in Jerusalem, and a staff member at the IBM research lab in Haifa, Israel. Since 2002 Edi has been working on the BlueGene/L project, designing and implementing the supercomputer's job management system. From 2006 to 2007 he joined the BlueGene team at the IBM T. J. Watson research center where he investigated the supercomputer operating system. His research interests include computer architectures, operating systems, supercomputing and parallel programming models, and systems performance evaluation.

Dror G. Feitleson has been on the faculty of the School of Engineering and Computer Science at the Hebrew University of Jerusalem since 1995. He is the founding co-organizer of JSSPP, the annual series of workshops on Job Scheduling Strategies for Parallel Processing. More recently, he organized the ACM Workshop on Experimental Computer Science, as part of an effort to promote experimental procedures and base systems research on solid data rather than assumptions. As part of this effort he also maintains the Parallel Workloads Archive.