The Importance of Feedback in Evaluating and Designing Parallel Systems Schedulers

Thesis submitted for the degree of
"Doctor of Philosophy"

by

Edi Shmueli

Submitted to the Senate of the Hebrew University
September, 2008

This work was carried out under the supervision of

Prof. Dror G. Feitelson

Acknowledgments

Six years ago, when I began working on the Blue Gene/L project in IBM Haifa, Jose Moreira of the IBM T.J. Watson Research Center who was a manager in the project at that time suggested that we contact Dr. Dror G. Feitelson, a researcher from the Hebrew University in Jerusalem, who investigates job scheduling for parallel-systems. A couple of weeks later, I found myself waking up as early as 5:30am in the morning and driving to Jerusalem to meet Dr. Feitelson for the first time. Who could have guessed that this one-time trip will turn into a routine, once a week, every week, for the next six years.

Dror, you first accepted me as your M.A. student, though officially I belonged to the Haifa University. In fact, my masters thesis, our thesis, was the first to be submitted in the newly formed computer science department in Haifa; thank you. I never thought to pursue a third degree; never cared much for academic research really, but “never” as I personally discovered, can be open for interpretation. Our fruitful discussions, your vast comprehensive knowledge and bright academic way of thinking captivated me. You accepted me, supported and guided me throughout the entire course of the PhD, even when I was half a globe away. You thought me uncompromising excellence, and got me to the peak of my academic performance. Thank you again, deeply.

Dr. Michael Rodeh. Ten years ago I entered your office in our old IBM building in MATAM, and asked to move to a research position in HRL. You said I should first have an academic degree, and began to type something in your computer. The next day I got a call from the Haifa University, asking me to come over and bring my grade chart. Two months later, I found myself sitting in class taking my first B.Sc course; the rest is history. It is amazing how this one short meeting totally changed my life. Ten years of who knows how many sleepless nights, depicted in this dissertation are now coming to an end, and it starts to feel worthwhile. Thank you too.

Finally, to the team of the parallel-systems lab at HUJI, Dan Tsafrir, Yoav Etsion, Tal Ben-Nun, Ahuva Mualem, Eitan Frachtenberg, Keren Quaknine, and all other past and present students for your friendship and the enjoyable times we spent together.

Thank you all.

Abstract

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 jobs get submitted during the simulation solely according to the timestamps from the trace, and there is *no feedback* in the workload between the arrival of new jobs, the load in the system, and the ability of the simulated scheduler to handle the load. The importance of this feedback is the subject of this research.

We argue that the lack of feedback in the workload affects not only the *evaluation* of the schedulers but also their *design*. It affects the evaluation since the generated workloads no longer reliably represent real workloads, which causes the performance predicted by the simulation to be inaccurate. It affects the design because the throughput metric which is the best indicator for user productivity cannot be used in open-system evaluation. This forces the schedulers to focus on the packing of jobs instead of on the users of the system directly, to try and optimize an alternative set of metrics that are only conjectured to correlate with user satisfaction.

As an alternative, we propose a novel simulation methodology named *site-level simulation* that uses user-models instead of traces to *dynamically* generate the workload for the evaluation. These models, whose behavior in simulation is similar to the behavior of real users, interact with the system and introduce feedback that improves the representativeness of the workload, and allows user-aware schedulers to be reliably evaluated and hence effectively designed.

The most important elements in a site-level simulation are the *user-models*. We present a novel analysis methodology through which we demonstrate that it is possible to uncover the users' behavior patterns directly from traces of systems, without conducting live experiments with real users. We show that users of parallel-systems are affected by the *response times* of their jobs and not the slowdown as was previously assumed, and that the longer the response the higher the probability for the users to abort their interactive sessions with the system. These findings form the basis for the user models we use in our simulations.

To experiment with site-level simulations we present *Site-Sim* — a site-level simulator that integrates users and schedulers under a single simulation framework, to reproduce the feedback effects found in real sites workloads. We then carry a series of carefully designed experiments to demonstrate the importance this feedback for the evaluation and the design of the schedulers.

We show that the conventional simulations tend to under or overestimate the performance of the schedulers, and that the prediction errors may reach hundreds of percents. We demonstrate how conventional load scaling further ruins the representativeness the workload by violating the precedence relations that naturally exist between jobs in reality.

We present *CREASY* — a novel user-aware scheduler that exploits knowledge on user behavior to try and improve user satisfaction, and compare its performance to the well-known EASY scheduler that focuses solely on the packing of jobs. We show that user productivity improves by tens of percents under the user-aware design, and that this stems from our scheduler's ability to maintain long user sessions under high loads. We also show that conventional performance metrics such as the mean job response time do not necessarily correlate with productivity, which means that it is even possible to dismiss potentially good design alternatives under the conventional simulations.

This work was conducted solely in the context of parallel-systems scheduling. However, the concept of incorporating feedback in the workload is applicable to many other types of systems such as I/O subsystems, memory hierarchies, and communication networks, that are still evaluated using trace-driven, open-system simulations. We believe the ideas we present in this work are applicable to these types of systems as well, which can similarly benefit from a highly representative workload and a reliable evaluation.

Contents

List of Figures

Chapter 1

Introduction

PARALLEL-SYSTEMS SCHEDULERS are usually evaluated in simulation before they become candidates for an actual deployment. High setup costs involved in such deployments and the negative impact an inefficient scheduler may have on the productivity of its users make the role of simulations critical, and emphasize the need for a reliable simulation methodology. This research exposes the shortcomings of the conventional, trace-driven, open-system simulations presently used to evaluate the schedulers, that stem from a workload generation process that lacks any feedback. It proposes and promotes a novel simulation methodology that uses user-models instead of traces to dynamically generate the workload for the evaluation, and demonstrates through a series of carefully designed experiments that it is more reliable. The first three sections in this introduction chapter briefly describe the conventional simulations, the problems that arise from not having feedback in the workload, and our proposed simulation methodology, respectively. Section 1.4 describes the overall structure of the dissertation.

1.1 The Conventional Simulations

In its simplest form a parallel-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.

Furthermore, there is no time-sharing nor preemption support in the 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 system may need to serve tens or even hundreds of users simultaneously, so naturally users are not allowed to access the system directly. Instead, they submit their jobs to the system scheduler which is the central software component that manages the system's resources, and rely on the scheduler to execute the jobs on their behalf.

Submitting a job implies providing to the scheduler a description of the jobs' resource requirements. 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.

The scheduler in turn accepts the jobs from the users and places them in a queue where they wait for processors to become available. Whenever the state of the system changes, either due to an arrival of a new job or a termination of a running job, it scans the queue and selects one or more jobs for execution based on the current and optionally the projected processor usage. Different schedulers use different heuristics to select the jobs, but all schedulers must prevent the starvation of jobs.

An important goal of the scheduler is to promote the productivity of its users, and this requires the scheduler to keep its users satisfied and motivate them to submit more jobs. When a new scheduler is proposed, the only way to accurately measure its performance is to deploy it in a real environment with live users, but this is often impractical due to high setup costs and the idle time spent on training the users.

Theoretical analysis e.g., queuing theory, can be very useful in order to get a rough indication on performance, but the need to use abstraction to enable mathematical tractability limits the detailed description of the scheduler. Since such details are often important, e.g., when comparing two closely related scheduling algorithms, evaluating the scheduler's performance using simulation becomes the only practical alternative.

The conventional simulations presently used to evaluate the schedulers are *trace-driven* which means that they use traces as the source for the workload. The traces come from real production-use parallel-systems and contain records of the jobs that were submitted to the system by its users over a period of time. Each record in the trace has several data fields that describe a job, and which are later treated as requirements for resources by jobs, in the course of simulation.

The records also include a *timestamp* that indicates when the job was originally submitted. To actually generate the workload, the conventional simulations use an *open-system* model that simply plays-back the trace according to the timestamps and irrespectively of the system's state. Sometimes the timestamps are scaled by a certain factor before the simulation begins in order to expand or reduce the inter-arrival times and change the load conditions for the simulation, but in either case there is *no feedback* in the workload between the arrival of new jobs, the load in the system, and the ability of the simulated scheduler to handle the load.

1.2 Importance of Feedback in the Workload

This lack of feedback in the workload manifests itself in several ways. We decided to structure this dissertation around what we believe are the two most prominent domains affected by the feedback: *scheduler evaluations* and *scheduler designs*. The next two sections briefly describe the problems that arise from not having feedback in each of the domains.

1.2.1 Scheduler Evaluations

The premise underlying the conventional simulations is that the generated workloads, and regardless of whether the load was modified or not, are indeed reliable representatives of real workloads, otherwise the performance predicted by the simulation would be inaccurate.

We argue that this is not the case since in reality there is a *continuous feedback* between the performance of the system and the behavior of its users: users submit fewer jobs if the system is already loaded and response times are long, but on the other hand would exploit periods of low load and short queuing times to submit as many jobs as possible.

Figure 1.1 illustrates these feedback effects in the workload. Each sub-figure represents a trace of a real parallel-system. We partitioned the traces into weekly slices, and counted the number of jobs submitted, and the average job node-seconds in every slice. When plotting one against the other we see that when there are many jobs they tend to be smaller but when the jobs are heavy, there are fewer of them.

Such feedback effects leave their signature in the traces in the form of *timestamps*. When playing back the trace according to these timestamps, the generated workload then matches the scheduling policy that was in effect when the trace was recorded instead of adapting itself to the scheduler being evaluated, which causes the performance predicted by the simulation to be highly inaccurate.

We also argue that load scaling as currently performed further ruins the representativeness of the workload by generating conditions that cannot exist in reality. Since users often wait for their jobs to complete before submitting additional jobs, some jobs simply cannot reside together in the scheduler's queue. However, when modifying the timestamps in the trace to simulate higher loads, not only do such jobs get to coexist together, some jobs may even be picked for execution while their predecessors are still queued.

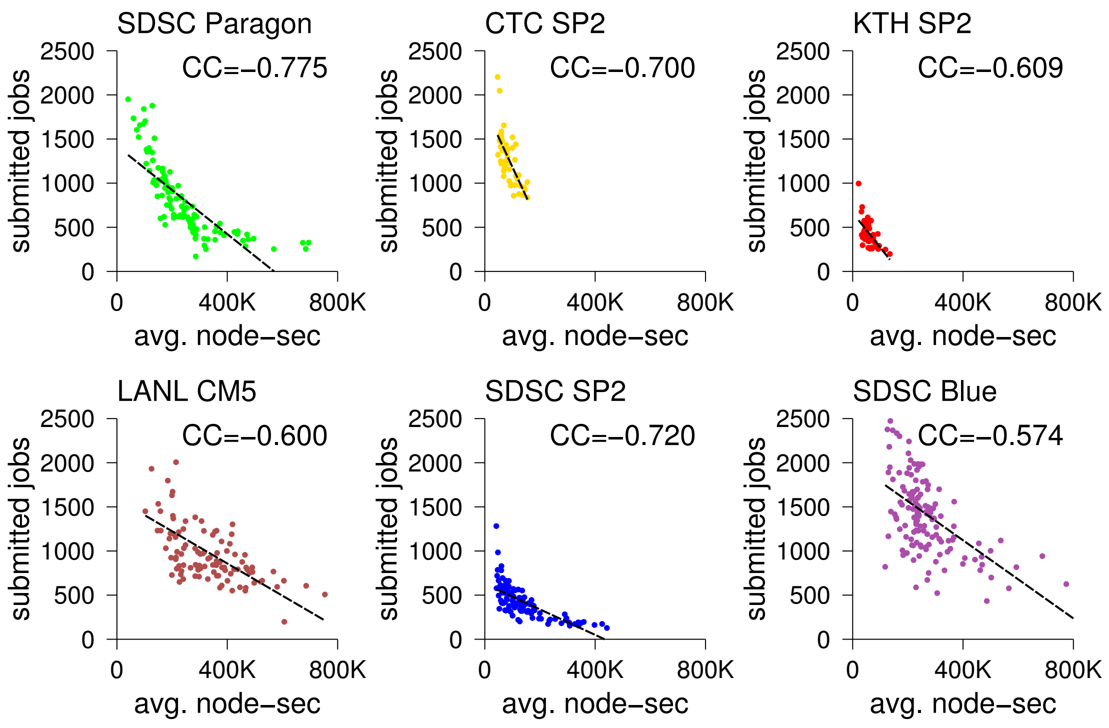


Figure 1.1: *Feedback in real workloads: in weeks where many jobs are submitted they tend to be smaller, and vice versa.*

1.2.2 Scheduler Designs

Another problem which is attributed to the open nature of the simulations is that the *throughput* of the scheduler being evaluated gets dictated solely by the timestamps from the trace, and it is not affected by the 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 vice-versa. This means that the throughput metric — which is the best indicator for user productivity, 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's actions, and on the other be also conjectured to correlate with user satisfaction. More specifically, the jobs' average *response-time* and *slowdown* 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 users of the system only implicitly through these metrics. They often try to optimize the packing of the jobs in the schedule, since tighter packing usually leads to lower average simulated values.

We argue however, that the only way to truly maximize productivity is to consider the users of the system directly — to strive to keep them satisfied and motivate them to submit more jobs and that the conventional, packing-based approach to scheduling leads to sub-optimal designs. We also argue the that these seemingly "user-friendly" metrics do not necessarily correlate with productivity, which means that it is even possible to dismiss potentially good design alternatives as poor under the conventional simulations.

1.3 Site-Level Simulations

As an alternative to these simulations we propose a novel simulation methodology named *site-level simulation*, that uses *user-models* instead of traces to *dynamically* generate the workload for the evaluation. These models, whose behavior in simulation is similar to the behavior of real users, interact with the system and introduce feedback that not only improves the representativeness of the workload, but also allows user-aware schedulers to be effectively designed. The later stems from the fact that the throughput metric in our simulations *is in fact* affected by the actions of the scheduler, which means that schedulers can be designed to improve user satisfaction directly, and their effect on productivity will be reliably evaluated.

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 the simulation. One of the major contributions of this work is the analysis methodology we developed, through which we demonstrate that it is possible to uncover the users' behavior patterns directly from traces of systems, without conducting live experiments with real users.

More specifically, we have found that the behavior of users of parallel-systems is affected by the *response times* of their jobs, not the slowdown as was previously assumed. We also found that response times affect the users' decision to continue or abort their interactive sessions with the system, and that this may relate to expectations that the users develop. These findings form the basis for the user models we use in our simulations.

To experiment with site-level simulations we present *Site-Sim* — a site-level simulator that integrates users and schedulers under a single simulation framework, to reproduce the feedback effects found in real sites workloads. We then carry a series of carefully designed experiments to demonstrate the importance of the feedback in the workload for both the evaluation and the design of the schedulers.

Our first set of experiments focuses on the representativeness of the workload. In a nutshell, we run site-level simulations of low-end and high-end schedulers, and use the traces produced by Site-Sim as input for conventional simulations. We show that the performance predicted by these simulations tends to be under or overestimated, and that the prediction errors may reach hundreds of percents. For example, the mean job response time of the FCFS scheduler is more than 24 hours when simulated using a trace of the EASY scheduler, but it is only $3\frac{1}{3}$ hours when evaluated in a site-level simulation having feedback in the workload.

We also use the traces to demonstrate how conventional load scaling further ruins the representativeness of the workload. We record in the traces information on precedence relations between the jobs, and modify the timestamps in the trace to simulate higher loads. We show that the percentage of jobs that get submitted while their predecessors are still active in the system increases rapidly with the load and furthermore, that the scheduler even picks jobs for execution that depend on the completion of other jobs that are still queued, which totally violates the original orderings of the jobs.

For the second set of experiments we present *CREASY* — a novel user-aware scheduler that inherits its backfilling algorithm from the original, packing-based EASY scheduler, but uses a novel prioritization scheme that exploits knowledge on user behavior to prioritize jobs that are critical to the users and improve user satisfaction. We compare the performance of our scheduler to that of EASY's and show that user productivity improves by more than 50% under the user-aware design, and that this stems from CREASY's ability to maintain long user sessions under high loads.

We also compare the two schedulers according to the conventional performance metrics and show that the average job response time under CREASY is 27% higher compared to EASY, while the average slowdown is 66% lower. This inconsistency, which is the outcome CREASY's tendency to prioritize short jobs that mostly affect the slowdown metric at the expense of longer ones that affect the response, makes it extremely difficult to identify the best scheduler for the system, and may even lead to choosing EASY over CREASY as the preferred design alternative.

1.4 Dissertation Structure

This dissertation evolves in three consecutive chapters, 3, 4 and 5, that together provide a complete and deep understanding of the importance of the feedback in the workload. Each chapter is self-contained and focuses on a subset of the problem. Chapter 3 focuses on the importance of the feedback for the evaluation of the schedulers and Chapter 5 concentrates on its importance for their design. Chapter 4 does not discuss the problems directly but provides important insights as to how parallel systems affect the behavior of their users — insights that form the basis for the user-models presented in Chapter 5.

All chapters begin with a Background section that introduces the problem, our approach to solution, and briefly describes the results. All chapters end with a Summary section that summarizes the chapter and motivates the next. Before that section we always include a Related Work section that surveys publications in related contexts.

Table 1.2 on page 8 describes the overall structure of the dissertation. The table has three main entries that correspond to chapters 3, 4 and 5, respectively. For consistency, we divided each of these entries into five sub-entries that describe the *problem* each chapter tackles, our proposed *solution*, what we believe are the main *contributions* of the chapter, the *results*, and the *next step* which motivates the following chapter. Chapters 2 and 6 of the dissertation are intentionally left out of the table; Chapter 2 discusses the methodological principles upon which this work is based, and Chapter 6 concludes the entire research and suggests future research directions.

The following table lists our refereed publications and the chapters to which they pertain. Extended versions of these publications also appeared as technical reports.

Ref.	**Year**	**Venue**	**Chap.**	**Comments**
[55]	2006	MASCOTS	3	Nominated for Best Paper Award (1 of 3)
[56]	2007	MASCOTS	4	
[57]	2008	TPDS	5	1'st reviewer: *"This work has the potential to make a substantial change to way future job scheduling research and development is performed."* 2'nd reviewer: *"I consider this paper one of great impact in the field...I expect this paper to be widely cited in the community."*

Table 1.1: *Our refereed publications and the chapters to which they pertain.*

Chapter 3: Feedback and Scheduler Evaluations	
The problem	Workloads in conventional simulations are not reliable representatives of real workload, leading to performance prediction errors and unsafe load scaling.
Our solution	Use site-level simulations in which the workload is generated dynamically to evaluate the performance of the schedulers.
Contribution	First attempt to understand user behavior from traces, identification of user sessions and batches, development of user models and their integration in Site-Sim.
Results	Conventional simulations suffer from prediction errors of hundreds of percents, load scaling generates conditions that cannot exist in reality.
Next step	Develop more realistic models of the users, so the simulation will not be limited to a static number of active sessions. Calls for a deeper understanding of user behavior.
Chapter 4: Understanding User Behavior	
The problem	Understanding user behavior typically requires research in psychology and the conduction of live experiments.
Our solution	A novel analysis methodology to uncover the effect of the system on its users directly from traces of systems.
Contribution	Three questions are answered: *which* performance metric is most important to the users, *how* does it affect their behavior, and *why* this happens.
Results	User behavior is correlated with the *response times* of their jobs, not the slowdown. Response times affect the decision of users to *continue* or *abort* their sessions with the system, and this may relate to *expectations* the users develop.
Next step	Use the above findings to develop more realistic models of the users, and a scheduler that exploits this to improve productivity.
Chapter 5: Feedback and Scheduler Designs	
The problem	Conventional simulations lead to sub-optimal scheduler designs, and may dismiss potentially good design alternatives as poor.
Our solution	Site-level simulations with comprehensive user models allow the reliable evaluation and hence the design of schedulers that focus on the users directly.
Contribution	Development of CREASY — the first truly user-aware scheduler that exploits knowledge on user behavior to improve user satisfaction.
Results	Productivity under CREASY improves by 50% compared to existing designs, while according to the conventional metrics its performance actually degrades.
Next step	Further investigate user behavior, enhance the models and revise CREASY.

Table 1.2: *Dissertation structure: this dissertation evolves in three consecutive chapters that together provide a complete and deep understanding of the importance of the feedback. Each chapter is self-contained and focuses on a subset of the problem.*

Chapter 2

Methodology

WE ARGUE that the conventional simulations lead to inaccurate performance predictions and sub-optimal scheduler designs, but we lack a live environment with a real parallel system and users to prove it. This imposed a great challenge that led us to seek alternative ways to demonstrate our point in a convincing manner. We chose to concentrate on simulations and developed *Site-Sim* — a site-level simulator that integrates users and schedulers under a single simulation framework, to simulate the workloads that would have been observed by the scheduler had we had a live environment. The most important elements in Site-Sim are the user models that generate the workload. An even greater challenge was to understand the behavior of the users and to capture this behavior in a model that can be used in simulation. Though intuitively it seems that this requires research in psychology, we found that it is possible to uncover the users' behavior patterns directly from parallel-systems *traces*, without conducting live experiments with real users. Using one simulation methodology to demonstrate the shortcomings of another, and relying on traces as the primary source of data are therefore the two key principles upon which this work is based. In the following two sections we briefly describe Site-Sim and the traces. Due to its importance, we defer the discussion on user behavior to Chapter 4.

2.1 Site-Sim: Our Site-Level Simulator

Site-Sim is a framework written in C++ that we developed specifically for running site-level simulations. We used Site-Sim extensively in all our experiments, both in Chapter 3 to demonstrate the importance of the feedback for the evaluation, and in Chapter 5 to explore design alternatives as we developed CREASY, our user-aware scheduler.

Site-Sim defines two types of entities, *users* and *schedulers*. The users generate the workload for the simulation by submitting jobs to the scheduler, and the scheduler in turn schedules the jobs and notifies the users when they complete. This interaction between the users and the scheduler continues throughout the entire course of the simulation, resulting in a workload that is generated dynamically with respect to the temporal load conditions that exist in the system.

We refer to Site-Sim as a "framework" since it does not explicitly define how the users behave, or how the scheduler should schedule the jobs. Instead, it exploits class inheritance in C++ only to define the interfaces through which the different entities communicate. For example, to submit a job the user needs to instantiate a job object and use the `void User::submitJob(Job j)` interface to submit it. Site-Sim in turn will notify the scheduler on the job's arrival using the `void Scheduler::arriveJob(Job j)` interface, and will notify the user when the job completes through the `void User::completeJob(Job j)` interface. The primary interfaces of Site-Sim are described in Appendix A.

Site-Sim maintains an internal event queue to guarantee the correct timing and delivery of events, but the exact behavior of the scheduler upon job arrival, or the users' upon job completion is left to the implementor of the interfaces. This is where model accuracy plays a critical role in the representativeness of the workload and the credibility of the simulation results.

While modeling the scheduler is relatively straightforward, modeling the users is obviously much more involved. We therefore implemented the user interfaces in two phases. The first was based on simplistic user models that do not support user arrivals or departures. We used these models in Chapter 3 to demonstrate the importance of the feedback for the evaluation.

In the second phase we implemented much more advanced models that were based on our findings from Chapter 4. This allowed us to simulate realistic user behavior which is affected by the performance of the system, and to exploit it by CREASY in Chapter 5 to demonstrate the feedback importance for schedulers design.

Site-Sim can also run conventional simulations by playing back traces in the *standard workload format* (SWF). These simulations use only one user model that reads the jobs from the input trace, and submits them to the scheduler according to the timestamps. There is no feedback in these simulations; job completion events are simply ignored.

Site-Sim accepts a number of command-line parameters that define its behavior. For those that are not specified it provides default values. Figure 2.1 illustrates a sample execution of Site-Sim. The command line parameters configure it to run five simulations in a loop, each with a different number of users ranging from 50 to 250. The default length of the simulation is six months of user activity; the scheduler to simulate is CREASY.

At the end of the simulation in addition to the system-wide statistics Site-Sim also generates a SWF trace of all the jobs that were submitted in the course of the simulation. We used these

```
#> ./sitesim -users=50-250 -step=50 -scheduler=CREASY
Users  Utilization  Throughput  Jobs/Session  Avg. Response  Avg. Slowdown  Scheduler
  50      0.35         28.0         2.76          12.16           7.01       CREASY
 100      0.55         44.9         2.70          29.91          15.28       CREASY
 150       0.7         54.8         2.69          48.83          19.26       CREASY
 200       0.8         62.7         2.69          72.88          21.98       CREASY
 250      0.89         70.5         2.69          99.44          24.30       CREASY
```

Figure 2.1: *Site-Sim accepts a number of command-line parameters that define its behavior. At the end of the simulation it produces system-wide statistics and a SWF trace that can be used for running conventional simulations.*

Trace	Site	Duration	Procs.	Users	Jobs
SDSC-PAR95-2.1-cln	San-Diego Supercomp. Ctr.	1/1995–12/1995	400	98	53,970
CTC-SP2-2.1-cln	Cornell Theory Center	6/1996–5/1997	430	679	77,222
KTH-SP2-2	Swedish Royal Inst. of Tech.	9/1996–8/1997	100	214	28,489
SDSC-SP2-3.1-cln	San-Diego Supercomp. Ctr.	4/1998–4/2000	128	437	59,725
SDSC-BLUE-3.1-cln	San-Diego Supercomp. Ctr.	4/2000–1/2003	1152	468	243,314

Table 2.1: *Our five traces represent many years of activity by hundreds of users. When available, we use the cleaned versions of the traces.*

traces in Chapter 3 to quantify the prediction errors under the conventional simulations. For these experiments we also used Site-Sim, naturally.

2.2 The Traces: Our Source of Data

System traces provide valuable information on past events that can be used not only for accounting, troubleshooting, and optimizations, but also for driving simulation studies of alternative designs. For the purpose of simulations the traces can be either played-back directly to generate the workload, or be analyzed and modeled first. Modeling is done by fitting probability distributions to the data in the trace, or alternatively by extracting empirical distributions from the data. In both cases, the workload in the simulation is generated by sampling the distributions instead of the raw data in the trace.

Traces of parallel systems are available from the Parallel Workloads Archive in a standard format called the Standard Workload Format (SWF) [12]. For our study we chose five traces, each from a different system. Each trace contains records of the jobs that were submitted to the system by its users over a period of time ranging from one to three years; together they represent many years of activity by hundreds of users and guarantee that our results are not particular to a certain location and time. Table 2.1 lists our traces. When available, we use the cleaned versions of the traces where flurries and other extraordinary activity have been removed [66].

Job records in the traces contain several data fields which can be roughly divided into three sets:

- Those that contain temporal information about the jobs.
- Those that describe the jobs requested and actual resource usage.
- And those that identify the owners of the jobs.

The first set of fields includes the *submit time* of the job, its *wait time* in the scheduler's queue, and its *actual runtime*. The second set includes its requested and allocated *number of processors* and *memory size*, and the third includes the *user* and *group* identifiers of the job's owner. There are also several other fields like the *estimated runtime*, the *return status* of the job, the *queue* and *partition* on which it executed, that may be related to the above sets, depending on the context.

The conventional simulations described in the introduction typically use the submit time, requested processors, runtime, and optionally the estimated runtime to play back the trace. Workload models use distributions that are based on individual or a combination of fields. The jobs' runtime for example can be modeled using solely the runtime field, or in correlation with the processors field to improve model accuracy.

For our site-level simulations we combined the data from all five traces and extracted empirical distributions from the joint data set. We modeled the characteristic of the jobs using the processors and runtime fields, but we also modeled the way the users submit the jobs and their reaction to their jobs when they complete. The later required a careful analysis of the traces to find which performance metric affects the users most and the way it affects their behavior. Being one of the major contribution of this work we dedicate Chapter 4 to this study of user behavior.

Chapter 3

Feedback and Scheduler Evaluations

THE CONVENTIONAL SIMULATIONS presently used to evaluate the performance of parallel-systems schedulers use an open-system model to generate the workload for the evaluation. In many cases recorded traces of real systems are simply played-back, assuming that they are reliable representatives of real workloads, and leading to the expectation that the simulation results accurately predict the schedulers' true performance. We show that the lack of feedback in the simulated workloads results in performance prediction errors that may reach hundreds of percents, and demonstrate how load scaling as currently performed further ruins the representativeness of the workload by generating conditions that cannot exist in reality. As an alternative, we propose a novel simulation methodology in which we model not only the actions of the scheduler but also the activity of users that in reality generate the workload for the scheduler. This advances the simulation in a manner that reliably mimics the feedback effects found in reality, and leads to a better match between the generated workloads and the scheduler's capabilities.

3.1 Background

The conventional simulations presently used to evaluate the performance of parallel-systems schedulers exercise the scheduler using a workload that is made of a stream of incoming jobs. The source for the stream is usually a trace that was recorded on a real system. At the end of the simulation, performance metrics collected for the scheduler predict its performance in reality.

To actually generate the workload, these simulations use an *open-system* model in which the trace is simply played-back according to the timestamps from the trace, and there is *no feedback* in the workload between the arrival of new jobs, the load in the system, and the ability of the simulated scheduler to handle the load. To experiment with 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 indeed reliable representatives of the workloads that are observed by the scheduler in reality. 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 prediction errors 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 naturally exist between jobs in reality.

To get accurate performance predictions and allow for safe load scaling, we propose a novel simulation methodology that we named *site-level simulation*, in which the workload is generated *not* from traces, but *dynamically*, in a manner that reliably mimics the feedback effects found in reality. A site-level model includes not only the scheduler but also the users that in reality, generate the workload for the scheduler. When users wait for their jobs to complete they introduce feedback in the workload that improves its representativeness, since the amount of waiting depends on the load in the system and the scheduler's ability to handle 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 the traces. We found that users' job submissions can be modeled using *batches* which are groups of jobs in which every job except the first is submitted asynchronously to its predecessor i.e. without waiting for it to complete, and with short inter-submission times between the jobs. Furthermore, these submissions can be modeled independently of the characteristics of the jobs themselves. The latter can be derived using a second model that we named the *workpool* model. Together, the two models dynamically generate the stream of jobs to be scheduled.

To experiment with site-level simulations we developed *Site-Sim* — a site-level simulator that integrates users and schedulers under a single simulation framework. Site-Sim enables the easy development of new job submission and workpool models, combining them in various ways to change the characteristics of the workload, and the evaluation of schedulers in a reliable way. It also generates a trace of all the jobs submitted in the course of the simulation, which we use to demonstrate the shortcoming of the conventional simulations.

This chapter is organized as follows: Section 3.2 describes the conventional simulations, the feedback signatures in the traces, and their effect on the evaluations. Section 3.3 introduces our site-level simulations, and describes the job submittal and workpool models we use to generate the workload. Section 3.4 describes Site-Sim and the experiments we performed to demonstrate the importance of the feedback for the evaluation of the schedulers. Section 3.5 surveys related work, and Section 3.6 summarizes this chapter and motivates the next.

3.2 Conventional Simulations

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 the conventional simulations, a model of the scheduler is exercised using a workload made of a stream of incoming jobs. Such a stream is often generated by *playing-back* traces that contain a list of jobs that were actually submitted to and executed on production-use parallel systems.

Within the trace, each job has a *timestamp* that indicates when the job was originally submitted, and several other attributes that describe the resources used by the job. For space-sharing parallel-systems executing rigid jobs, typical attributes include the job's *size* — the number of processors used by job, and its *runtime* — the interval of time during which these processors were occupied and unavailable for use by other jobs. When the trace is used for simulation, these attributes are treated as requirements for resources for the scheduler being evaluated. Jobs may also have a *runtime estimate* — a rough estimation provided by the user at submission time that the scheduler can use to plan ahead [70, 10, 63, 36, 65, 64].

To play-back the trace the conventional simulations use an *open-system* model in which jobs get submitted solely according to the timestamps from the trace and irrespectively of the system's state. Sometimes these timestamps are scaled by a certain factor, so as to increase or decrease the load conditions for the simulation but in either case, there is *no feedback* in the workload between the arrival of new jobs, the load in the system, and the ability of the simulated scheduler to handle the load.

The alternative to the open-system is the *closed-system* model that is characterized by having unconditional feedback in the workload. In this model, the timestamps in the trace are ignored, and new jobs gets submitted only after previous jobs complete. The problem is that this leads to extreme regularity: there are no bursts of activity in the workload which severely limits the optimizations that can be performed, and there is no way to manipulate the load for the evaluations. For these reasons, the conventional simulations adopted the open model in generating the workload.

During simulation statistics are recorded for each individual job which includes the *wait time* — the time the job spent in the scheduler's queue waiting for processors to become available, the *response time* — the time elapsed between submission to completion (wait time + runtime), 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 average for each of these metrics is calculated, and is used to predict the scheduler performance in reality.

3.2.1 Feedback Signatures

We argue that workloads generated by playing-back traces are not reliable representatives of real workloads, because the traces contain a *signature* of the feedback effects that existed between the users and their scheduler when the trace was recorded, and that playing-back this 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 3.1 illustrates that such *self regulation* by users — avoiding submittal of 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 parallel-systems[1]. In these scatter plots, each log is partitioned into weekly slices, and each slice is represented by a dot. 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 traces in the form of *timestamps*. When playing-back the trace according to these timestamps, the generated workload then matches the scheduling policy that was in effect on the traced system, instead of adapting itself to the scheduler being evaluated. This means that the rate of submissions will not decrease if the scheduler fails to handle the load, nor will it increase if it handles the load easily. This causes the performance predicted by the simulation to be highly inaccurate.

[1]Original data and additional information is available in the Parallel Workloads Archive [12].

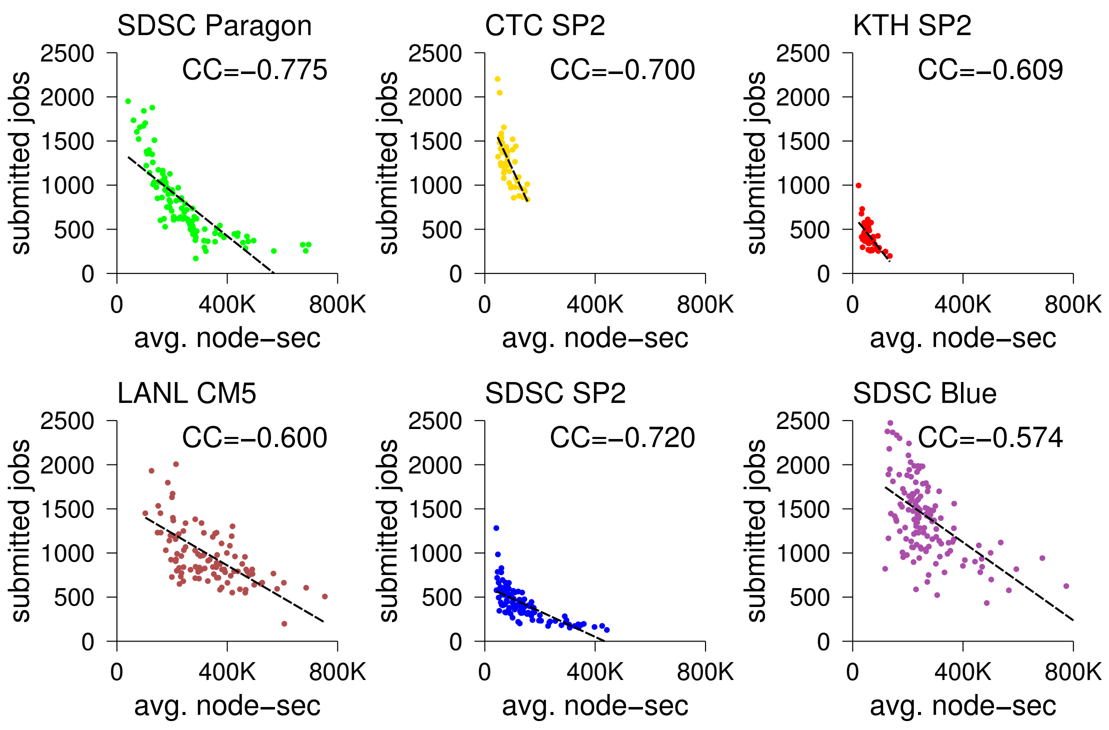


Figure 3.1: *Feedback in real workloads: in weeks where many jobs are submitted they tend to be smaller, and vice versa.*

We also argue that load scaling as currently performed further ruins the representativeness of the workload by generating conditions that cannot exist in reality. One example is the violation of dependencies between the completion of jobs and subsequent submissions. As noted above, users often wait for their jobs to complete before submitting more jobs, which means that some jobs simply cannot reside together in the scheduler's queue. However, when modifying timestamps to scale the load, not only that jobs get submitted while their predecessors are still active, the scheduler may even pick jobs for execution that depend on the completion of other jobs that are still queued.

Finally, the conventional simulations use performance metrics such as the average job response time and slowdown that are only conjectured to be good proxies for user satisfaction. They do not support metrics that quantify user productivity directly, e.g., *throughput*. This is due to the open nature of the simulations, which causes the throughput to be dictated solely by the timestamps from the trace, and not by the actions of the scheduler being evaluated.

The simulation methodology we propose below incorporates feedback into the workload to get accurate performance predictions, but unlike the pure closed-system model described above, it postulates bursts of jobs, allowing the scheduler to effectively perform optimizations. Load scaling is safe and preserves the workload representativeness, and throughput (and hence productivity) becomes a metric that can be directly measured.

3.3 Site-Level Simulations

We propose a novel simulation methodology named *site-level simulation*, to accurately predict the performance of the schedulers. The essence of this methodology is that workloads are generated *dynamically* during the simulation in a manner that reliably mimics the feedback effects found in real sites workloads. In particular, we simulate not just the scheduler but also the users of the system, that in reality generate the workload for the scheduler. During their sessions of activity with the system, users often wait for their jobs to complete and when they do — they introduce feedback in the workload that improves its representativeness.

In addition to the scheduler and the users, a truly complete site-level simulation would also include a detailed model of the system. This may be important because the performance of specific applications may be affected by the system's architecture, or by interference from other jobs [68, 43]. However, such detailed simulations require much more information about applications and take much longer to run. Since we wish to focus on the feedback effects related to the workload generated by the users, we assume that job runtimes are not affected by the system's architecture. This assumption is also made by the conventional simulations.

As illustrated in Figure 3.2, at any given time during a site-level simulation the workload observed by the scheduler is the combination of workloads generated by each individual user session that is active at the system at that time. Each sessions is made of two models: a *job submittal* model and a *workpool* model. The submittal model defines the *structure* of the session, i.e. when jobs are submitted and when the user waits for his jobs to complete. The workpool model defines the actual *characteristics* of the jobs. Furthermore, the fact that these two models are independent of each other contributes to the flexibility of the simulation, and allows to experiment with combinations of models and alter the characteristics of the workload produced by each session. Table 3.1 summarizes the main differences between the conventional and site-level simulations.

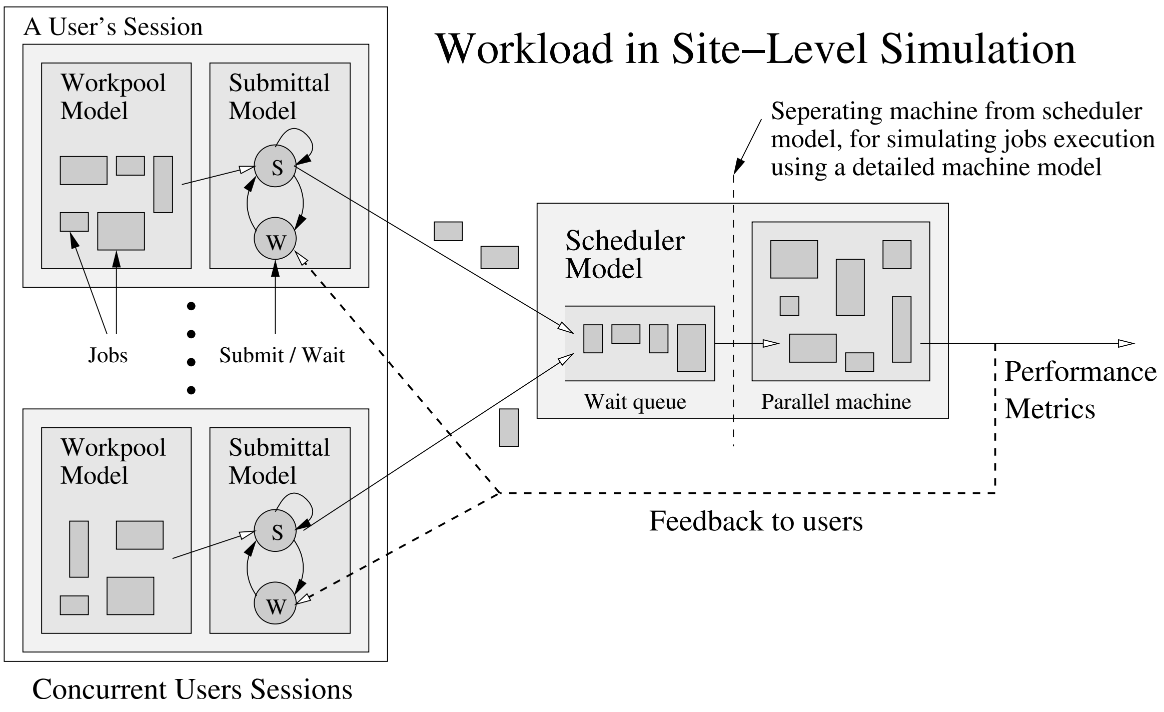


Figure 3.2: *At any given time during a site-level simulation the workload observed by the scheduler model is the combination of the workloads generated by all user sessions that are active at the system at that time. Each such session is made of a job submittal model that introduces feedback into the workload, and a workpool model that defines the characteristics of the jobs being submitted.*

3.3.1 Modeling Job Submissions

Users interact with computer systems in periods of continuous activity known as sessions [26, 1]. For parallel-systems, a session is made of one or more jobs being submitted to the scheduler.

Zilber et al. analyzed several parallel systems traces and classified user sessions [69]. A preliminary step to extracting sessions data was to determine the session boundaries. This was done by setting a threshold on the think times distribution: shorter think times are considered to be think times within a session, while longer ones are considered breaks and the jobs that follow them start a new session.

The CDF of think times in five of these traces, the SDSC-SP2, CTC-SP2, KTH-SP2, SDSC-BLUE and SDSC-PAR95 is shown in Figure 3.3 (the data is available in the Parallel Workloads Archive [12]). The plots indicate that at about twenty minutes of think time the CDF stops its steep climb, which means that a large portion of the jobs are submitted within twenty minutes

Category	Conventional Simulations	Site-Level Simulations
Workload source	Traces	User sessions
Submission rate	Pre-determined by trace timestamps	Dynamic by submittal model
Job characteristics	Job attributes in the traces	Defined by workpool model
Load scaling	Trace (de)-compression	Changing the # of users

Table 3.1: *Main differences between conventional and site-level simulations.*

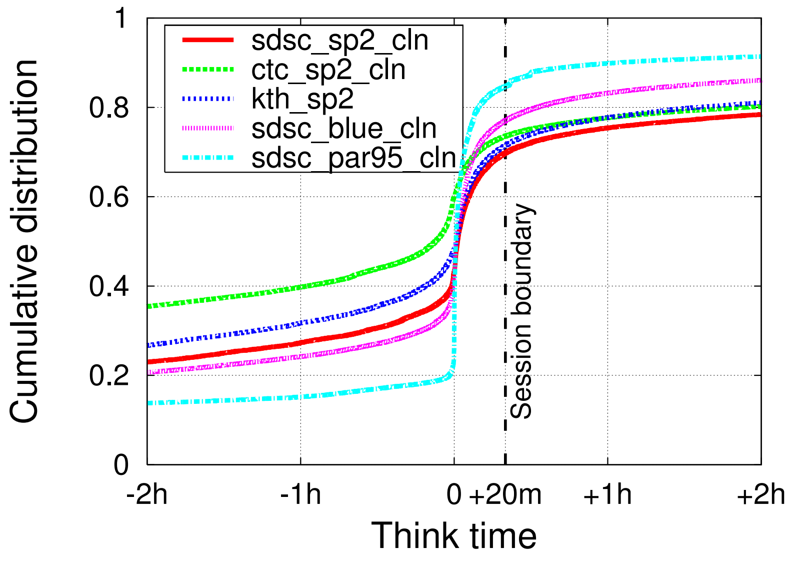


Figure 3.3: *CDF of think times in the five traces: a large portion of the jobs are submitted within twenty minutes from the completion of a previous job, indicating sessions. A major fraction of the think times is below zero, indicating asynchronous submissions.*

from 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 how users submit jobs, is the fact that a major fraction of the think times, over 50% for some traces, are below zero. These negative values stem from the definition of think time as *the time between the completion of a job and the submission of the next*, and indicate that jobs are sometimes submitted before previous jobs complete.

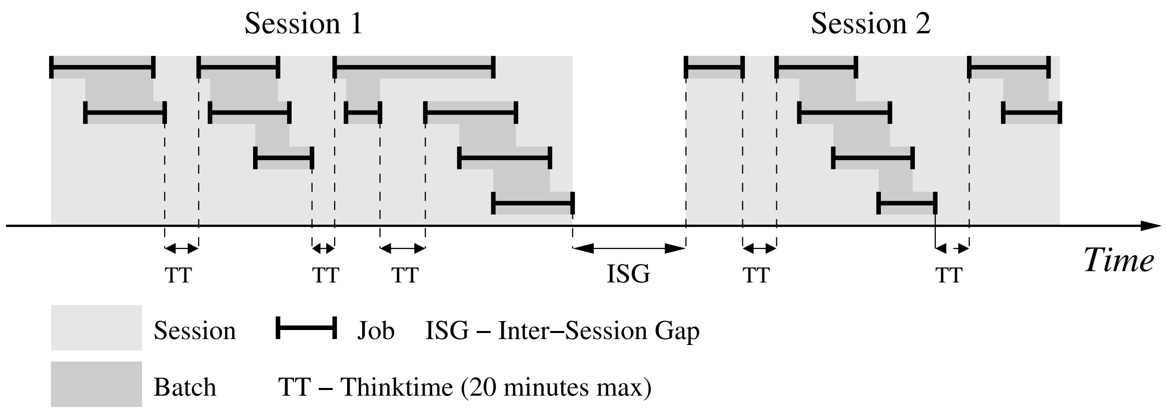


Figure 3.4: *Sessions and batches.*

With respect to modeling these submissions it means that within sessions, users submit jobs either *synchronously* or *asynchronously*. Synchronous submissions are those that may depend on the completion of previous jobs, and hence provide the desired level of feedback which is based on the load in the system and the ability of the scheduler to handler the load. Asynchronous submissions on the other hand are totally independent of previous jobs and the system's state.

We use the term *batch* to denote a set of jobs submitted asynchronously to one another, and the term *batch-width* to denote the number of jobs in each batch. Thus, a single job submitted synchronously is simply a special case of a 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 job submitted last 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 asynchronously to one another, as illustrated in Figure 3.4.

To model the way users submit jobs we thus need three sets of data:

- The distribution of batch widths.
- The distribution of inter-submission times within batches.
- The distribution of think times between batches, up-to twenty minutes.

Data for these distributions can naturally be obtained by analyzing traces of parallel systems. 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 of feedback, we used empirical distributions from the five traces.

Figure 3.5 shows the distribution of batch widths in our five traces. Obviously the distributions are quite similar indicating that this data is representative of user job submissions 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 distribution of inter-submission times for asynchronous submissions within batches, and the distribution of the think times between batches are shown in Figures 3.6 and 3.7, respectively. For our simulations we combined the data from all five traces into a single representative distribution.

3.3.2 Modeling Workpools

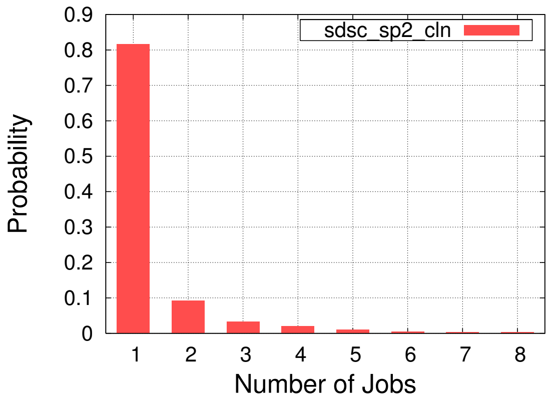
We modeled the above job submissions independently of characteristics of the jobs themselves; the later are derived using a second model called the *workpool model*. Though in principle different sessions can use statistically different models e.g. one model for lightweight daytime jobs and another for heavy nighttime jobs, in our current implementation the models use the same empirical distributions.

A basic workpool model is essentially composed of two distributions that correspond to the two main attributes of rigid parallel jobs:

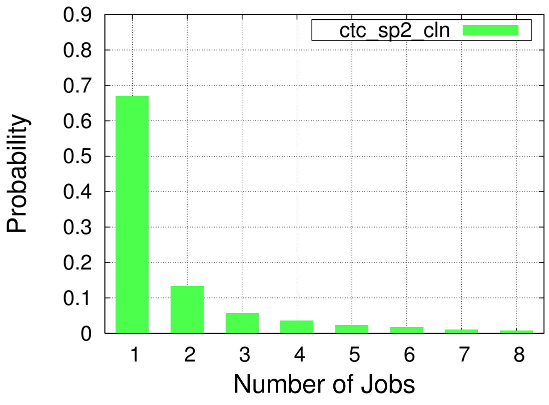
- *Size* — the number of processors the job requires in order to execute, assuming pure space-slicing.
- *Runtime* — the actual time the job will run once the processors have been allocated by the scheduler.

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

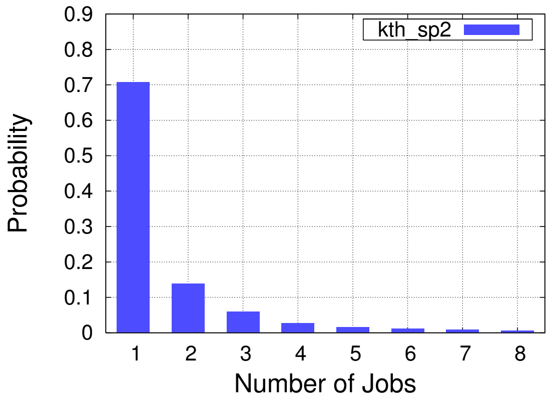
Just like the job submission model, we model workpools using empirical data drawn from the five traces. The distribution of the job sizes in the traces is shown in Figure 3.8. As has been observed before, this is a modal distribution with most jobs using power-of-two processors [20]. The distributions of runtimes and repetitions are shown in Figures 3.9 and 3.10, respectively. Again, we combined the data from all five traces into a single representative distribution to be used in the simulation.



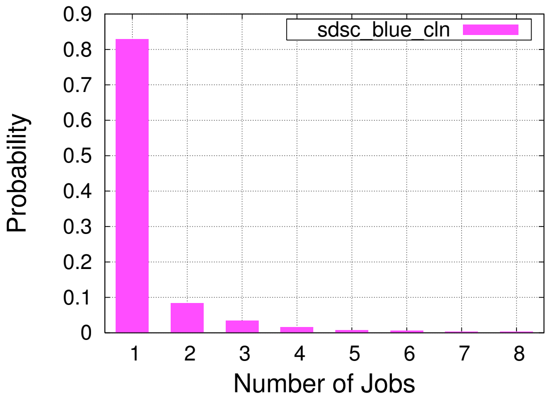
(a) SDSC-SP2



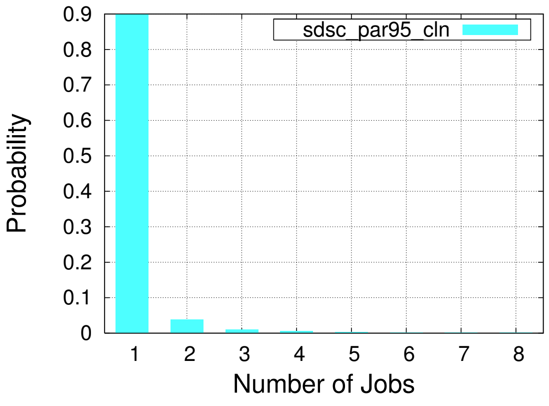
(b) CTC-SP2



(c) KTH-SP2

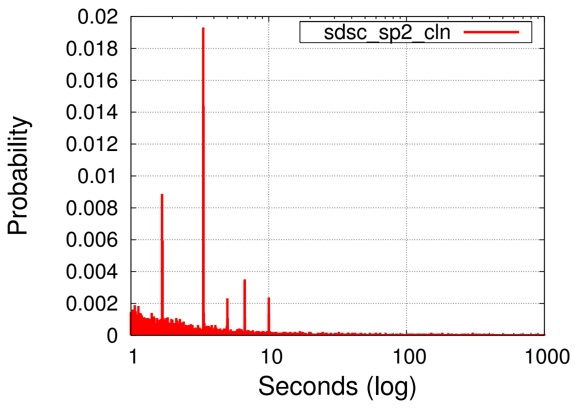


(d) SDSC-BLUE

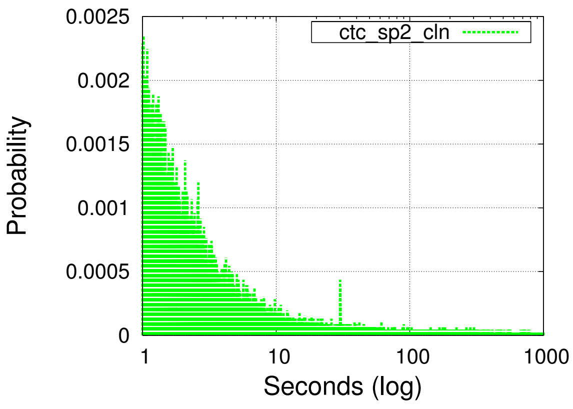


(e) SDSC-PAR95

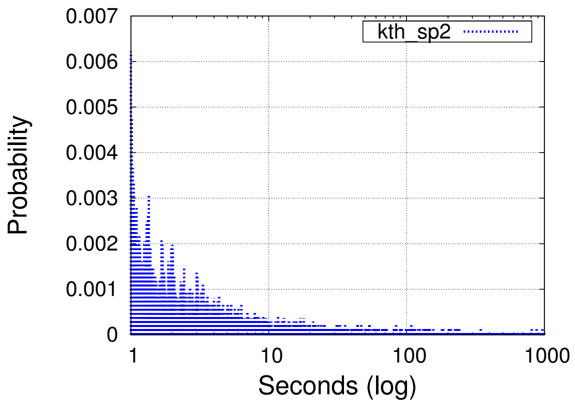
Figure 3.5: *Distribution of batch widths in the five traces.*



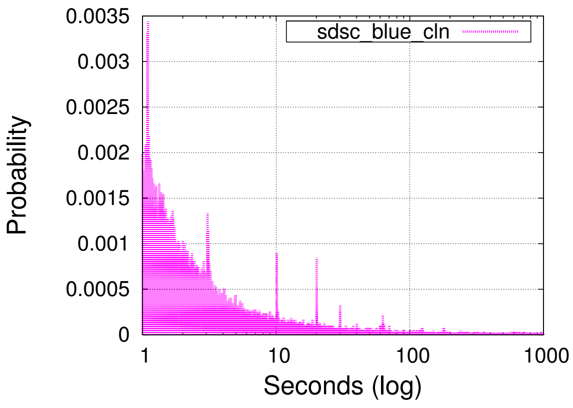
(a) SDSC-SP2



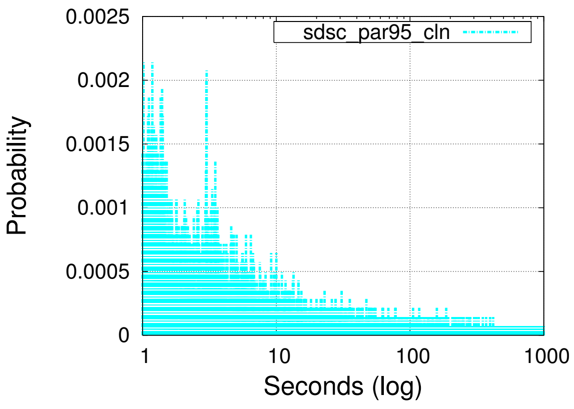
(b) CTC-SP2



(c) KTH-SP2

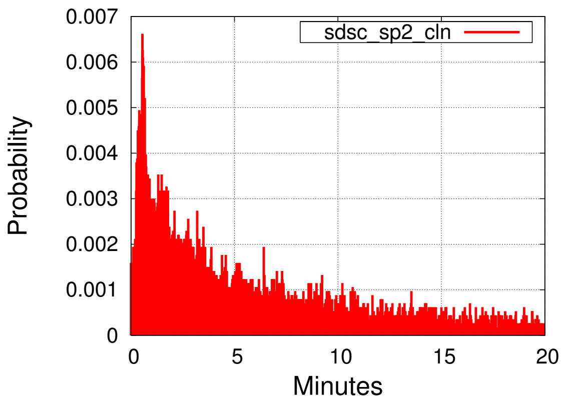


(d) SDSC-BLUE

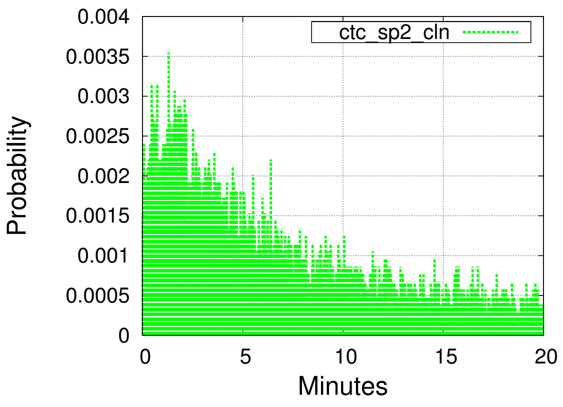


(e) SDSC-PAR95

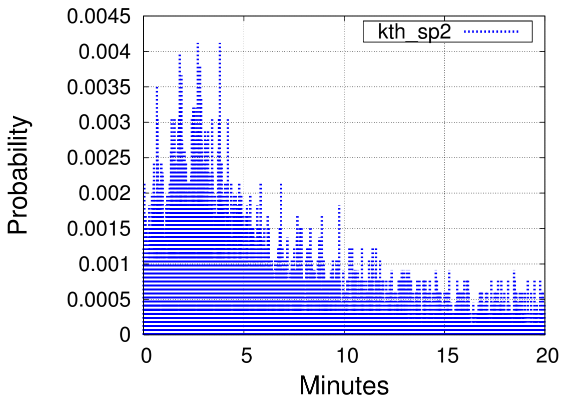
Figure 3.6: *Distribution of job inter-submission times within batches.*



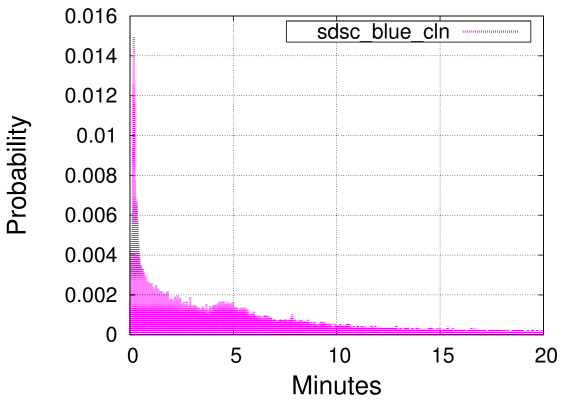
(a) SDSC-SP2



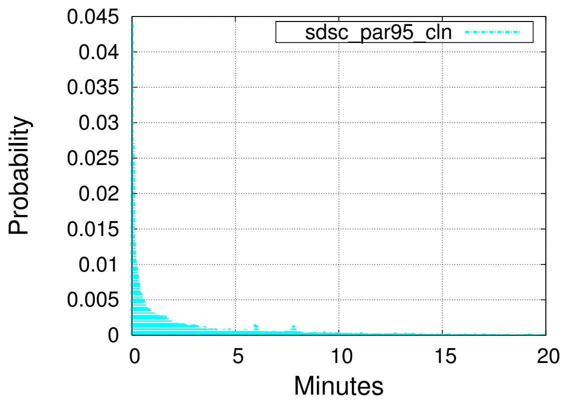
(b) CTC-SP2



(c) KTH-SP2

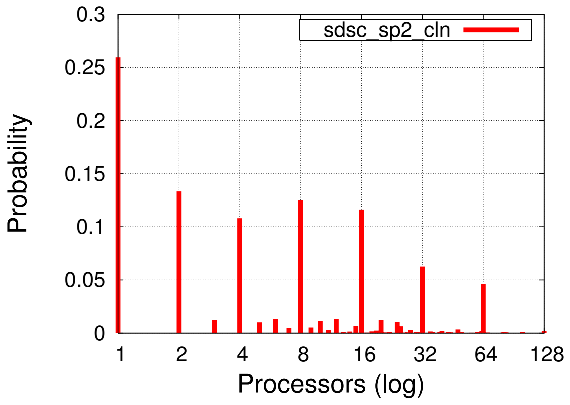


(d) SDSC-BLUE

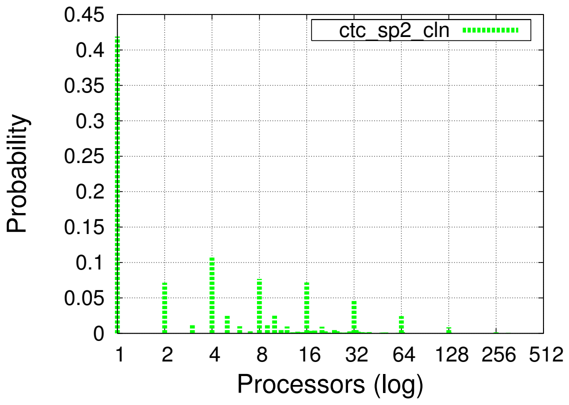


(e) SDSC-PAR95

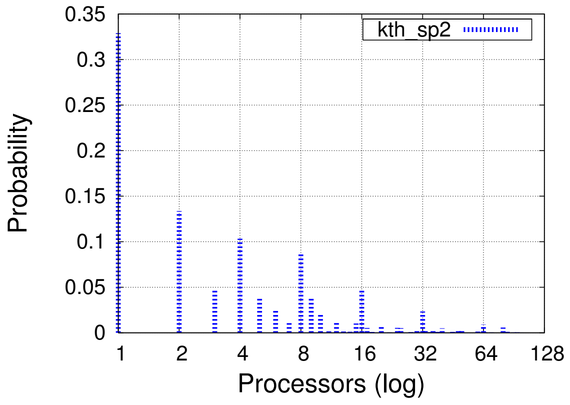
Figure 3.7: *Distribution of think times between batches, up-to twenty minutes.*



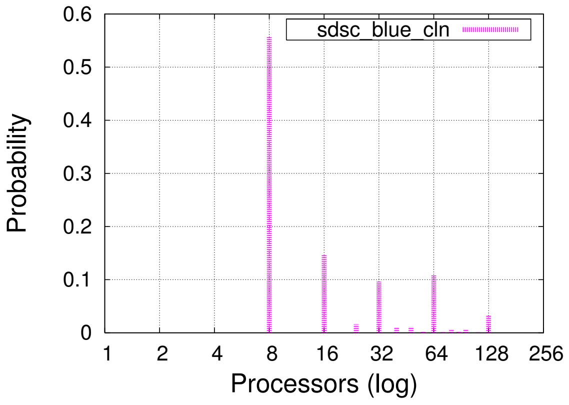
(a) SDSC-SP2



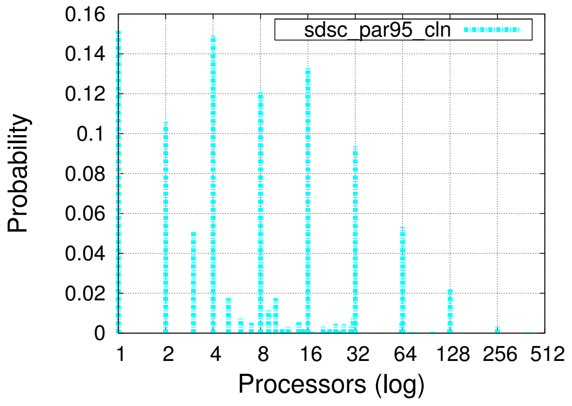
(b) CTC-SP2



(c) KTH-SP2

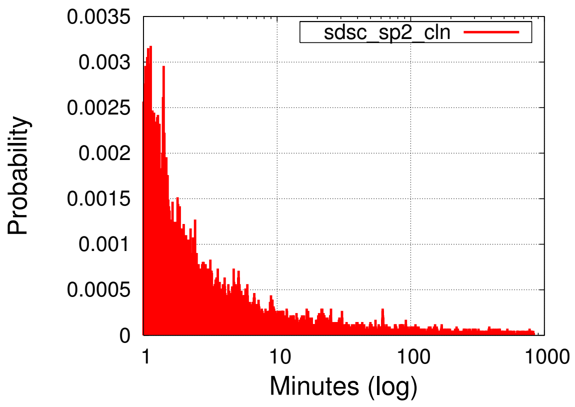


(d) SDSC-BLUE

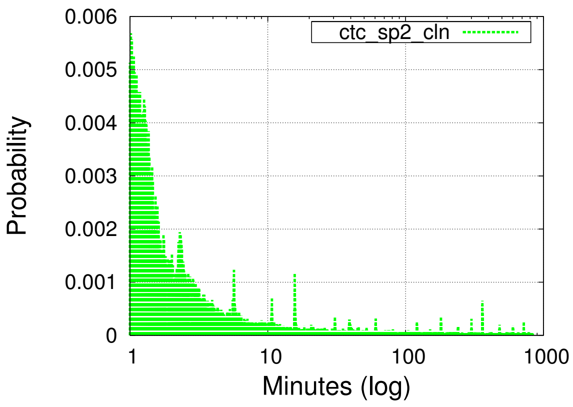


(e) SDSC-PAR95

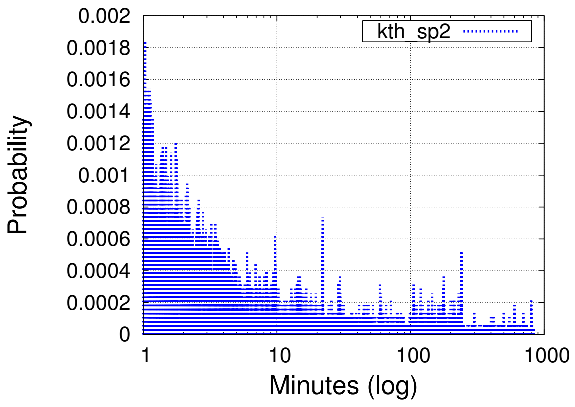
Figure 3.8: *Distribution of job sizes in the five traces.*



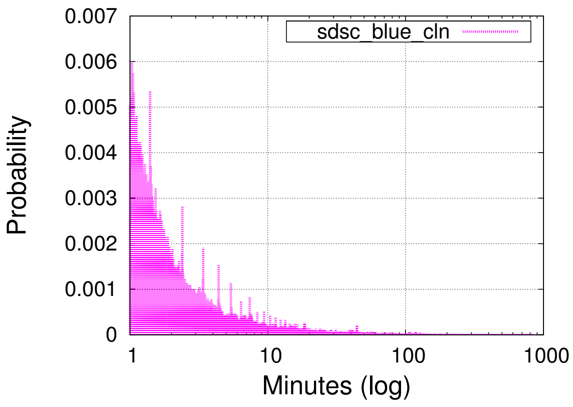
(a) SDSC-SP2



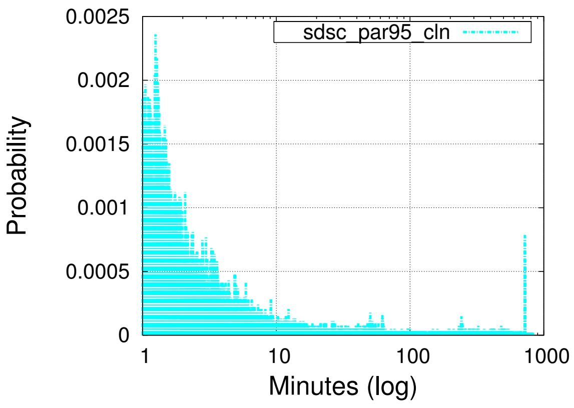
(b) CTC-SP2



(c) KTH-SP2

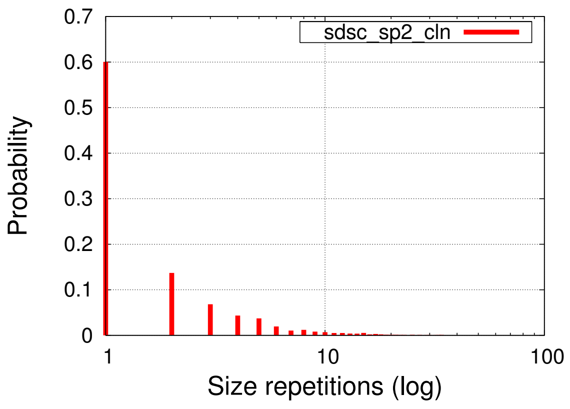


(d) SDSC-BLUE

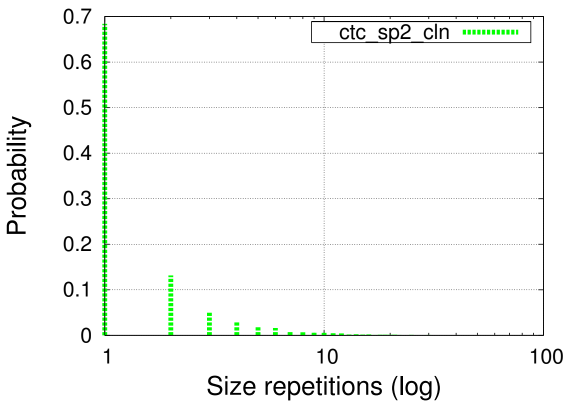


(e) SDSC-PAR95

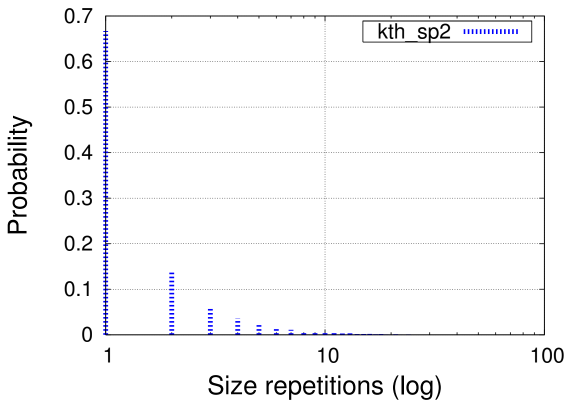
Figure 3.9: *Distribution of job runtimes.*



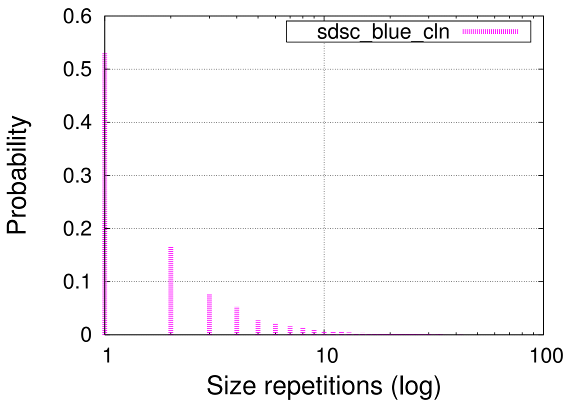
(a) SDSC-SP2



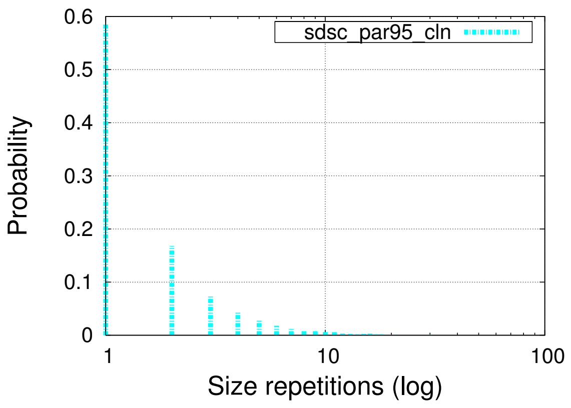
(b) CTC-SP2



(c) KTH-SP2



(d) SDSC-BLUE



(e) SDSC-PAR95

Figure 3.10: *Distribution of job size repetitions.*

3.4 Simulation Results

Our framework for site-level simulations, *Site-Sim*, enables the easy development and combination of job submittal and workpool models, and the reliable evaluation of schedulers using the dynamically generated workloads.

Site-Sim defines two types of entities: *users* and *schedulers*. The users generate the workload for the scheduler during their sessions of activity with the system, and the scheduler in turn accepts the jobs from the users, and schedules the jobs for execution on their behalf. Though in reality the number of concurrent sessions changes dynamically throughout the day as new users arrive and active users depart the system, our simulations so far are limited to using a static number of sessions. Such dynamics are discussed and incorporated into the simulation in the following chapters.

Site-Sim can also run conventional simulations by playing-back traces in the standard workload format (SWF) as defined in *http://www.cs.huji.ac.il/labs/parallel/workload/swf.html*. For these simulations only one user is defined; its job submission model uses the timestamps from the trace, and its workpool model also uses the traces for the jobs' characteristics.

In the simulations reported below we typically define 10 users that use the same job submission and workpool models. Simulation takes a few minutes on an Intel 1.86GHz Pentium processor. At the end of each simulation, Site-Sim generates per-user and system-wide statistics, and also an SWF trace of all the jobs submitted in the course of the simulation. These traces can be used for post-mortem analysis, and also to drive conventional simulations.

As noted in Sections 3.3.1 and 3.3.2, both our job submittal and workpool models use empirical distributions that are based on combining the data from all traces into a single representative distribution. We tested this approach by running long simulations, comparing the simulated distribution against the original distributions from the traces, and validating that they are indeed similar.

3.4.1 Inaccurate Performance Predictions

Users often wait for their jobs to complete before submitting additional jobs. If the system uses a low-end scheduler, their jobs will suffer from long waits in the queue, and the submission of new jobs will be delayed as a result. On the other hand, if the system is managed by a high-end scheduler, queuing times will be short, jobs will complete faster, and the submission of additional work will be accelerated.

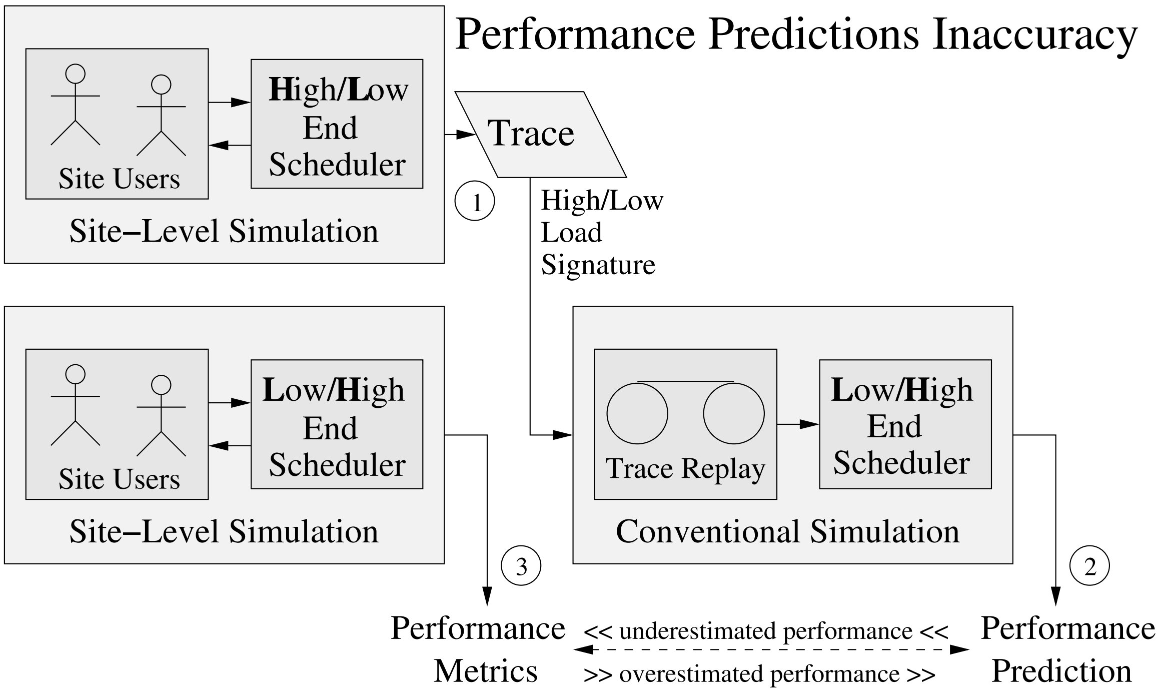


Figure 3.11: *Experiment illustration: (1) A site-level simulation generates a trace with a signature of one scheduler. (2) Conventional simulation driven by that trace predicts the performance of a second scheduler. (3) A site-level simulation of the second scheduler produces performance metrics used to quantify the prediction inaccuracies of the conventional simulation.*

As explained above, system traces contain a signature of the feedback effects between the users and the scheduler in the form of timestamps, which means that different schedulers produce different signatures in the traces. A trace of a high-end scheduler is likely to contain a signature that when played-back using the open-system model, would generate higher loads compared to a trace of a low-end scheduler.

To quantify how inaccurate the performance predicted by the conventional simulations can be, we designed an experiment in which a low-end scheduler is evaluated using a trace of a high-end scheduler, and vice-versa, as illustrated in Figure 3.11. The idea is that the trace of the high-end scheduler will generate such high loads that the low-end scheduler will simply not be able to handle. Since there is no feedback in the workload, the submission rate of the jobs will not decrease, and the simulation will predict poor performance for the low-end scheduler, excessively *underestimating* its true capabilities. The high-end scheduler on the other hand will be able to handle the load from the low-end scheduler trace easily, but since the submission rate will not increase as it would have in reality, its performance will actually be *overestimated*.

Metric (average)	**EASY Site-level**	**FCFS Conventional**	**FCFS Site-level**	**Prediction inaccuracy**
Response [h]	2:22	24:16	3:18	634%
Wait [h]	38m	22:31	1:33	1345%
Slowdown	21.4	1127	78.7	1332%

Table 3.2: *Underestimated performance for FCFS.*

We used Site-Sim to generate the two traces. We ran site-level simulations of 10 concurrently active user sessions that produce work for a parallel-system of 128 processors. For the high-end scheduler we used EASY — a classic scheduler originally developed for the IBM SP system, which employs backfilling to execute jobs from the back of the queue to reduce fragmentation and improve responsiveness [39]. For the low-end scheduler we used plain FCFS (first-come-first-served). Since EASY requires runtime estimates for the jobs, we chose to supply the jobs' actual runtimes, that is, estimates in our simulations were perfectly accurate.

Underestimated Performance: We ran a site-level simulation of the high-end EASY scheduler, and used the resulting trace to drive a conventional simulation of FCFS. Our results indicate extremely poor performance for FCFS: more than 24 hours on average for the jobs to respond, and $22\frac{1}{2}$ hours of wait time in the queue. Obviously, given such performance predictions, one would never consider using FCFS, especially when the average job response under EASY is less than $2\frac{1}{2}$ hours.

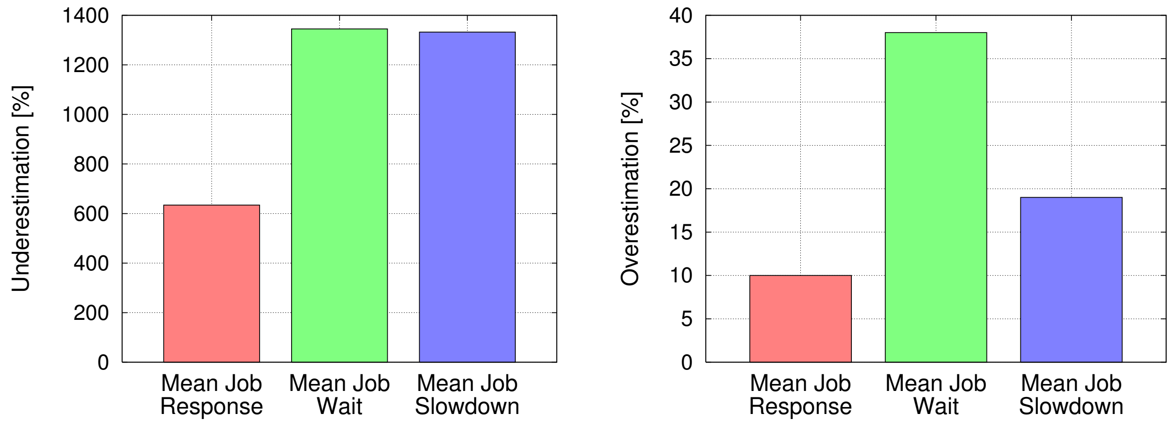
We then repeated the site-level simulation, but used FCFS directly. The results this time indicate that FCFS performs reasonably well considering its limitations: jobs respond on average in $3\frac{1}{3}$ hours (just 39% more than EASY), and the average wait is $1\frac{1}{2}$ hours. Naturally, FCFS is still outperformed by EASY, but it is not as bad as predicted by the conventional simulations. In fact, these simulations underestimated FCFS's average job response time by 634%, its average wait by 1345%, and its average slowdown by 1332%, as summarized in Table 3.2.

Note that the comparison of EASY to FCFS using a site-level simulation is no longer based on serving the same *jobs* as in conventional simulations, but on serving the same *user population*. FCFS did actually serve fewer jobs, but its throughput was only 10% less than EASY's for the simulated load of 10 concurrent sessions. The throughput metric is further discussed in Section 3.4.3.

Metric (average)	FCFS Site-level	EASY Conventional	EASY Site-level	Prediction inaccuracy
Response [h]	3:18	2:08	2:23	10%
Wait [h]	1:33	23m	38m	38%
Slowdown	78.7	17.3	21.4	19%

Table 3.3: *Overestimated performance for EASY.*

Overestimated Performance: We repeated the above experiment, in the opposite direction. We ran a 10-user, site-level simulation of FCFS, and used the trace to drive a conventional simulation of EASY. The results indicate far too optimistic performance for EASY: two hours on average for the jobs to respond, and just 23 minutes of wait in the queue. A site-level simulation of EASY with feedback however indicates that its average job response time was overestimated by the conventional simulations by 10%, its average wait by 38%, and its average slowdown by 19%, as shown in Table 3.3.



(a) Underestimated performance for FCFS (b) Overestimated performance for EASY

Figure 3.12: *Underestimated and overestimated performance in conventional simulations.*

To summarize, underestimated performance due to lack of feedback in the workload tends to be much larger than the overestimated performance. The response time metric is the least sensitive to performance prediction inaccuracies, while the wait is the most sensitive; slowdown is somewhere in between. All this is summarized in Figure 3.12.

3.4.2 Safe Load Scaling

One of the important uses of simulations is to predict the system performance under different load conditions. In the conventional simulations, load is scaled by modifying the timestamps in the trace before the simulation begins. The timestamps are multiplied by a constant factor which causes the gap between the jobs to increase of decrease, depending on whether the factor is greater or smaller than one, respectively. When played-back using the open-system model, this changes the rate in which jobs are submitted to the scheduler, and alters the load conditions for the simulation.

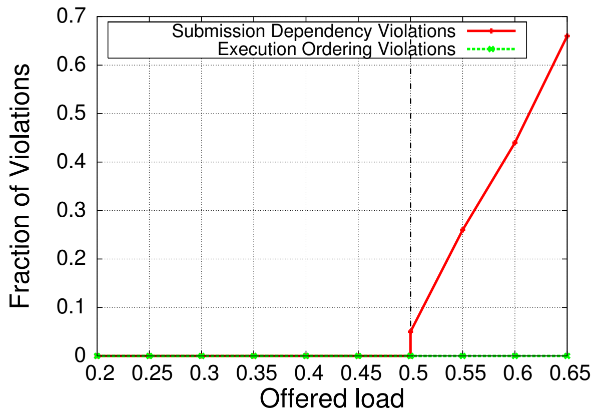
Such a modification to the traces however, may effect the representativeness of the workload by generating conditions that cannot exist in reality, for example, violation of dependencies that naturally exist between jobs. Since users often wait for their jobs to complete before submitting more jobs, some jobs simply cannot reside together in the scheduler's queue, but when modifying the timestamps to simulate higher loads, not only that such jobs do get to coexist together, the scheduler may even pick jobs for execution that depend on the completion of older jobs that are still queued.

To quantify these violations, we ran two site-level simulation, the first with the FCFS scheduler, and the second with EASY, both with 10 concurrently active user sessions. We recorded in the output traces information on the dependencies between the jobs, and used the traces to drive conventional simulations of both FCFS and EASY. For these simulations, we modified the timestamps to simulate a range of offered loads[2]. For FCFS we simulated offered loads ranging from 0.2 to 0.65, and for EASY we simulated loads from 0.2 to 0.9, since EASY can sustain higher loads. We instrumented Site-Sim to count the number of *submission dependency violations* — the number of times jobs get submitted to the scheduler but depend on the completion of other jobs that are still active, and the number of *execution ordering violations.* — the number of times the scheduler picks jobs for execution that depends on the completion of jobs that are still queued.

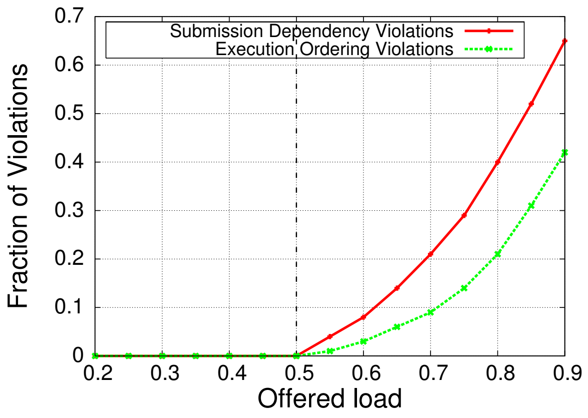
Figure 3.13 shows the fraction of jobs whose submission or execution involved violations. In all sub-figures, the dashed vertical line shows the original load from the trace, prior to any timestamp modification.

For the FCFS trace in sub-figures (a) and (b), we see that for both schedulers, the fraction of submission dependency violations starts to increase at the offered load of 0.5. For the FCFS

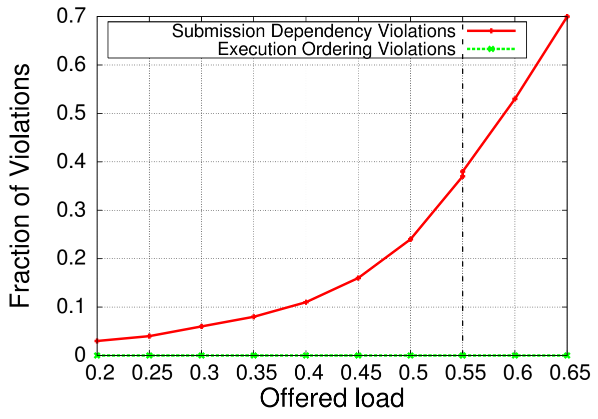
[2]Offered load is the load imposed on the system in an open-system model.



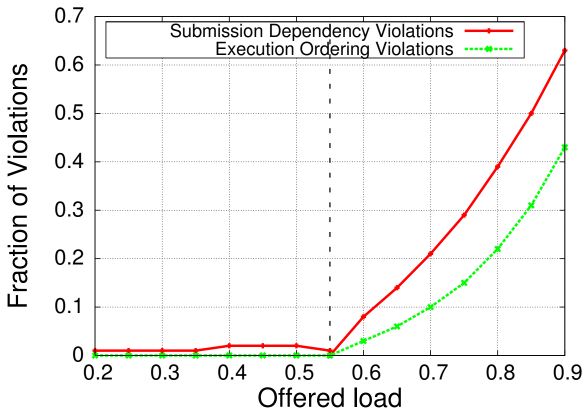
(a) FCFS trace / FCFS scheduler



(b) FCFS trace / EASY scheduler



(c) EASY trace / FCFS scheduler

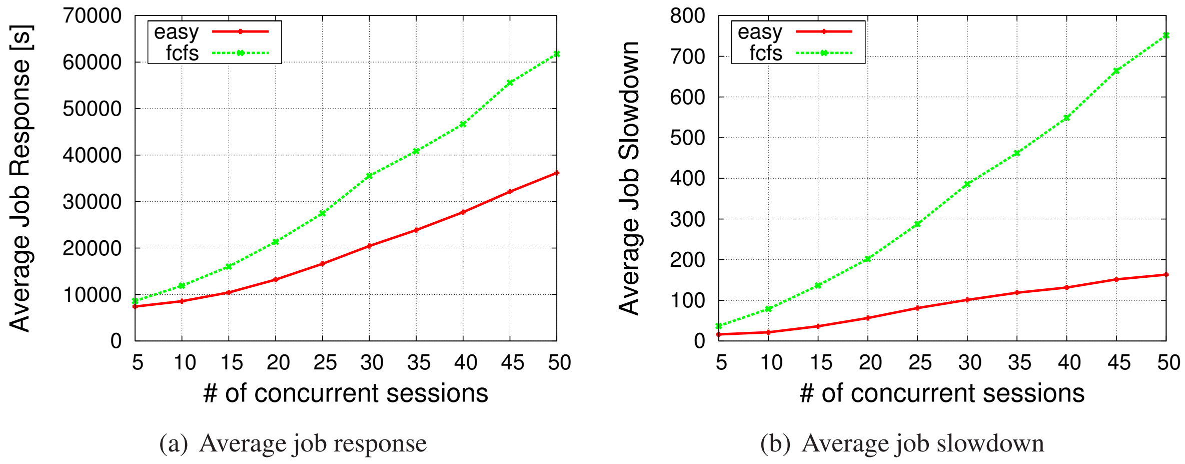


(d) EASY trace / EASY scheduler

Figure 3.13: *Submission-dependency and Execution-ordering violations.*

scheduler the increase is almost linear, while for EASY it is hyperbolic. Neurally, since FCFS executes jobs purely according their arrival order, there are not execution ordering violations under FCFS.

For the EASY trace in sub-figures (c) and (d), we see that when simulating FCFS the fraction of submission dependency violations starts to increase far before the dashed line which represents the original load from the trace. This is due to the high-load signature in the EASY trace, that is too much for FCFS to handle, even when scaling the timestamps backwards, to simulate lower loads. We also observe that few submission dependency violations occur at reduced loads even under the EASY scheduler, but when reaching the dashed line, there are no violations of either type under EASY. This is because the offered load equals the original load from the trace, so the lack of feedback in the conventional simulations has no effect on the representativeness of the



(a) Average job response (b) Average job slowdown

Figure 3.14: *Load scaling in site-level simulations: the results for the wait time metric are similar to the response, only slightly shifted down.*

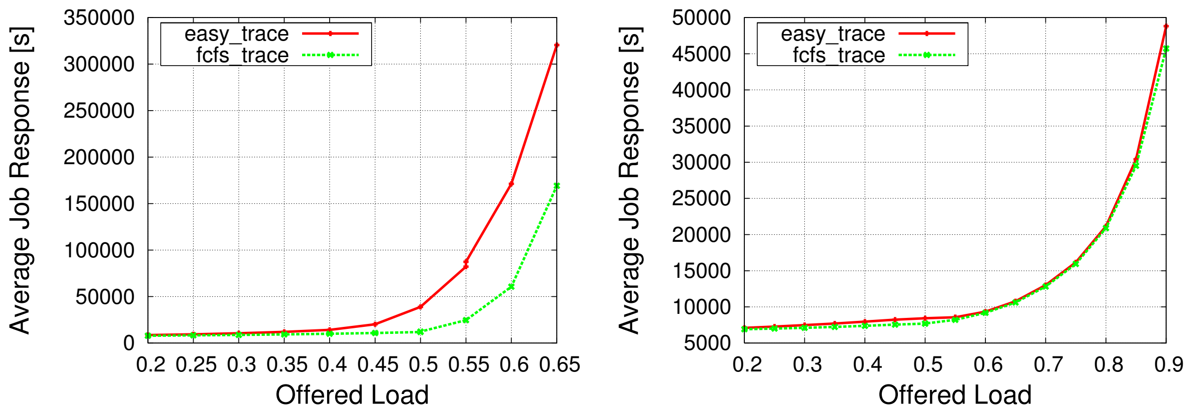
workload.

Under site-level simulations on the other hand, the load is scaled by simulating a higher number of users, which effectively increases the number of concurrently active sessions and hence the rate in which jobs are submitted to the scheduler. There are *no* violations of any kind, since the feedback in the workload ensures that a job that depends on the completion of other jobs will not be submitted until its predecessors complete and after a period of think time.

Figure 3.14 compares the performance of EASY and FCFS as a function of the number of concurrent sessions in a *site-level* simulation. As expected, EASY always outperforms FCFS, but the more interesting phenomenon is the shape of the curves; instead of the curves often seen in open-system simulations which tend to infinity when the load approaches the saturation point, as shown in Figure 3.15, the degradation in performance when feedback is involved is much milder, as the feedback curbs the creation of additional work.

3.4.3 Quantifying Productivity

An important goal of any parallel-system scheduler is to promote the productivity of its users, but the conventional simulations lack a metric that directly quantifies productivity. The only metric that may be considered related is the *average system utilization* which implicitly reflects the amount of work that was performed. However, utilization in open-system simulations is determined by the rate in which new jobs are submitted, and not the actual performance of the scheduler.



(a) FCFS scheduler (b) EASY scheduler

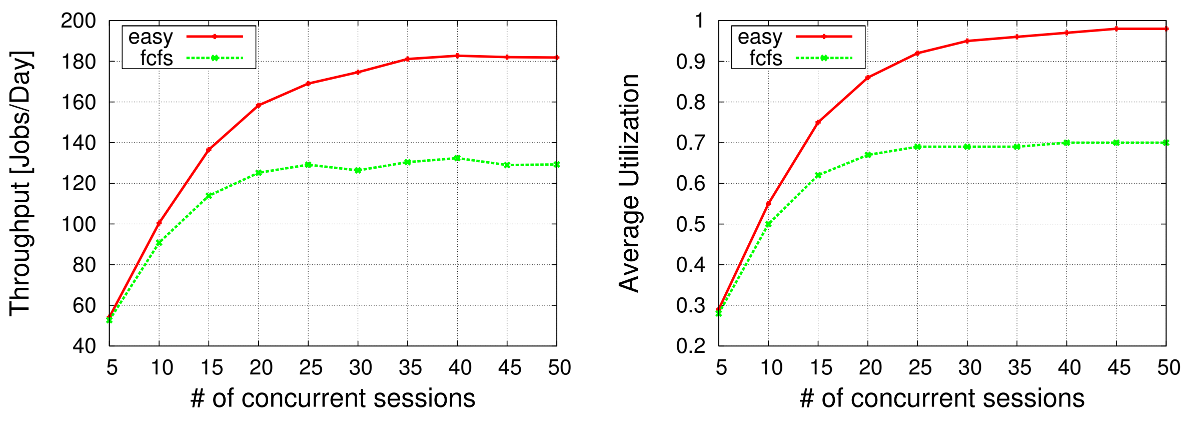
Figure 3.15: *Load scaling in conventional simulations: the curves tend to infinity when the load approaches the saturation point, due to the open nature of the simulation.*

Site-level simulations on the other hand provide a metric that quantifies productivity directly: the *job throughout* which is defined as the number of jobs executed by the system in a given time frame. Figure 3.16(a) shows the average job throughput in a 24 hours time frame under FCFS and EASY. For comparison, we also show the utilization of the system in figure 3.16(b). As can be seen, the two metrics are highly correlated, and both level out when the system becomes saturated. Beyond this point adding more concurrent sessions does not contribute to the throughput, but only increases the average response time of the jobs.

3.5 Related Work

Parallel-systems schedulers have been traditionally evaluated using trace-driven, open-system simulations, in which the arrival rate of the jobs in the workload is already given, and is not affected by the performance of the system [60, 30, 45, 59, 67, 35, 54, 61, 46]. The alternative is to use models to generate the workload, but most models either try to reproduce the arrivals already found in the traces [6, 29, 11, 42], or even use a Poisson model to further simplify the arrival process of the jobs [20, 16]. Models involving feedback have been suggested, but in other contexts.

Ganger and Patt in their work on I/O subsystems evaluations observed that neither the open nor the closed-system model are satisfactory in their pure form, because real workloads are a mix with only some items being critical for progress [22]. This led to work by Hsu and Smith



(a) Average job throughput (b) Average utilization

Figure 3.16: *Utilization and throughput: implicit and explicit productivity measures.*

who added feedback to I/O traces in order to study the effectiveness of various I/O optimization techniques [27]. M. Borella studied the influence of hosts processing speeds on network packets inter-arrivals in the context of gaming traffic [4]. Schroeder et. al observed that neither the closed nor the open system model are entirely realistic, and that real-world applications often fit best into partly-open models [49].

While the mechanics of using traces in simulations is straightforward, there is more than one way to model users to 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 interactive systems workloads [24]. Hlavacs et al. suggested a layered user model made of sessions, application, and commands, and demonstrated its use for network simulations [26]. Arlitt analyzed the internal structure of user sessions to the 1998 World Cup Web server [1]. Liu et al. presented a Web traffic model at the session level to evaluate the performance Web servers [40], and Krishnamurthy et al. developed synthetic workloads with inter-request dependencies to correctly stress-test session-based systems [33]. Zilber et al. presented a comprehensive study of parallel-systems traces based on users and sessions, which can complement ours by defining diverse submittal and workpool models [69].

All above models can be used to generate a more representative workload, but they lack a description of how real users react to the performance observed from the system. In the next chapter we show how we uncover the effect of the system on its users from the system traces, and develop more realistic models of the users to be used in simulation.

The basic batch scheduling algorithm is First-Come-First-Serve (FCFS), in which jobs are considered in their order of arrival [50]. The poor system utilization of FCFS led researchers to explore alternatives to improve performance. Backfilling is one such optimization, that allows small jobs from the *back* of the queue bypass larger jobs that arrived earlier, to *fill* holes in the schedule.

Backfilling was first implemented in a production system in the context of EASY, the Extensible Argonne Scheduling sYstem for the IBM SP1 system [39]. This specific version was based on aggressive backfilling in which only the first job in the queue gets a reservation. The obvious alternative is to use conservative backfilling in which every job gets a reservation, and which also produces a more predictable schedule. Mu'alem and Feitelson compared the two approaches and pointed that their relative performance may actually depend on the workload [45].

3.6 Summary

We have shown that user sessions on parallel-systems 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 completes. This imparts a measure of feedback on the workload generation process and results in a better match between the workload and the scheduler capabilities. Ignoring this feedback 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 trace was originally recorded.

The simulations presented in this chapter were limited to using a static number of active sessions. Real users however, arrive and depart the system at different times and in response to the performance observed from the system, so the number of concurrently active sessions changes dynamically throughout the day. The next step is therefore to incorporate these dynamics into the simulation, but this requires a preliminary study of user behavior in order to understand which aspects in the performance of the system affect the users, and what exactly is the effect.

Though intuitively it seems that such a study requires research in psychology, we found that it is possible to uncover the users' behavior patterns directly from the traces of systems, without conducting live experiments with real users. Chapter 4 presents the novel trace analysis methodology that we developed for this purpose, and our findings regarding the effect of the system on its users. In Chapter 5 we incorporate these findings into the simulation and introduce much more realistic user models which we use to demonstrate the importance of the feedback for the design of the schedulers.

Chapter 4

Understanding User Behavior

INTUITIVELY, it seems that understanding how the performance of a system affects its users requires research in psychology and the conduction of live experiments. We demonstrate that it is possible to uncover the effect from traces of systems. In particular, we show that the behavior of users of parallel systems is correlated with the response times of their jobs, not the slowdown as was previously assumed. We also show that response times affect the decision of users to continue or abort their interactive session with the system, and that this may relate to expectations the users develop. Though this study was conducted in the context of parallel systems, we believe our findings are general enough and may pertain to other types of systems as well.

4.1 Background

Understanding how the performance of a system affects the behavior of its users helps improve system design. In the context of parallel systems, for example, it will allow the design of better job schedulers, with the goal of satisfying users by exploiting knowledge about user behavior to better plan future actions.

Intuitively, it seems that exploring user behavior requires research in psychology and the conduction of live experiments with real users. The problem is that live experiments are often impractical to conduct; few users have the time or patience to actually record the reasons for their behavior patterns.

We suggest a novel methodology to uncover the effect on users from *traces* of the system. The traces we use contain records of jobs that were actually submitted by the users, scheduled, and finally executed on real, production-use parallel systems. We show that using these traces we are even able to reason about user motivation, not just about the causal relationship between performance and behavior.

The performance of the system can be measured using different metrics, all which are assumed to be important to the users. In particular, the response time of jobs (the time from submission to termination), and the slowdown (the response time normalized by the actual execution time) are two metrics often used in performance evaluations. The first question is therefore, *which* of these metrics is most important to the users in the sense that it affects their subsequent behavior.

Intuitively, the slowdown of jobs is important to users because it reflects the degree to which the performance they actually observed from the system met their expectations. For example, it may be fine for a 10-minute job to wait 5 minutes in the queue (a slowdown of 1.5), but for a 1-minute job to be delayed 14 minutes (same response time of 15 minutes, but slowdown of 15), may be considered unacceptable. Somewhat surprisingly, we found that user behavior is strongly correlated with the response time of their jobs, not the slowdown. This finding calls for a reassessment of suggestions that jobs should be prioritized according to their slowdown [23].

The next question is *how* exactly response times affect the behavior. In reality, users tend to submit several jobs one after the other in periods of activity that are known as sessions. Previous work had already discovered how session data can be identified and extracted from the traces [69]. We found that the decision of the users to continue submitting jobs, or alternatively to abort their session, is affected by the response time of their jobs. Specifically, we show that the higher the response time, the higher the probability for the user to abort his interactive session with the system.

The third and final question this chapter answers is *why* this happens. It is well known that user behavior is affected by expectations [58], but unfortunately, such informations does not appear explicitly in the traces. Instead, we show that it is possible to isolate specific scenarios in the traces. In particular, we examined the scenario where response times met the expectations of the users, and the scenario where they did not. We found that although the users' perception and motivation are different in the two cases, their actual behavior happens to be very similar.

This chapter is organized as follows: Section 4.2 describes the traces we used for our analysis. Section 4.3 answers the question of *which* performance metric is most important to the users. Section 4.4 answers the question of *how* that metric affect their behavior, and Section 4.5 answers the question of *why* this happens. Section 4.6 surveys related publications, and Section 4.7 summarizes the chapter.

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

Table 4.1: *The five traces we used for our analysis: together, they represent many years of activity by hundreds of users.*

4.2 Trace Data

The data we used for our analysis come from traces that contain records of jobs that were submitted and executed on a variety of large-scale parallel machines over periods ranging from one to three years. Each job record contains several fields, four of which are relevant for our study: the *user* who submitted the job, the *time of submission*, the job's *wait time* in the scheduler queue, and the job's actual *execution time*, once it got started. The first field allows us to analyze the data on a user basis. The other three fields allow us to find when each job terminated, and to calculate its *response time*, its *slowdown*, and the *think time* between the termination and the submission of the next job by the same user.

We used five traces to ensure that our results are not particular to a certain location and time: together, they represent many years of activity by hundreds of users. They are listed in Table 4.1, and are available on-line from the Parallel Workloads Archive [12]. When available, we use the cleaned versions of the traces, where flurries and other extraordinary activity have been removed [66].

4.3 Metrics Correlation with User Behavior

When a user submits a job for execution, this is typically not an isolated event. Rather, users tend to submit several jobs one after the other. In many cases there is a dependency between successive jobs: when a job terminates, the user examines its result, makes corrections and adjustments, and submits another job. The time between the completion of a job and the submission of the next job is known as the *think time*.

The system scheduler, in turn, accepts these jobs from the users and places them in its wait queue. When resources become available, it scans the queue and selects jobs for execution according to some prioritization criteria, and subject to possible reservation constraints.

Obviously, the scheduler's actions depend on the jobs submitted by the users, but user behavior is also dependent on feedback from the scheduler. An efficient scheduler that streams jobs through the system at a high rate encourages users to submit more jobs, while an inefficient scheduler that causes resources to be wasted and jobs to be delayed discourages the submittal of additional work [55].

While existence of such a feedback to the users is intuitively clear, the effect on their behavior is not. The performance of the system can be measured using different metrics, all of which may be assumed to be important to the users. The challenge is to find the metric that is really important to the users in the sense that it affects their subsequent behavior.

We focus on the *response time* and *slowdown* of the jobs. The response time of a job is the time elapsed from its submission to termination; it is the sum of the time it spent waiting in the scheduler's queue and the time it actually executed. Intuitively, response time is important to users because they must wait for their jobs to terminate before they can examine the results and submit more jobs.

Slowdown is the response time normalized by the actual execution time. It is also intuitively important to users because it reflects the degree to which the performance they actually observed from the system matched their expectations: A slowdown value that is near 1 indicates that the job response time was close to its execution time, whereas a high slowdown value indicates that response time had lengthened far beyond what would have been expected based on the job's actual runtime.

Finding a metric that reflects the behavior of the users is more challenging. Metrics like the average job submission rate are not useful because averages necessarily mix multiple effects with multiple responses. Instead, we propose to use the *think times* that follow jobs in the traces. The rationale is that think times capture the user's immediate response to the job that has just terminated. A short think time indicates that the user was waiting for his job to complete, that he is satisfied with performance, and intends to continue the interaction. A long think time indicates that the user was probably not waiting for the job, possibly because he had given up on using the system.

Because different jobs receive different levels of service, and different users react differently to their jobs, the response times, slowdowns, and think times of the jobs in the traces exhibit large variance. Tabulating the interaction on an individual job basis in this case is not useful.

Response time bin	**R1**	**R2**	**R3**	**R4**	**R5**
Response range	0–5	5–15	15–45	45–135	135–∞

Table 4.2: *Response-time bins (ranges are in minutes).*

Slowdown bin	**S1**	**S2**	**S3**	**S4**	**S5**
Slowdown range	1–1.41	1.41–2	2–4	4–16	16–∞

Table 4.3: *Slowdown bins.*

Instead, we partition the jobs into classes that represent different levels of feedback to the users, and study the aggregate user reaction to the jobs in each class.

The classes are obtained by dividing the jobs into five bins, once according to the response time metric, and again according to the slowdown. The levels of feedback represented by the response-time bins are "very fast response", "reasonably fast response", "medium response", etc. and the levels of feedback represented by the slowdown bins are "very low slowdown", "low slowdown", and so on. The boundary points between the bins are roughly logarithmic, as shown in Tables 4.2 and 4.3. This has the advantage of resulting in classes of approximately the same sizes in terms of the number of jobs assigned to each.

Once the jobs are grouped in bins, we can analyze the think times that follow the jobs on a per-bin basis, and examine reaction of the users to the different classes of feedback. Figure 4.1 shows the *median* think times for the five response-time bins, and the five slowdown bins.

In the case of response time, the relationship between performance and subsequent user behavior is clear and consistent for all five traces studied. Jobs in bin 1, which had response times of up to 5 minutes, have the smallest median think time — also about five minutes. The median think time is larger — up to 20 minutes — for bin 2, 20 to 60 minutes for bin 3, etc. This means that users' subsequent behavior is correlated with the response time of their jobs: the faster their jobs respond, the quicker users submit additional jobs.

In the case of slowdown, on the other hand, the relationship is much less clear. First, the results for the different traces are highly dispersed, with the highest median think time a factor of 20 or 30 larger than the lowest one for each bin. Second, the order of the traces varies too; for example, the SDSC SP2 trace exhibits the lowest think time median for bin 2, and the highest median for bin 4. Finally, some of the results are non-monotonic: in the SDSC Paragon trace, the think times following jobs in slowdown bin 4 are lower than those following jobs in bins 1, 2, and 3. Consequently, there is no easy and general way to characterize the relationship between

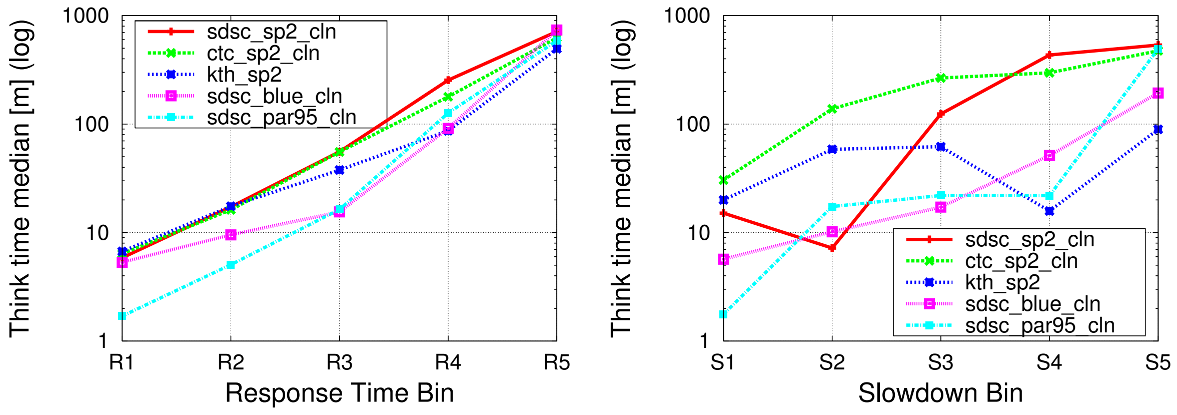


Figure 4.1: *Correlation of performance metrics with user behavior: Left: high response times correlate with longer think times. Right: for slowdown there is only a weak correlation.*

the jobs' slowdown and the subsequent user behavior.

In the above, we considered the response-time bins to be homogeneous, but in reality, different jobs in the same bin may have far different wait and execution times. This means that the slowdown of the jobs within the same response bin exhibit a large variance. For example, a 15-minutes job that waited 5 minutes in the queue and a 1-minute job that waited 19 minutes both belong to the same response bin R3, since both responded in 20 minutes. The question is whether these two jobs indeed trigger a similar user behavior, even though the first has a slowdown of 1.33 and the second has a slowdown of 20.

To answer this question, we need to examine our response bins more closely. The natural way to do this is to simply divide each bin into sub-bins based on the slowdown metric. We used the same ranges as for the original slowdown bins.

Figure 4.2 illustrates these bins graphically. The horizontal axis represents the jobs' wait time, and the vertical axis represents their execution time. The response bins are bounded between the solid lines, and create diagonal regions going from top-left to bottom-right. The slowdown bins are bounded between the dashed lines, and create radial regions emanating from the origin. The intersections of the two types of regions represent the sub-bins. We named these sub-bins according to response and slowdown bins their jobs belong to, so for example sub-bin R3S1 holds jobs whose response time belongs to response bin R3, and whose slowdown belongs to slowdown bin S1, etc.

Table 4.4 shows the median think time following the jobs in each of the sub-bins, using data combined from all five traces. As expected, this exhibits a strong dominant effect of the response

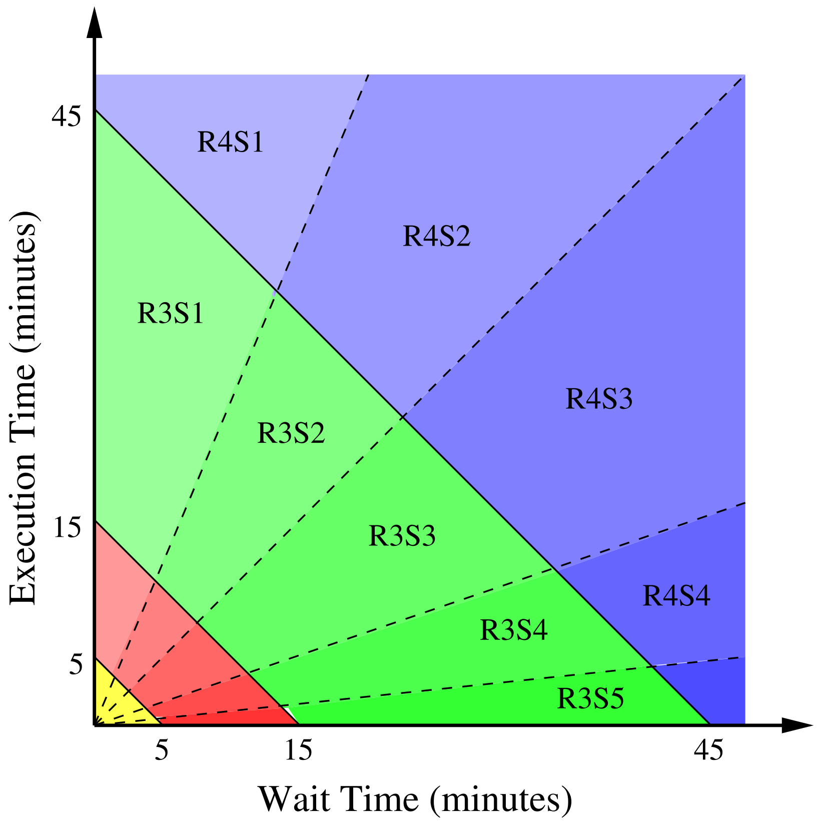


Figure 4.2: *Graphical illustration of the bins: The response bins are bounded between the solid lines, and the slowdown bins are bounded between the dashed lines.*

Bin	**S1**	**S2**	**S3**	**S4**	**S5**
R1	3.6	6.6	8.2	8.8	8.9
R2	9.5	14	18	19	18
R3	17	49	50	41	40
R4	119	149	159	195	119
R5	524	600	652	712	702

Table 4.4: *Medians of think times for sub-bins.*

times. In each column, we see that the median think time grows dramatically and monotonically with the response-time bin. Looking at rows does not reveal any such pattern for slowdown. However, jobs with the very lowest slowdowns do consistently tend to lead to lower think times than the other jobs with the same response time.

The conclusion of the above discussion is that the response-time metric is the one most important to the users since it has the strongest correlation with their subsequent behavior. The question that immediately follows is therefore *how* exactly these response times affect the behavior. We show in the next section that it affects users' decision to continue or abort their interactive session with the system.

4.4 Jobs Response Time and User Sessions

Sessions are periods of continuous activity by the users. This does *not* mean that their jobs must be continuously active throughout the session. A job may complete, and the user may think for a while before submitting the next job. If the think time is too long, on the other hand, it may well indicate that the user took a break. In this case, subsequent jobs will belong to a different session. The question is, therefore, how to distinguish between jobs that are separated by actual think times and belong to the same session, from jobs that belong to different sessions.

Zilber et al. answered this question by examining the distribution of think times in the traces [69]. Figure 4.3 reproduces part of their data (they used two additional traces in addition to our five), showing the CDF of the think times in the traces. We first observe that think times can be negative. This stems from the definition that think time is the time between the completion of a job and the submission of the next job, and indicates that sometimes users submit jobs without waiting for their previous jobs to complete. Such jobs are often submitted in *batches* — one after the other with very short gaps between the successive submissions, and without being affected by feedback from previous jobs [55]. Consequently, the negative think times are not useful for our study of feedback to the users, and we therefore ignore them in rest of this chapter.

Focusing on the positive think times, we see a steep climb in the CDF curve of all traces for think times of a few minutes, which levels off at about twenty minutes. This means that a large portion of the jobs are submitted within twenty minutes of the completion of a previous job, which is an indication for continuous activity periods by the users. Furthermore, beyond twenty minutes the think times are evenly distributed, without any features indicating a natural threshold. Zilber et al. therefore defined the sessions' think time boundary to be *twenty minutes*;

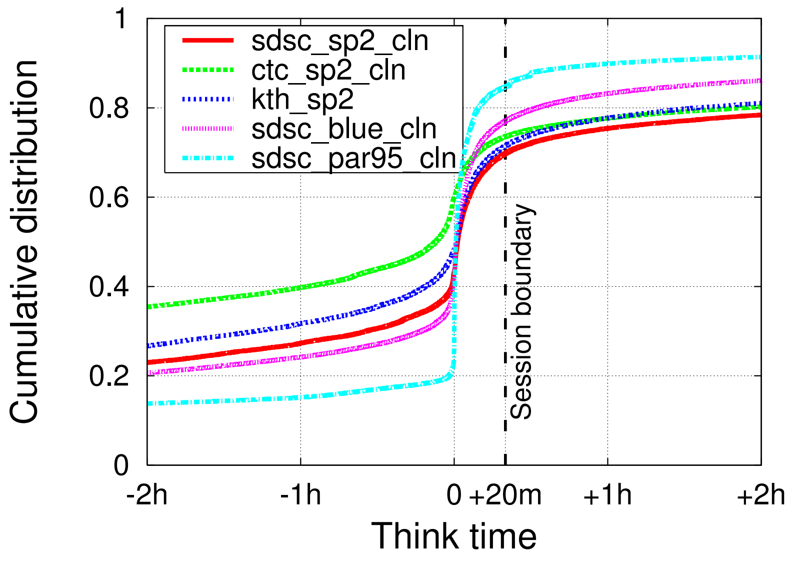


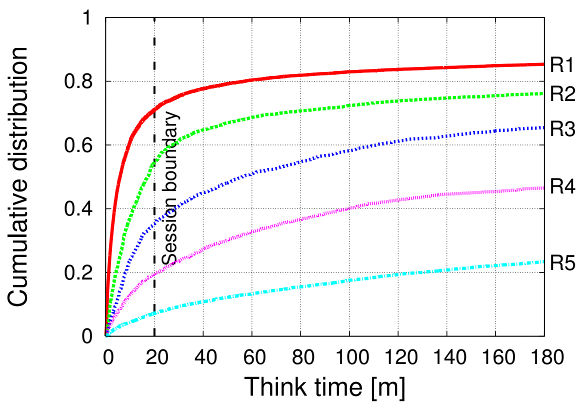
Figure 4.3: *CDF of think times for the five traces: Negative values indicate that one job started before the previous one completed. Session boundary is defined to be twenty minutes*

jobs submitted after a think time that is longer than twenty minutes are considered to start a new session. In our work we adopt this definition.

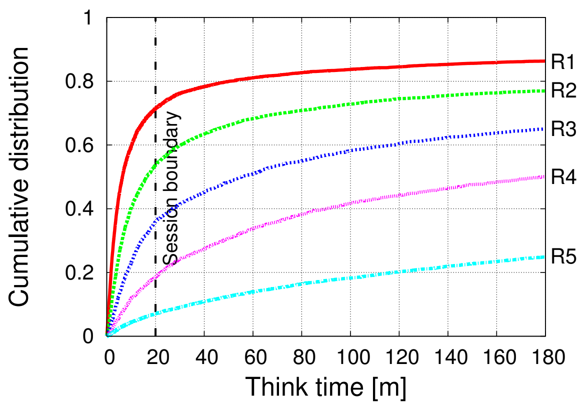
We remain focused on the response-time bins of the previous section, but this time we expand our analysis, and consider not only the median think times, but the entire distribution of think times following the jobs in each bin

The CDF of think times for the different bins is shown in Figure 4.4. The immediate impression is that the five sub-figures that represent the different traces are very similar. In all traces, there is a noticeable and a similar gap between the CDF curves of the different bins. Furthermore, for all traces, the curves follow the same vertical order: response bin R1 has the highest CDF, bin R2 has the second-highest CDF, etc.

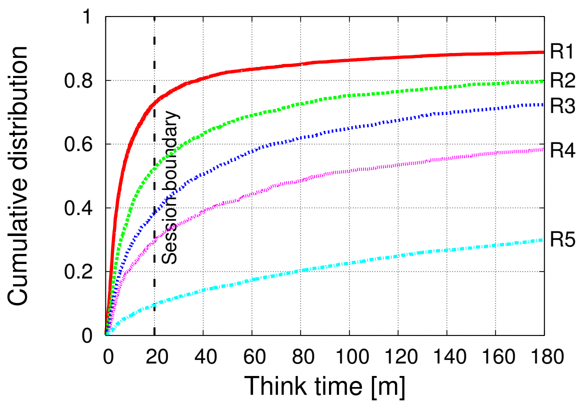
A closer examination of the figure also reveals that in all traces, all bins exhibit the same steep climb in the CDF up-to the session's twenty-minutes think time boundary, and beyond that point, all curves level off. In fact, the major difference between the curves is in the *percentage of the jobs that were submitted below the session boundary*, and it is this difference that determines the vertical order of the curves. For the SDSC-SP2 trace, for example, 72% of the subsequent jobs for response-bin R1 were submitted below the session boundary. For response bin R2, only 55% of the jobs were submitted below this boundary, and so on. The higher the bin number, the lower the percentage of jobs that were submitted below the session boundary.



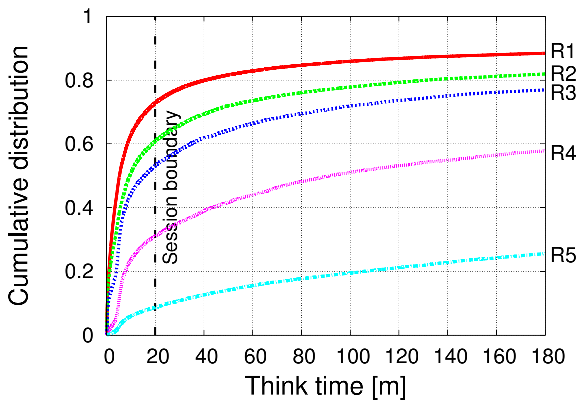
(a) SDSC-SP2



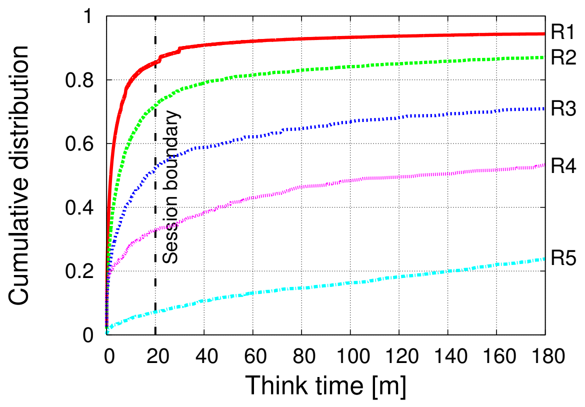
(b) CTC-SP2



(c) KTH-SP2



(d) SDSC-BLUE



(e) SDSC-PAR95

Figure 4.4: *CDF of think times for the five response-bins: The five sub-figures that represent a different trace each are very similar. In all, the higher the bin number, the lower the percentage of jobs that were submitted below the twenty-minutes session boundary.*

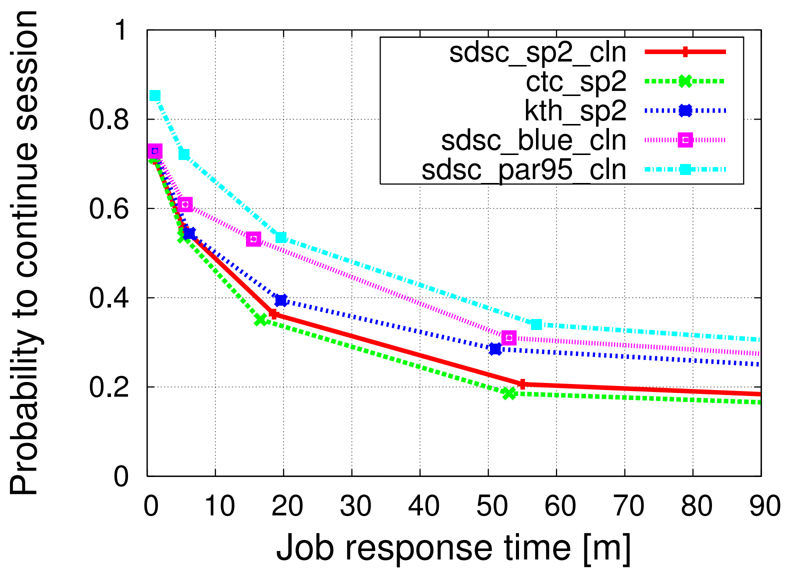


Figure 4.5: *Jobs' response time effect on users' behavior: The higher the response time of the jobs, the lower the probability that users will continue submitting jobs within the same session. The mapping is* non-linear *and is highly similar for all traces.*

Figure 4.5 summarizes these results for all five traces. For each bin we extracted the percentage of subsequent jobs that were submitted below the session boundary, using the CDF of think times of the bin. We also calculated the median of *response time* of the jobs in the bin, and used the median to represent the bin. We then plotted one against the other. The result is a *mapping* between the jobs response times, and the probability for the users to continue their sessions. We see that for all traces, the higher the response time of the jobs, the lower the probability for users to continue submitting jobs within the same session. The mapping itself is *non*-linear; the probability to continue the session initially drops rapidly as response time increases, and continues to drop more slowly for higher response times.

The conclusion is that the jobs' response times affect the users decision to continue or abort their interactive session with the system. In the next section we show that this decision may stem from expectations the users develop regarding the response time of their jobs.

4.5 User Performance Expectations

It is well known that user behavior is affected by expectations [58]. Live experiments conducted in the context of the web have found that users' tolerance to server delays is strongly affected by their expectations regarding the duration of the delay [5]. Accordingly, one would expect users of parallel systems to also develop expectations regarding the time frame by which their jobs should respond. The question is how these expectations affect their behavior, and what happens when response times lengthen beyond their expectations.

To answer this question we focus on a subset of our previous results. Specifically, we define two bins, so that one holds jobs that had a short wait, and the other holds jobs that had a short execution. Assuming that users expectations would be related to the *execution time* of their jobs, and not the wait times which are an artifact of certain conditions that existed in the system, these two bins actually represent two different scenarios: the one with the short waits represents the scenario where response times met their expectations, and the other represents the case where they expected a quick response, but it got lengthened because of long waits in the scheduler's queue. The threshold we chose for the bins is five-minutes of wait, and five-minutes of execution, respectively.

Due to the nature of the bins, the response times of the jobs they hold exhibit a large variance. The bin with the short waits for example, may hold two jobs that waited only a minute for execution, but the first executed for a few seconds, and the other for several hours. We therefore divided our bins into sub-bins, based on the response-time metric, and using the same ranges we used before as indicated in Table 4.2. This enabled us to examine the user behavior under the two scenarios, while also considering the effect of the response time of their jobs.

Figure 4.6 illustrates these bins graphically. Again, the horizontal axis represents the jobs' wait time, and the vertical axis represents their execution time. Our two bins are bounded between the two solid lines: the vertical bin holds the jobs with the short waits, and the horizontal bin holds the jobs with the short execution. The dashed diagonal lines represent the boundaries of the response bins, which intersect with our main bins and define the sub-bins. The sub-bins with the short waits carry the prefix 'W', and the ones with the short execution carry the prefix 'E'. All carry a suffix that identifies the response bin to which their jobs belong to, e.g., R1, R2, etc.

For each sub-bin we extracted the percentage of jobs that were submitted below the session twenty-minutes boundary, and plotted this percentage against the median of response time of the jobs in the sub-bin. The result is shown in Figure 4.7. The two curves represent the probability for the users to continue their session as a function of the response time of their jobs. One curve represents the scenario where response times met their expectations, and the second represents the case where they did not.

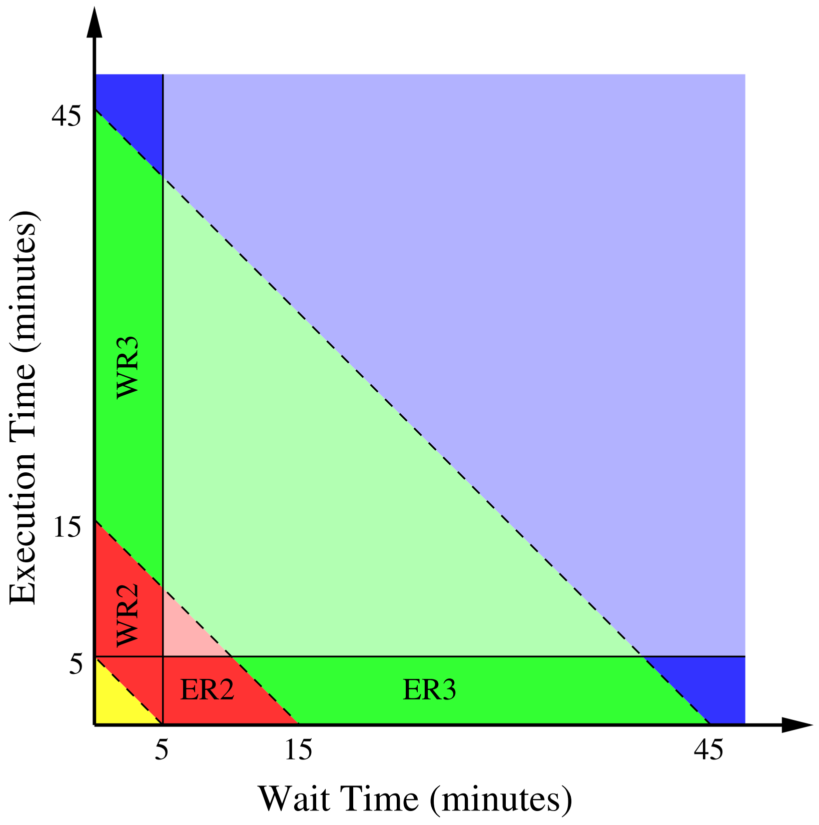


Figure 4.6: *Graphical illustration of the bins: The vertical bin holds the jobs with the short waits, and the horizontal bin holds the jobs with the short execution.*

In accordance with our previous results, we see that in both scenarios the probability for users to continue their session decreases as response time increases. What is surprising though is the high level of similarity between the curves, despite of the fact that they represent two essentially different scenarios.

Our proposed explanation to this difficulty is that the users' perception and motivation are indeed different in the two cases, but that their actual behavior just happens to be very similar. In the first scenario, response times meet users expectation. The fact that the probability to continue the sessions decreases as response times increase is then a straightforward result of the fact that users expect long response times, and therefore tend not to wait for their jobs. The longer they expect the response time to be, the higher the probability for them to discontinue their sessions.

In the second scenario execution times are short and users expect a quick response. They start to wait for their jobs, but when response times lengthens beyond their expectation because the jobs wait for a long time in the scheduler's queue, they tend to lose their patience and abort their

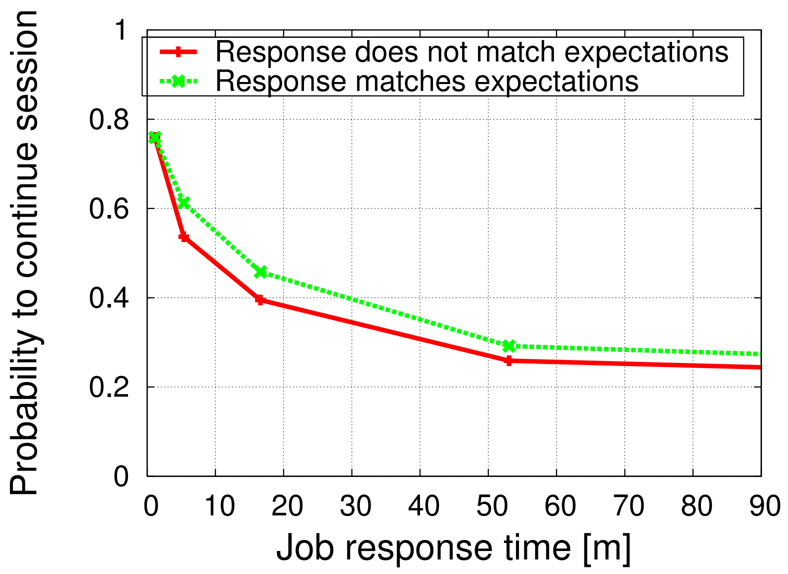


Figure 4.7: *Performance expectations and user behavior: The probability for users to continue their sessions decreases as the response time of their jobs increases, regardless of whether they expected that response or not.*

sessions. In this case the user behavior is indeed affected by the performance of the scheduler, but it happens in such a way that makes the end result appear similar to the behavior had they anticipated the long response time in advance.

We can also examine the bins with respect to the slowdown metric. In the bin with the short waits, slowdown *decreases* as response time increases. In the bin with the short execution times on the other hand, slowdown *increases* with the response time of the jobs. Still, the behavior of the users appears to be similar in both cases. This corroborates the results from Section 4.3, and indicates that response time is a much more reliable predictor of user behavior than slowdown.

4.6 Related Work

Early studies of the interaction between users and computers were conducted in the context of a single interactive system [7, 8, 58]. W. Tetzlaff described a system for recording state data on users of the IBM's vm/370 system, to help locate performance bottlenecks and their effect on the users [62]. Embley and Nagy examined theoretical models and experimental results related to the behavior of users during text editing tasks [18]. Klein et al. demonstrated how systems can be designed to help users recover from negative emotional states [31].

With the emergence of networked systems, studies began to appear which relate to the social behavior of users. Henderson and Bhatti analyzed session-level traces of multi-player networked games, and found that users decision to join a game is affected by the number of players already playing the game [25]. Balachandran et al. examined the distribution of users and network load in public-area wireless networks and found that there is not much correlation between the two [3].

Studying user reaction to system performance was often assumed to require live experiments with real users. G. N. Lambert for example, examined the effect of system response times on user productivity in development projects [34]. Similarly, Bouch et al. used live experiments to investigate the tolerance of users to web server delays [5], and Lee and Snavely examined user satisfaction live, at the San Diego Supercomputer Center [36].

Others however, have shown that it is possible to uncover the effect of a system on its users directly from the system traces. Tran et al. developed a model of web surfers reaction to network congestion simply by analyzing HTTP packet-traces [15]. Similarly, Chen et al. developed a model of Skype's users satisfaction purely from their VoIP traces [9]. In the context of parallel-systems however, we could not find any related reference.

In fact, when parallel-systems schedulers are evaluated, the workload used in the simulation is usually generated from traces, which means that the arrival rate of the jobs is already given, and is not affected by the performance of the system [60, 45, 59, 67, 35, 54]. The alternative is to use models to generate the workload, but most models either try to reproduce the arrivals already found in the traces [6, 29, 11, 42], or even use a Poisson model to further simplify the arrival process of the jobs [20, 16]. In all cases, there is no feedback in the workload between the performance of the system, user behavior, and the arrival of new jobs.

In the previous chapter we investigated the importance of this feedback and found it may lead to performance prediction errors of hundreds of percents [55]. We proposed a simulation methodology that uses user-models instead of traces to dynamically generate the workload, but our models were simplistic and reacted uniformly to differences in system performance. The findings presented in this chapter can help extend our models to support a more realistic behavior, in which users may abort their sessions as a result of poor system performance.

There is no single consensus regarding the performance metric that is most important to the users. More than 35 years ago, Brinch Hansen suggested to prioritize jobs using the slowdown metric [23], but since then many scheduling policies have been proposed and each evaluated differently. A few selected cases are listed in Table 4.5: all believe their metrics of choice

Year	Scheduling policy	Metrics
1999	Slack based backfilling [60]	Wait time
2001	EASY and conservative backfilling [45]	Response, Slowdown
2002	Selective reservation backfilling [59]	Slowdown, Turnaround
2002	Relaxed backfilling [67]	Total wait time
2002	Multiple queue backfilling [35]	Slowdown
2005	Lookahead based backfilling [54]	Response, Slowdown
2007	Probabilistic Backfilling [46]	Wait time

Table 4.5: *Scheduling policies and metrics used for their evaluation: All believe their metric is the one that is most important to the users, but none had ever investigated why.*

are the ones most important, but none had ever investigated why. Feitelson has examined the convergence of a few common metrics in the course of simulation, and found that different metrics converge differently depending on the workload and the scheduler being used [21]. User reaction to the metrics however, was not part of his study. Lee and Snavely presented utility functions as a mean to represent the value users attach to their jobs in a flexible manner [37].

Zilber et al. were the first to present a comprehensive study of parallel-systems traces based on users and sessions [69]. In Section 4.4, we adopted their definition of sessions as sets of jobs submitted within twenty minutes from the completion of a previous job.

4.7 Summary

A good scheduler should strive to promote the productivity of its users, but this requires an understanding of user behavior. Surprisingly, in virtually all performance evaluations, the effect of the scheduler on the users is ignored. The conventional simulations simply use traces to generate the workload, but in the trace the arrival rate of the jobs is already given, and is not affected by feedback from the scheduler. Furthermore, the metrics by which schedulers are compared vary from one evaluation to the other. Each analyst believes their metric is the one that is most important to users, but this is not justified.

In this study we investigated the effect of the performance of the system on the behavior of its users, and found that user behavior correlates with the response times of the jobs, not the slowdown as was previously sometimes assumed. We continued to investigate the actual type of the effect and found that response times affect the users' decision to continue of abort their sessions, and that the higher the response times of the jobs, the lower the probability for users to continue submitting jobs within the same session. Finally, we have shown that the decision to abort the session may stem from certain performance expectation that the users develop, regarding the time frame by which their jobs should respond.

We did *not* reach these findings using live experiments. Instead, all we did was to examine traces that contain raw data on jobs that were submitted to real, production-use parallel systems. We are not the first to examine these traces. In fact, some of the older traces were first analyzed more than ten years ago. We are though the first to take a different, slightly less obvious look at things. An important conclusion of this study is that a lot of interesting observations are still out there in the traces. All it takes is different angle and a fresh way of thinking to extract them.

In the next chapter we incorporate these findings into the simulation and introduce realistic user models whose behavior is affected by the performance of the system. We also present CREASY — a novel user-aware scheduler that exploits knowledge on user behavior to improve user satisfaction, demonstrate that it improves the productivity of its users by more than 50% compared to existing designs, and complete our discussion on the importance of the feedback in the workload.

Chapter 5

Feedback and Scheduler Designs

THE open-system, trace-driven simulations used to evaluate the performance of the schedulers, have led the schedulers to focus on the packing of jobs in the schedule, as a mean to improve a number of performance metrics that are only conjectured to be correlated with user satisfaction, with the premise that this will result in a higher productivity in reality. We argue that the lack of feedback in the workload in these simulations actually leads to sub-optimal scheduler designs, and to even dismissing potentially good design alternatives. We incorporate our findings regarding the behavior of users of parallel-systems into our site-level simulations, and introduce a much more realistic user behavior which is affected by the performance of 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.

5.1 Background

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 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 jobs get submitted during simulation solely according to the timestamps from the trace, irrespective of the system state, which further means 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. A scheduler capable of motivating its users to submit more jobs will not cause more jobs to be submitted, and vice-versa.

This inability to influence throughput is an inherent problem in open-system 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. 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 users of the system only implicitly through these metrics. They often try to optimize the packing of jobs in the schedule, since tighter packing usually leads to lower average simulated values. We are not aware of any scheduler that considers its users explicitly, nor of any attempt to investigate whether these seemingly "user-friendly" metrics indeed correlate with higher productivity.

We argue however, that the only way to truly maximize productivity is for the scheduler to consider the users of the system directly — to strive to keep them satisfied and motivate them to submit more jobs, and that the conventional, packing-based approach to scheduling leads to sub-optimal designs. We further argue the that the conventional performance metrics do not necessarily correlate with productivity, which means that it is even possible to dismiss potentially good design alternatives as poor under the conventional simulations.

To allow truly user-aware schedulers to be effectively designed, we enhance our site-level simulations from Chapter 3, which as already described, use *user-models* instead of traces to *dynamically* generate the workload for the evaluation. We incorporate our findings regarding the behavior of users of parallel-systems from Chapter 4 into the simulation, and introduce a much more realistic user behavior which is affected by the performance of the system. We also incorporate daily and weekly cycles into the simulated workload, to produce levels of activity similar to those found in reality.

These enhancements, and the fact that in site-level simulations, schedulers capable of motivating their users to submit more jobs *do actually* cause the throughput of the jobs to increase, enable the design of user-aware schedulers that strive to improve user satisfaction, since their effect on productivity can now be directly and reliably measured via the throughput metric.

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.

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 chapter is organized as follows. Section 5.2 describes common schedulers designs and how the conventional simulations led to these designs. Section 5.3 describes our enhanced site-level simulations, 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 5.4 presents our novel user-aware scheduler, CREASY. Section 5.5 describes the experiments we performed to demonstrate the importance of the feedback for the design of the schedulers. Section 5.6 surveys related work, and Section 5.7 summarizes the chapter.

5.2 Common Scheduler Designs

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.

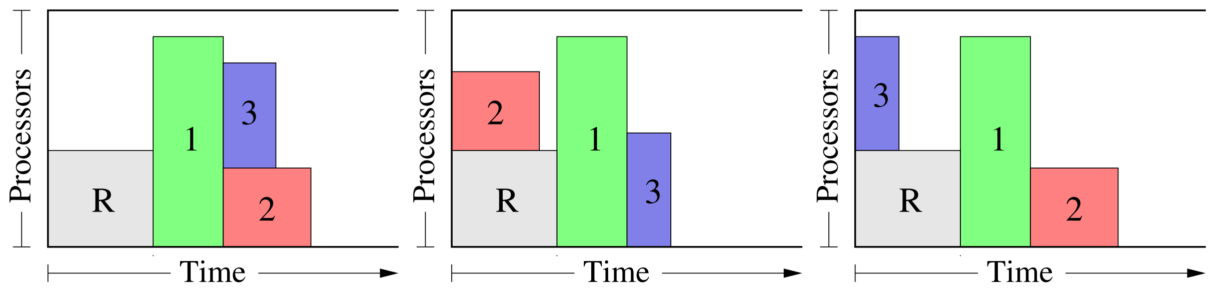
The role of the scheduler in such a system is to accept the jobs from the users, to allocate processors and to execute the jobs on the selected processors. For simplicity, we ignore issues like network contention, heterogeneous node configurations, and security.

The users of the system submit their jobs by providing to the scheduler a description of the jobs' resource requirements. 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.

The behavior of the schedulers upon job arrival differ greatly. Most schedulers maintain a queue where the jobs wait for processors to become available [39, 54]. 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 [35]. 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 [45], or in a place where it delays some of these jobs by a small factor [60].

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 [45]. 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 none 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



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

Figure 5.1: *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.*

job is still active at the system, and is excepting a fast response.

Figure 5.1 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 5.1(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 5.1(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 [39].

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. 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 5.1(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.

5.2.1 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 *without any feedback* between the completion of jobs and the arrival of new jobs. 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 de-accelerate 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 that considers its users explicitly, nor of any investigation as to whether these seemingly "user-friendly" metrics indeed correlate with higher productivity.

Category	Conventional Simulations	Site-Level Simulations	Site-Level Enhanced
Workload source	System traces	User models	
Workload generation	Open-system model	User-scheduler interaction	
Load scaling	Trace (de)-compression	Number of users	
Performance metrics	Response time, slowdown	Throughput, session length	
Number of sessions	Dictated by trace	Static	Dynamic
Cycles of activity	Dictated by trace	Not incorporated	Incorporated

Table 5.1: *Conventional, Site-level, and Enhanced site-level simulations.*

5.3 Enhanced Site-level Simulations

As described above, the conventional simulations lead to the design of schedulers that consider the system users only implicitly. To enable truly user-aware schedulers to be effectively designed, we enhance our site-level simulations from Chapter 3 with our findings regarding the behavior of users from Chapter 4. More specifically, instead of simulating only a static number of user sessions, we introduce user models that dynamically start and end their sessions with the system as a reaction to the performance they observe from their jobs. We also divide the user population into different classes to simulate daily and weekly cycles, similar to those found in real workloads.

These enhancements, and the fact that in site-level simulations, schedulers capable of motivating their users to submit more jobs *do actually* cause the throughput of the jobs to increase, enable the effective design of truly user-aware schedulers. These schedulers should strive to improve user satisfaction to reduce the likelihood for session aborts, which in turn translates into higher overall productivity that can be measured directly via the throughput metric. Table 5.1 summarizes the differences between the conventional, our original site-level simulations, and the above mentioned enhancements.

In the following section we briefly describe our findings regarding the behavior of users of parallel-systems, and focus only on those that directly pertain to our user models. More details on the methodology we develop to uncover the users' behavior patterns from traces of parallel-systems can be found in Chapter 4 and in [56]. 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

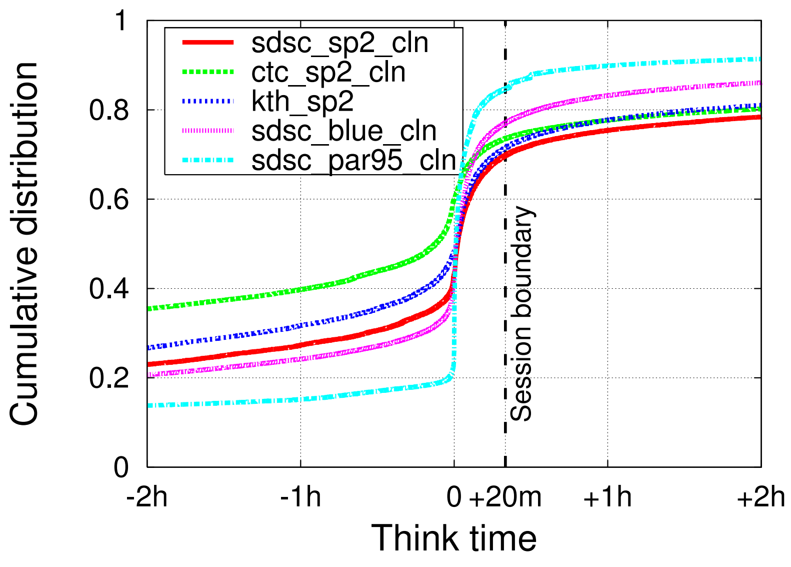


Figure 5.2: *CDF of think times in the five traces: negative values indicate that sometimes users submit jobs without waiting for their previous jobs to terminate. The steep climb in all curves which levels-off at about twenty minutes lead to defining the sessions think time threshold to be twenty-minutes.*

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 5.3.2. 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 5.4. The simulation results of our scheduler reported in section 5.5 were also obtained using Site-Sim.

5.3.1 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

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

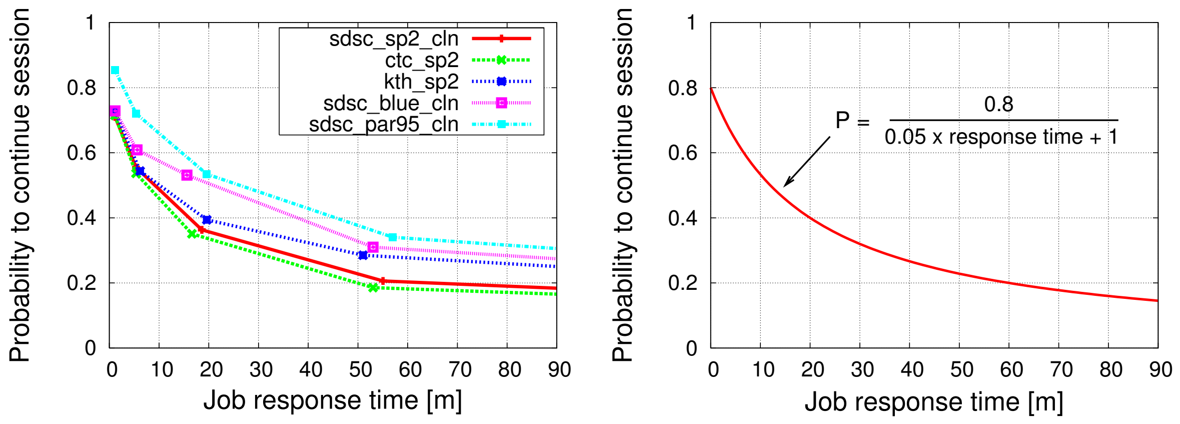
Table 5.2: *The five traces we used for our analysis: together, they represent many years of activity by hundreds of users.*

different traces of parallel systems [69]. Figure 5.2 shows the CDF of the think times in five of these traces, which are listed in Table 5.2, and are available on-line from the Parallel Workloads Archive [12]. 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.

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 [36].

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



(a) Original trace data (b) Data combined and curve smoothened

Figure 5.3: *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.*

and submit more jobs. The mapping is illustrated in Figure 5.3(a) and it forms the basis for session dynamics model described in Section 5.3.2.1.

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 the feedback on the design of the schedulers, our simple models suffice.

5.3.2 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 5.3.2.4 we provide examples as to how these models interact during simulation.

5.3.2.1 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.

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 5.3(a) and smoothened the resulting curve, as shown in Figure 5.3(b). We found that the curve can be roughly described by Equation 5.1. Next, during simulation whenever job j terminates, we calculate the response time of the job, and use Equation 5.1 to determine the probability $p\_cont(j)$ that the user who submitted the job will continue his session with the system.

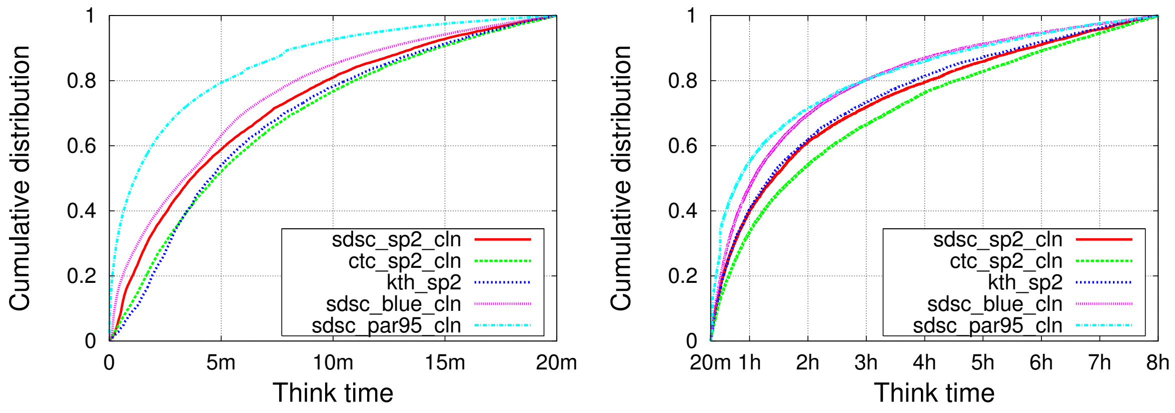
$$p\_cont(j) = \frac{0.8}{0.05 \times resp\_time(j) + 1} \tag{5.1}$$

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 5.2.

$$decision = \begin{cases} \text{continue session} & \text{with probability } p\_cont(j) \\ \text{abort session} & \text{with probability } 1 - p\_cont(j) \end{cases} \tag{5.2}$$

Once we know whether the 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 5.2. 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



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

Figure 5.4: *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.*

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 5.4. For the simulations, we combined the data from all five traces into a single representative distribution.

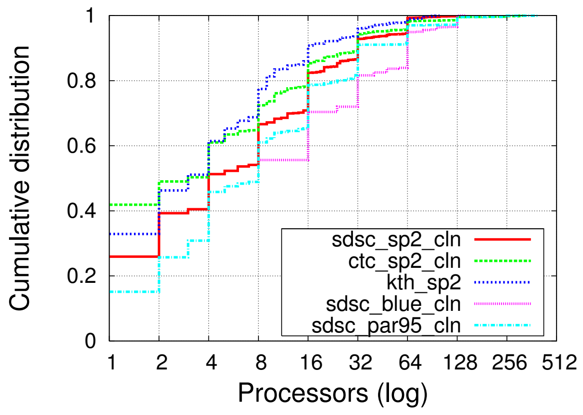
5.3.2.2 Job Submission Model

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.

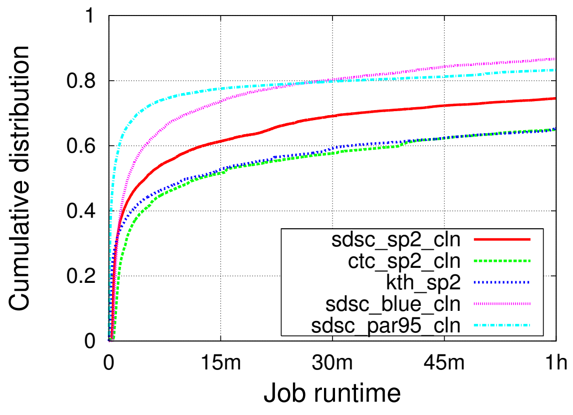
To generate the attributes, we once again used distributions that are based on empirical data from the traces. The CDFs of the job sizes and runtimes are shown in Figure 5.5. The first is a modal distribution with most jobs using power-of-two processors, and the second is a rather skewed distribution 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 [20].

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



(a) CDF of jobs' sizes



(b) CDF of jobs' runtimes

Figure 5.5: *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.*

for the jobs[1], and the bottom level repeating them to generate effects of locality [13]. 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 5.6(a). Again, we combined all traces into a single representative distribution for use in the simulation.

To actually submit the jobs, we closely examine Figure 5.2 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 5.7. 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

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

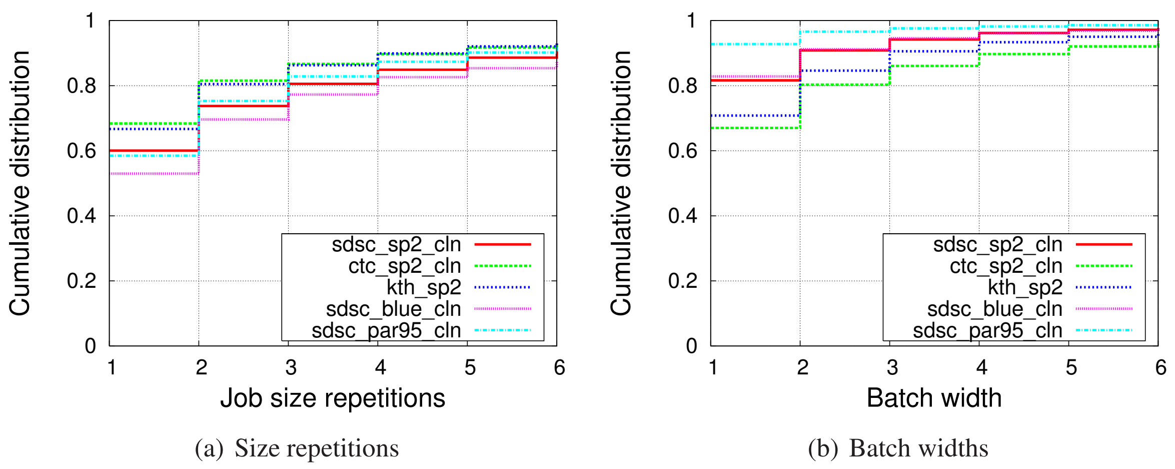


Figure 5.6: *CDF of job size repetitions and batch widths in the traces: for the simulations, we combined the data from all five traces into a single representative distribution.*

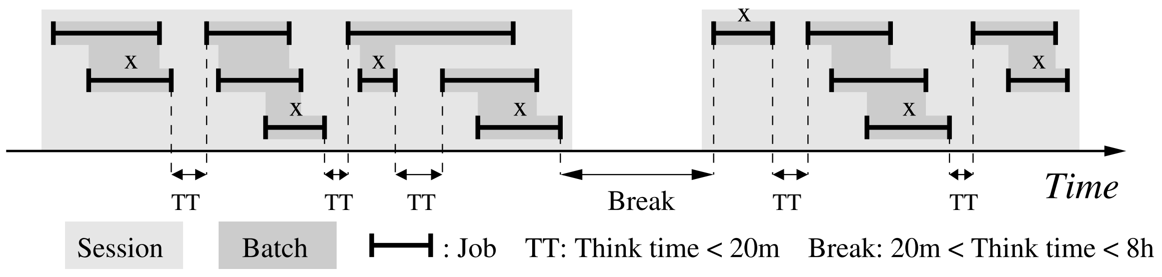


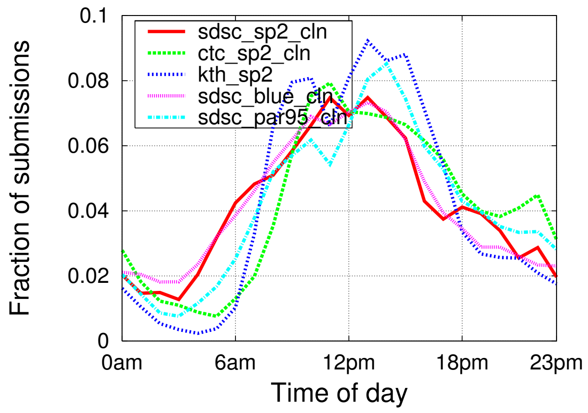
Figure 5.7: *Sessions, batches, and think times: the jobs marked with an* X *are those used by the session dynamics model to derive the think time until the next job.*

batches is shown in Figure 5.6(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.

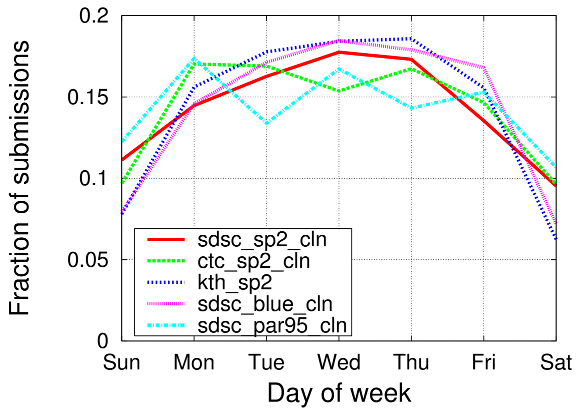
5.3.2.3 Activity Cycles Model

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 week-ends. 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 [41].

Figure 5.8(a) shows the distribution of job submissions during the 24-hours *daily* cycle in the



(a) Daily Cycles



(b) Weekly Cycles

Figure 5.8: *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.*

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 5.8(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", 7:30am - 17:30pm daytime window. This means that the user will submit his first job at 7:50am and be suspended at 17:50pm.

5.3.2.4 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 5.9. Our user arrives to work at 7:30am sharp, and the job 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 minutes 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 submits a job at 17:10pm that responds five minutes later. The time is verified, and the dynamics model decides

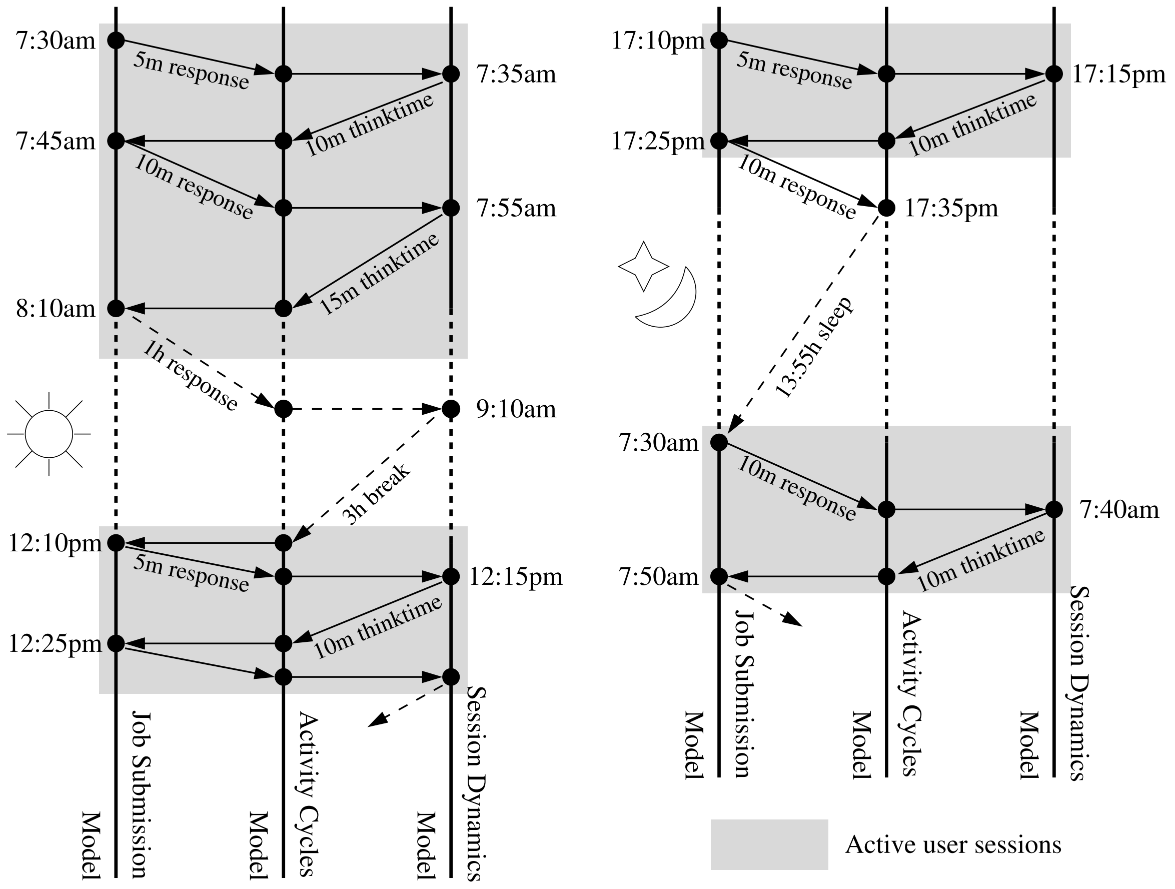


Figure 5.9: *Two examples for the models interaction during simulation: active user sessions are shown in the dark gray.*

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.

5.4 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. Our scheduler is describe below, and its simulation results are presented in Section 5.5.

5.4.1 Criticality of Jobs

Our scheduler is similar to the EASY scheduler from Section 5.2 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 [47, 48], 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 5.3 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 chances for a session abort. We therefore defined the criticality of jobs using Equation 5.3, which is the *derivative* of Equation 5.1 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.

$$criticality(j) = \frac{0.04}{(0.05 \times estimated\_response\_time(j) + 1)^2} \tag{5.3}$$

Note that in the denominator of Equation 5.3 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 5.3 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 5.3, 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 5.4: the criticality part on the left is taken directly from Equation 5.3, 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.

$$priority(j) = \alpha \times criticality(j) + seniority(j) \tag{5.4}$$

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 5.3 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.

5.5 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 5.4 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.

5.5.1 User Productivity

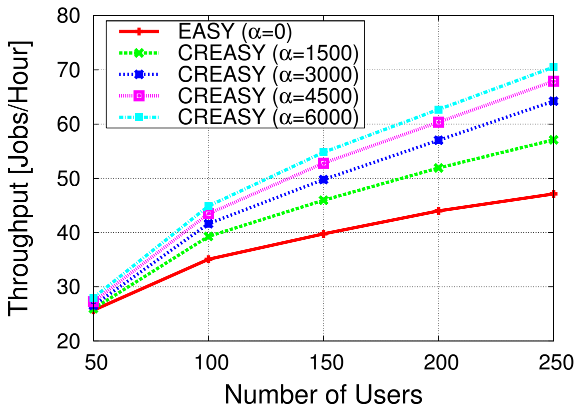
Improving productivity is an important goal for any parallel system scheduler, and the best indicator for user productivity is the throughput — the number of jobs the users submit to the system over a period of time.

Figure 5.10(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.

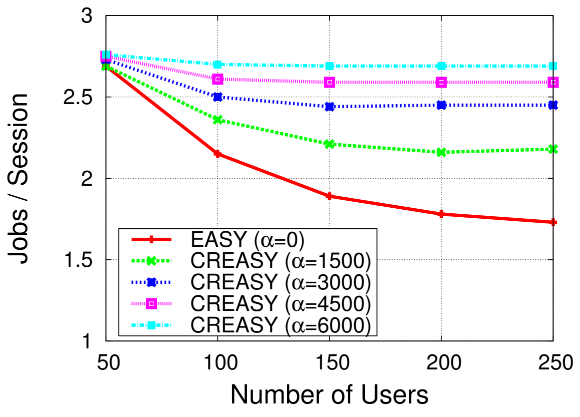
For the largest site we simulated, of 250 users, the throughput under the EASY scheduler is 47 jobs/hour, while under CREASY with $\alpha = 6000$ it is 71 jobs/hour which is an exceptional improvement of more than 50%. Improvement is milder but is still very significant for CREASY with lower α values: 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. 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 5.10(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 $\alpha = 6000$.



(a) Average job throughput



(b) Average session length

Figure 5.10: *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).*

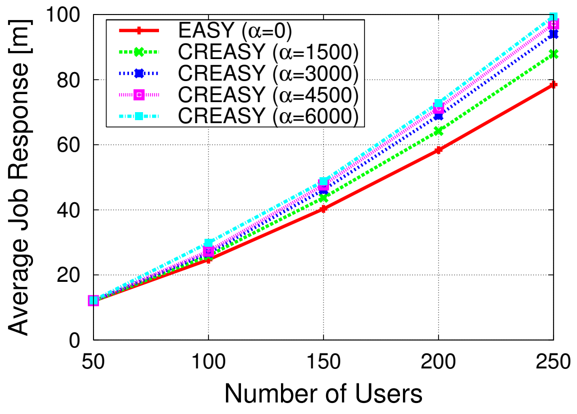
The reason session length drops is rooted in the core design of the schedulers. The original EASY scheduler does *not* consider the critically of the jobs. As a consequence, when the load begins to increase more and more users under EASY abort their sessions as a result of their jobs being delayed by the scheduler. This causes average session length to decrease, and explains the poor throughput of the scheduler in Figure 5.10(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.

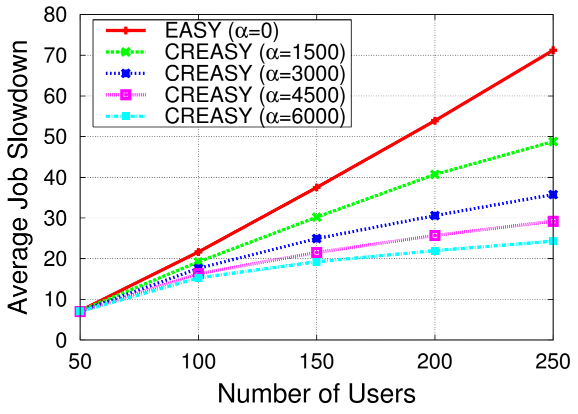
Finally, when $\alpha = 6000$, the drop in session length is hardly noticeable even under the highest loads. This means that CREASY was capable of virtually isolating the interactive users from the load conditions that exist in the system, successfully providing them with a highly responsive environment even under the most extreme load conditions.

5.5.2 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 5.11 shows the performance of our five schedulers according to two of the most commonly used metrics: the average job response time and the slowdown.



(a) Average response time



(b) Average slowdown

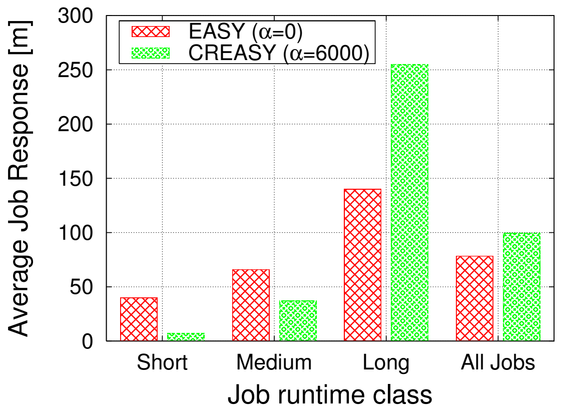
Figure 5.11: *Inconsistency according to the conventional metrics: the schedulers with the* lower *α values outperform the ones with the higher values according to the response-time metric (a), but according to the slowdown, the ones with the* higher *values have significantly better performance (b).*

Similar to Figure 5.10, we see that the differences in performance between the schedulers become significant only when the load begins to increase. In contrast from Figure 5.10 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 5.11(a). The ones with the *higher* values still have significantly better performance according to the slowdown.

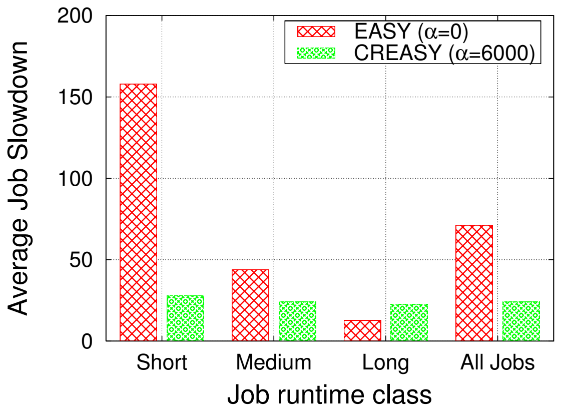
Under the highest 250 users load for example, the average job response time under the EASY scheduler is 78 minutes, while under CREASY with $\alpha = 6000$ it is 99 minutes, which is a 27% degradation in performance for CREASY. On the other hand, the average job slowdown under EASY is 71, while under 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 5.10(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



(a) Per-class average job response time



(b) Per-class average job slowdown

Figure 5.12: *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.*

than 10 minutes. We chose these boundaries based on the distribution of runtimes from Figure 5.5(b), in order to create classes of approximately the same size.

Figure 5.12 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 5.12(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.

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 5.12(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 5.13 summarizes the change 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.

5.6 Related work

The basic batch scheduling algorithm is First-Come-First-Serve (FCFS), in which jobs are considered in their order of arrival [50]. If there are enough free processors to run the first job, it is started immediately, but if there are not enough processors available, the job is delayed, and all subsequent jobs are delayed as a result, so as not to violate the FCFS order.

The poor system utilization of FCFS led researchers to explore alternatives to improve performance. Inspired by Shortest-Job-First (SJF)[2], Shortest-Processing-Time-First (SPT) scheduling tries to produce optimal average response times for the jobs [32, 2]. The opposite algorithm, Largest-Processing-Time-First (LPT) executes the longest jobs first so as to minimize makespan, at the expense of longer average response [53]. Similarly, Smallest-Job-First (SJF) and

[2]CREASY differs from SJF in that it considers the response time rather than the runtime of the jobs.

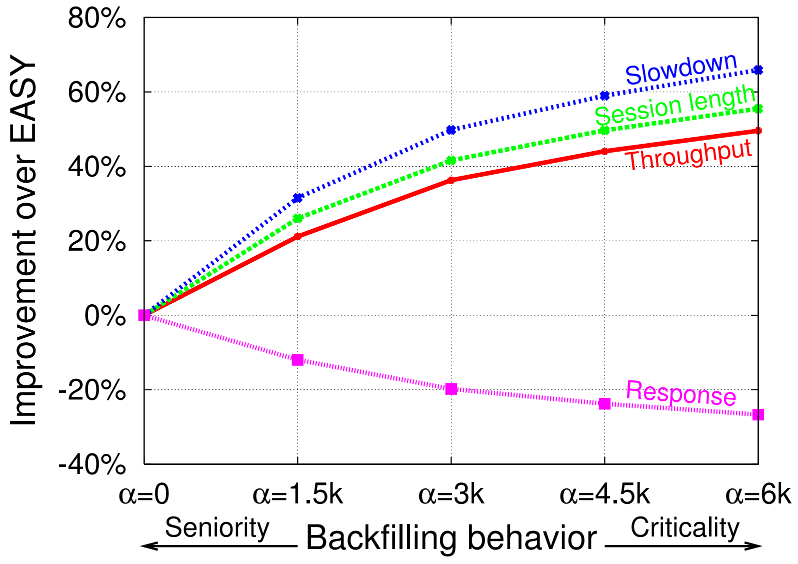


Figure 5.13: *Improvements in performance for CREASY relative to the original EASY scheduler: performance improves with higher α values according to the throughput, session-length, and the slowdown metrics, but degrades according the response-time metric.*

Largest-Job-First (LJF) execute the jobs with the smallest and largest size first, respectively [44, 17]. Finally, the Smallest-Cumulative-Demand-First (SCD), executes the jobs in the order of the product of their size and runtime [52]. All the above algorithms however, may expose jobs to starvation under some circumstances.

Backfilling is an optimization that tries to balance between the goals of performance and execution guarantees. Backfilling in general allows small jobs from the *back* of the queue to bypass larger jobs that arrived earlier, to *fill* holes in the schedule. Backfilling algorithms differ in the way they reserve processors for the older jobs, and in the way they prioritize the jobs in the queue, which may affect their order of execution.

Backfilling was first implemented in a production system in the context of EASY, the Extensible Argonne Scheduling sYstem for the IBM SP1 system [39]. This version was based on aggressive backfilling in which only the first job in the queue gets a reservation, but this was shown to be enough to guarantee the execution of all jobs. The obvious alternative is to use conservative backfilling in which every job gets a reservation, and which also produces a more predictable schedule. Mu'alem and Feitelson compared the two approaches and pointed that their relative performance may actually depend on the workload [45].

The MAUI scheduler also used in production includes a tunable parameter that allows the administrator to set the number of reservations [28]. Chiang et al. suggested that making 2–4 reservations is a good compromise [10]. Alternatively, one can set the number of reservations dynamically. Srinivasan et al. suggested a selective approach in which a job gets a reservation only if its slowdown exceeds a certain threshold [59]. This is essentially equivalent to Talby and Feitelson's "flexible backfilling", in which every job gets a reservation, but backfilling may violate these reservations up to a certain slack that is determined dynamically [60]. Ward et al. suggested a relaxed backfilling strategy that also uses slacks, but whose width is static [67].

The original EASY scheduler, as well as many other schedulers, prioritize the jobs according to their arrival order [39]. Flexible backfilling, however, combines administrative, user, and scheduler priorities in setting the slacks for the jobs [60]. Keleher et al. evaluated the effectiveness of SJF prioritization with randomness [30]. Chiang et al. proposed prioritization that is based on the resource consumption of the jobs [10]. Lawson and Smirni demonstrated the effectiveness of having multiple queues for different jobs classes, based on the jobs' projected execution times [35]. Shmueli and Feitleson demonstrate the use of lookahead for optimizing the packing of the jobs in the schedule [54], and Talby and Feitelson showed how to combine different scheduling policies in an adaptive manner [61].

In all the above examples, however, the schedulers were evaluated using trace-driven, open-system simulations, without any feedback in the workload. In Chapter 3 we investigated the importance of this feedback and demonstrated it may lead to performance prediction errors of hundreds of percents [55]. We proposed a simulation methodology that uses user-models instead of traces to dynamically generate the workload, but our early models were simplistic and reacted uniformly to differences in system performance.

In this chapter we extended our models to support a more realistic behavior, in which users may abort their sessions as a result of poor system performance. Our new models are based on our findings from Chapter 4, that users are affected by the response times of their jobs, and that long response may cause users to abort their sessions with the system [56]. This was the basis for the session-dynamics model presented in Section 5.3.2.1, that depicts the users' reaction to their jobs. We also enhanced our models with awareness for activity cycles; Lo and Mache demonstrated the benefits of having prime and non-prime time queues for batch scheduling and defined rules of thumb for each queue [41]. Feitelson and Nitzberg analyzed the distribution of jobs during the day, nights, and weekends on the NASA Ames iPSC/860 system [19].

The use of feedback in system design was demonstrated in several contexts. Ganger and Patt demonstrated how modeling an entire systems can help improve I/O subsystems performance [22]. Similarly, Hsu and Smith added feedback to I/O traces in order to study various I/O optimization techniques [27]. Scott and Sohi demonstrated how feedback schemes in networked systems can be used to avoid congestion and improve overall system performance [51]. Feedback control theory was also shown to be highly effective. Diao et al. used multiple-input, multiple-output control theory to dynamically tune various system parameters in the Apache Web server, to enforce policies of interrelated metrics [14], and Lee et al. used feedback control in shared storage to assure agreed-upon response time to applications, and to maximize aggregate throughput [38].

5.7 Summary

For more than two decades parallel-systems schedulers are being evaluated using simulations that suffer from severe limitations. In particular, the lack of feedback in the workload in these simulations led to the design of schedulers that focus solely on the packing of jobs instead of on the users of the system directly, to try and improve an alternative set of performance metrics that are only conjectured to be correlated with user satisfaction.

Through a combination of a novel simulation methodology that incorporates feedback in the workload, and a user-aware scheduler that considers the criticality of the jobs to the users, we have demonstrated that the conventional, packing-based approach to scheduling is far from optimal, and highlighted the potential in user-aware designs. We have further shown that the conventional performance metrics do not necessarily correlate with productivity, which means that it is even possible to dismiss potentially good design alternative as poor, under the conventional simulations.

The user models presented in this chapter differ from those of chapter 3. In particular, the session dynamics model that handles the dynamic starting and ending of user sessions as a reaction to the performance of their jobs, and the activity cycles model that incorporates daily and weekly cycles into the simulation, result in a much more realistic user behavior, which we exploited in CREASY to demonstrate the importance of the feedback for the design of the schedulers.

Our journey to explore the importance of the feedback ends here, but there are still many research directions left to investigate. In the next chapter we conclude the entire research, and suggest future research directions which we feel are both interesting and challenging.

Chapter 6

Discussion and Conclusions

The conventional simulations presently used to evaluate the performance of parallel-systems schedulers are trace driven. To generate the workload from the trace they use an open-system model that simply plays-back the trace according to its timestamps, and there is *no feedback* in the workload between the arrival of new jobs, the load in the system, and the ability of the simulated scheduler to handle the load.

This lack of feedback manifests itself in several ways, affecting both the *evaluation* of the schedulers and their *design*. It affects the evaluation since the generated workloads no longer reliably represent real workloads, which causes the performance predicted by the simulation to be inaccurate. It effects the design because the throughput metric which is the best indicator for user productivity cannot be used in open-system evaluations. This leads the schedulers to focus on the packing of jobs instead of on the users of the systems directly, to try and optimize an alternative set of metrics which are only conjectured to correlate with user satisfaction.

In this research we presented an alternative simulation methodology named *site-level simulation*, that uses user-models to *dynamically* generate the workload for the evaluation, instead of using traces which pre-determine the workload even before the simulation begins. These models, whose behavior in simulation is similar to the behavior of real users, interact with the system and introduce feedback, which not only improves the representativeness of the workload, but also allows user-aware schedulers to be reliably evaluated and hence effectively designed.

To experiment with site-level simulations we developed *Site-Sim* — a site-level simulator that integrates users and schedulers under a single simulation framework. We also developed *CREASY* — a novel user-aware scheduler that exploits knowledge on user behavior and prioritizes jobs that are critical for the users to improve user satisfaction. We then carried out a series of carefully designed experiments to demonstrate the importance of the feedback for the evaluation and the design of the schedulers.

Our first set of experiments focused on the representativeness of the workload. Using the traces produced by Site-Sim we demonstrated that the conventional simulations may under or overestimate the performance of the schedulers by hundreds of percents. We also showed how conventional load scaling further ruins the representativeness the workload by violating precedence relations that naturally exist between jobs in reality.

In the second set of experiments we compared the performance of CREASY to the original, well-known packing-based EASY scheduler, and demonstrated that user productivity improves by tens of percents under the user-aware design. We also compared the two schedulers using the conventional performance metrics and showed that these metrics do not necessarily correlate with productivity, which means that it is even possible to dismiss potentially good design alternatives under the conventional simulations.

Though we feel we have clearly demonstrated the advantage in using our simulation methodology, we also acknowledge the fact that as opposed to traces which are easy to collect and straightforward to use, relying on user models to generate the workload will always be open for interpretation. We understand the need to keep the simulations simple, and empathize with the researchers' reluctance to conduct time and resource consuming experiments to understand user psychology.

Recognizing it is of critical importance to our simulations, we devoted approximately one third of our research effort to the study of user behavior. We developed a novel analysis methodology through which we demonstrated that it is possible to uncover the users' behavior patterns directly from parallel-systems traces, without conducting live experiments with real users. We believe that this is one of the major contribution of our work, and a necessary step toward the general acceptance of our methodology among its critics.

Nevertheless, we have only scratched the surface of the virtually endless domain of human behavior study. Real users for example are known to be influenced by contextual factors such as the type of task they perform, their experience, and the cumulative time they interact with the system. They are also sensitive to fairness in the system, and might consider fairness to be more important than productivity. Understanding these factors will help improve our models accuracy, and doing so using only the traces is one of the major challenges we leave behind.

Another factor to consider is the jobs the users submit. Daytime jobs are known to be different from the jobs submitted during the night; interactive jobs are usually lighter and much more critical to the users compared to the heavy batch jobs that execute over nights and weekends. In our simulations we intentionally chose not to make this distinction and to incorporate the minimal level of details we felt is necessary for discussing feedback. However, with a relatively small amount of effort it is possible to isolate different job classes in the traces and to model these classes for a better use in the simulations.

The final step, once the models get improved and the workload refined, is to revise CREASY to consider the aggregate effect of all these factors on the users, to re-evaluate its performance and demonstrate that it can still significantly improve user productivity under the new, much more complex but realistic conditions. This task alone is extremely challenging, but it is a necessary step toward the actual deployment of the first truly user-aware parallel-system scheduler.

Appendix A

Site-Sim Interfaces

Site-Sim is a framework written in C++ that we developed specifically for running site-level simulations. It defines two types of entities, *users* and *schedulers*. The users generate the workload for the simulation by submitting jobs to the scheduler, and the scheduler in turn schedules the jobs and notifies the users when they complete.

Site-Sim does not explicitly define how the users behave, or how the scheduler schedules the jobs. Instead, it exploits class inheritance in C++ only to define the interfaces through which the different entities communicate. The exact behavior of the scheduler upon job arrival, or the users' upon job completion, is left to the implementor of the interfaces.

Modeling the scheduler actions is relatively straightforward and requires the implementation of an internal job queue and an allocation algorithm to process the queue. Modeling the users on the other hand is much more involved and we therefore implemented the user interfaces in two phases: first for Chapter 3 without supporting user arrivals or departures, and then for Chapter 5 with a dynamic arrival and departure behavior which is affected by the system's performance and the time-of-day.

In the remainder of this appendix we briefly describe only the primary user and scheduler interfaces. It is important to note that Site-Sim's architecture is far more complex. It uses an internal event queue to guarantee the correct timing and delivery of events to the users and the scheduler. This mechanism which constitutes the heart of the framework is totally hidden from the implementors of the interfaces, which are only exposed to the high level details of the architecture that are required for implementing the models.

A.1 User Primary Interfaces

- `void User::completeJob(Job j)` and `void User::wakeUp()` — Site-Sim uses these interfaces to notify the users that their jobs have completed, and to wake up sleeping users, respectively.
- `long User::getTime()` — Users use this interface to query the time-of-day in order to decide whether to continue and submit more jobs or to go to sleep. The return value is specified in *seconds* from the beginning of the simulation. It is the responsibility of the interface implementor to translate this value into meaningful time-of-day units.
- `void User::sleep4(long time)` — This sends the user to sleep for `time` seconds. It is the responsibility of the interface implementor to translate time-of-day units into seconds for sending the users to long night sleeps when supporting daily cycles.
- `void User::submitJob(Job j)` — The user submits his jobs to the scheduler using this interface. Prior to submission, the job attributes in the job object must be correctly initialized, e.g., with the number of required processors and runtime. Site-Sim uses standard-error to report illegal values.
- `int User::queueLength()` — This allows the user to query the number of jobs currently present in the scheduler's queue. Advanced user models can use this number to estimate the load in the system in order to decide on whether to continue and submit more jobs or to take breaks.
- `int User::getSeed()` — At initialization Site-Sim assigns a unique seed value to each user. The `getSeed()` interface allows the user to query his seed and call the POSIX `void srand(unsigned int seed)` function to set this seed prior to using random number generators. This is useful when the user samples distributions to generate job attributes and wishes to reproduce the exact same sequence of numbers across simulations and regardless of how other entities in the simulation use the random number generator.

A.2 Scheduler Primary Interfaces

- `void Scheduler::arriveJob(Job j)` and `void Scheduler::termJob(Job j)` — Site-Sim uses these interfaces to notify the scheduler on job arrivals and terminations, respectively. Typical schedulers implement an internal job queue to which they add new jobs upon arrival, and notify the users that their jobs have completed upon termination.

- `void Scheduler::Allocate()` — Since several jobs may arrive to the scheduler at the same time it is useful to first add these jobs to the queue and only then run the allocation algorithm on the populated queue. Site-Sim therefore separates job arrival events from allocation events. It uses the `Allocate()` interface to notify the scheduler that no more jobs will arrive at the present time unit, to allow the scheduler to execute its allocation algorithm in an optimal manner.

- `void Scheduler::notifyUser(Job j, long time)` — Schedulers use this interface to notify the users on job completions. The user ID is part of the job object and is transparently set by Site-Sim when the user submits his job. The `time` field can be used by the scheduler to delay the notification event to a later time.

- `int Scheduler::getMachineSize()` and `int Scheduler::getUsedSize()` — The scheduler uses these two interfaces to query the total number of processors in the system, and the number of processors currently used, respectively. The system size can be set through command-line parameters at simulation start; the default size is 128 processors.

- `void Scheduler::downProc(int p)` and `void Scheduler::upProc(int p)` — Site-Sim uses these interfaces to notify the scheduler that a certain processor in the system has been shut-down or reactivated, respectively. Such events are needed when incorporating failure models into the simulation to simulate a truly realistic system behavior. In our simulations we assumed a 100% reliable hardware and did not use these interfaces.

Bibliography

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

[2] O. Arndt, B. Freisleben, T. Kielmann, and F. Thilo. A comparative study of online scheduling algorithms for networks of workstations. *Cluster Computing*, 3(2):95–112, 2000.

[3] A. Balachandran, G. M. Voelker, P. Bahl, and P. V. Rangan. Characterizing user behavior and network performance in a public wireless lan. *SIGMETRICS Perform. Eval. Rev.*, 30(1):195–205, 2002.

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

[5] 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*, pages 297–304, New York, NY, USA, 2000. ACM Press.

[6] M. Calzarossa and G. Serazzi. A characterization of the variation in time of workload arrival patterns. *IEEE Trans. Comput.*, C-34(2):156–162, Feb 1985.

[7] S. K. Card, T. P. Moran, and A. Newell. The keystroke-level model for user performance time with interactive systems. *Commun. ACM*, 23(7):396–410, 1980.

[8] S. K. Card, A. Newell, and T. P. Moran. *The Psychology of Human-Computer Interaction*. Lawrence Erlbaum Associates, Inc., Mahwah, NJ, USA, 1983.

[9] 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, pages 399–410, New York, NY, USA, 2006. ACM Press.

[10] 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*, number 8, pages 103–127. Springer-Verlag, Jul 2002. Lect. Notes Comput. Sci. vol. 2537.

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

[12] D. G. Feitelson. Parallel workload archive. http://www.cs.huji.ac.il/labs/parallel/workload/.

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

[14] Y. Diao, N. Gandhi, J. L. Hellerstein, S. Parekh, and D. M. Tilbury. Using mimo feedback control to enforce policies for interrelated metrics with application to the apache web server. In *Proceedings of the IEEE/IFIP Network Operations and Management Symposium*, pages 219–234, 2002.

[15] 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.

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

[17] J. E. G. Coffman, M. R. Garey, and D. S. Johnson. Approximation algorithms for bin packing: a survey. pages 46–93, 1997.

[18] D. W. Embley and G. Nagy. Behavioral aspects of text editors. *ACM Comput. Surv.*, 13(1):33–70, 1981.

[19] Feitelson and Nitzberg. Job characteristics of a production parallel scientific workload on the NASA ames iPSC/860. In D. G. Feitelson and L. Rudolph, editors, *Job Scheduling Strategies for Parallel Processing – IPPS'95 Workshop*, volume 949, pages 337–360. Springer, 1995.

[20] D. G. Feitelson. Packing schemes for gang scheduling. In D. G. Feitelson and L. Rudolph, editors, *Job Scheduling Strategies for Parallel Processing*, pages 89–110. Springer-Verlag, 1996. Lect. Notes Comput. Sci. vol. 1162.

[21] D. G. Feitelson. Metrics for parallel job scheduling and their convergence. *Job Scheduling Strategies for Parallel Processing*, 2221:188–206, 2001.

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

[23] P. B. Hansen. An analysis of response ratio scheduling. In *IFIP Congress (1)*, pages 479–484, 1971.

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

[25] T. Henderson and S. Bhatti. Modelling user behaviour in networked games. In *MULTIMEDIA '01: Proceedings of the ninth ACM international conference on Multimedia*, pages 212–220, New York, NY, USA, 2001. ACM.

[26] 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*, page 218, Washington, DC, USA, 1999. IEEE Computer Society.

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

[28] D. B. Jackson, Q. Snell, and M. J. Clement. Core algorithms of the Maui scheduler. In *JSSPP '01: Revised Papers from the 7th International Workshop on Job Scheduling Strategies for Parallel Processing*, pages 87–102, London, UK, 2001. Springer-Verlag.

[29] J. Jann, P. Pattnaik, H. Franke, F. Wang, J. Skovira, and J. Riodan. Modeling of workload in MPPs. In D. G. Feitelson and L. Rudolph, editors, *Job Scheduling Strategies for Parallel Processing*, pages 95–116. Springer Verlag, 1997. Lect. Notes Comput. Sci. vol. 1291.

[30] P. J. Keleher, D. Zotkin, and D. Perkovic. Attacking the bottlenecks of backfilling schedulers. *Cluster Computing*, 3(4):245–254, 2000.

[31] J. Klein, Y. Moon, and R. W. Picard. This computer responds to user frustration. In *CHI '99: CHI '99 extended abstracts on Human factors in computing systems*, pages 242–243, New York, NY, USA, 1999. ACM.

[32] S. Krakowiak. *Principles of operating systems*. MIT Press, Cambridge, MA, USA, 1988.

[33] D. Krishnamurthy, J. A. Rolia, and S. Majumdar. A synthetic workload generation technique for stress testing session-based systems. *IEEE Transactions on Software Engineering*, 32(11):868–882, 2006.

[34] G. N. Lambert. A comparative study of system response time on program developer productivity. *IBM Systems Journal*, 23(1):36–43, 1984.

[35] B. G. Lawson and E. Smirni. Multiple-queue backfilling scheduling with priorities and reservations for parallel systems. *SIGMETRICS Perform. Eval. Rev.*, 29(4):40–47, 2002.

[36] 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.*, 20(4):495–506, 2006.

[37] C. B. Lee and A. E. Snavely. Precise and realistic utility functions for user-centric performance analysis of schedulers. In *HPDC '07: Proceedings of the 16th international symposium on High performance distributed computing*, pages 107–116, New York, NY, USA, 2007. ACM Press.

[38] H. D. Lee, Y. J. Nam, and C. Park. Regulating I/O performance of shared storage with a control theoretical approach. In *NASA/IEEE Conference on Mass Storage Systems and Technologies (MSST)*, April 2004.

[39] D. Lifka. The ANL/IBM SP scheduling system. In D. G. Feitelson and L. Rudolph, editors, *Job Scheduling Strategies for Parallel Processing*, pages 295–303. Springer-Verlag, 1995. Lect. Notes Comput. Sci. vol. 949.

[40] Z. Liu, N. Niclausse, and C. Jalpa-Villanueva. Traffic model and performance evaluation of web servers. *Perform. Eval.*, 46(2-3):77–100, 2001.

[41] V. Lo and J. Mache. Job scheduling for prime time vs. non-prime time. In *Intl. Conf. Cluster Comput.*, number 4, pages 488–493, Sep 2002.

[42] 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.

[43] 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.

[44] S. Majumdar, D. L. Eager, and R. B. Bunt. Scheduling in multiprogrammed parallel systems. *SIGMETRICS Perform. Eval. Rev.*, 16(1):104–113, 1988.

[45] 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.*, 12(6):529–543, 2001.

[46] A. Nissimov and D. G. Feitelson. Probabilistic backfilling. In E. Frachtenberg and U. Schwiegelshohn, editors, *Job Scheduling Strategies for Parallel Processing*, pages 102–115. Springer Verlag, 2007. Lect. Notes Comput. Sci. vol. 4942.

[47] D. Raz, H. Levy, and B. Avi-Itzhak. A resource-allocation queueing fairness measure. In *SIGMETRICS '04/Performance '04: Proceedings of the joint international conference on Measurement and modeling of computer systems*, pages 130–141, New York, NY, USA, 2004. ACM Press.

[48] G. Sabin and P. Sadayappan. Unfairness metrics for space-sharing parallel job schedulers. In D. G. Feitelson, E. Frachtenberg, L. Rudolph, and U. Schwiegelshohn, editors, *Job Scheduling Strategies for Parallel Processing*, pages 238–256. Springer Verlag, 2005. Lect. Notes Comput. Sci. vol. 3834.

[49] B. Schroeder, A. Wierman, and M. Harchol-Balter. Open versus closed: a cautionary tale. In *NSDI'06: Proceedings of the 3rd conference on 3rd Symposium on Networked Systems Design & Implementation*, pages 18–18, Berkeley, CA, USA, 2006. USENIX Association.

[50] U. Schwiegelshohn and R. Yahyapour. Analysis of first-come-first-serve parallel job scheduling. In *SODA '98: Proceedings of the ninth annual ACM-SIAM symposium on*

Discrete algorithms, pages 629–638, Philadelphia, PA, USA, 1998. Society for Industrial and Applied Mathematics.

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

[52] K. C. Sevcik. Application scheduling and processor allocation in multiprogrammed parallel processing systems. *Perform. Eval.*, 19(2-3):107–140, 1994.

[53] J. Sgall. line scheduling – a survey. Lecture Notes in Computer Science. Springer-Verlag, Berlin, 1997.

[54] E. Shmueli and D. G. Feitelson. Backfilling with lookahead to optimize the packing of parallel jobs. *J. Parallel Distrib. Comput.*, 65(9):1090–1107, 2005.

[55] 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*, pages 167–178, Washington, DC, USA, 2006. IEEE Computer Society.

[56] 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)*, pages 274–280, Oct 2007.

[57] E. Shmueli and D. G. Feitelson. On simulation and design of parallel-systems schedulers: Are we doing the right thing? *IEEE Transactions on Parallel & Distributed systems.*, To appear.

[58] B. Shneiderman. *Designing the user interface: strategies for effective human-computer interaction*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1986.

[59] 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*, pages 55–71, London, UK, 2002. Springer-Verlag.

[60] 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, page 513, Washington, DC, USA, 1999. IEEE Computer Society.

[61] D. Talby and D. G. Feitelson. Improving and stabilizing parallel computer performance using adaptive backfilling. In *IPDPS '05: Proceedings of the 19th IEEE International Parallel and Distributed Processing Symposium (IPDPS'05) - Papers*, page 84.1, Washington, DC, USA, 2005. IEEE Computer Society.

[62] W. H. Tetzlaff. State sampling of interactive VM/370 users. *IBM Systems Journal*, 18(1):164–180, 1979.

[63] D. Tsafrir, Y. Etsion, and D. G. Feitelson. Modeling user runtime estimates. In *Workshop on Job Scheduling Strategies for Parallel Processing (JSSPP)*, pages 1–35. Cambridge, Massachusetts, June 2005. Lecture Notes in Computer Science, volume 3834.

[64] D. Tsafrir, Y. Etsion, and D. G. Feitelson. Backfilling using system-generated predictions rather than user runtime estimates. *IEEE Trans. Parallel Distrib. Syst.*, 18(6):789–803, June 2007.

[65] D. Tsafrir and D. G. Feitelson. The dynamics of backfilling: solving the mystery of why increased inaccuracy may help. In *IEEE International Symposium on Workload Characterization (IISWC)*, pages 131–141, San Jose, California, October 2006.

[66] D. Tsafrir and D. G. Feitelson. Instability in parallel job scheduling simulation: the role of workload flurries. In *IEEE International Parallel and Distributed Processing Symposium (IPDPS)*, page 10, Rhodes Island, Greece, April 2006.

[67] 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*, pages 88–102, London, UK, 2002. Springer-Verlag.

[68] A. Wong, L. Oliker, W. Kramer, T. Kaltz, and D. Bailey. System utilization benchmark on the Cray T3E and IBM SP2. In D. G. Feitelson and L. Rudolph, editors, *Job Scheduling Strategies for Parallel Processing*, pages 56–67. Springer Verlag, 2000. Lect. Notes Comput. Sci. vol. 1911.

[69] 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.

[70] D. Zotkin and P. J. Keleher. Job-length estimation and performance in backfilling schedulers. In *HPDC '99: Proceedings of the 8th IEEE International Symposium on High Performance Distributed Computing*, page 39, Washington, DC, USA, 1999. IEEE Computer Society.