# Preserving User Behavior Characteristics in Trace-Based Simulation of Parallel Job Scheduling

Netanel Zakay Dror G. Feitelson
School of Computer Science and Engineering
The Hebrew University of Jerusalem, 91904 Jerusalem, Israel
netanel.zakay@mail.huji.ac.il, feit@cs.huji.ac.il

***Abstract*—Evaluating the performance of a computer system requires the use of representative workloads. Therefore it is customary to use recorded job traces in simulations to evaluate the performance of proposed parallel job schedulers. We argue that this practice retains unimportant attributes of the workload, at the expense of other more important attributes. Specifically, using traces in open-system simulations retains the exact timestamps at which jobs are submitted. But in a real system these times depend on how users react to the performance of previous jobs, and it is more important to preserve the logical structure of dependencies between jobs than the specific timestamps. Using dependency information extracted from traces, we show how a simulation can preserve these dependencies. To do so we also extract user behavior, in terms of sessions and think times between the termination of one batch of jobs and the submission of a subsequent batch.**

## I. Introduction

When a new scheduler design is suggested, it is impractical to experiment with it in production use. Instead it is first evaluated in simulation, and only if it demonstrates significant improvements in performance can it become a candidate for an actual deployment. Reliable simulations are therefore critical for the choices made in reality.

The simulations commonly used to evaluate schedulers are trace driven, and use an open-system model to play back the trace and generate the workload for the evaluation. This means that new requests get issued during simulation solely according to the timestamps from the trace, irrespective of the logic behind the behavior of the users and of the system state. Therefore, the workload may not be representative of the behavior of real users. Moreover, the throughput of the system being evaluated is also dictated by the timestamps, instead of being affected by the actual performance of the scheduler.

The base assumption of such simulations is that if we use recorded traces, the workload will be representative and therefore the performance metrics will be reliable. However, they don't take into account that traces contain a signature of the scheduler that was used on the traced system [18]. In other words, the users' actions are not a universally true workload, but rather reflect their reactions to the scheduler's decisions. This means that real users would react differently to the decisions of the new scheduler. Therefore, when we want to evaluate a new scheduling policy, and to use a representative workload, the simulation should reflect user reactions to the evaluated scheduler rather than to the original scheduler. It is more important to preserve the logic of the users' behavior than to repeat the exact timestamps.

For example, assume that a user sends a job A and then another job B which depends on the results of A. During a simulation the first job may be handled at a different time, which may lead to incorrect logic of the workload:

1) Case 1: A is scheduled later than originally. However, in the simulation B will arrive according its original timestamp. The result is a possibly smaller difference between A and B. Moreover, job B may even be scheduled before job A.
2) Case 2: A is scheduled earlier during the simulation. We would expect the user to send B earlier too. But instead job B will again arrive to the system solely according to its timestamp.

This means that when we simulate a different scheduler, the workload no longer represents the behavior of real users. This may lead to unrepresentative simulations and unreliable evaluations. For example, a consequence of blindly using the timestamps is that when the system is saturated, it keeps on receiving jobs according to the timestamps, causing unrealistic load conditions. In reality, we would expect the users to sense the load and the slow responses, and to send fewer new jobs. The opposite case is also a serious problem — when the system has available resources the simulation can't exploit them by submitting more jobs.

This leads us to the second main drawback of conventional simulations. As long as the system is not saturated, the throughput during the simulation is dictated by the timestamps, instead of being affected by the actual performance of the scheduler. However, the throughput is probably the best indicator for user productivity, and testifies to the scheduler's capacity for keeping its users satisfied and motivating them to submit more jobs. The common solution is to use metrics like the response time or slowdown, that, on one hand, can be affected by the scheduler, and, on the other hand, are conjectured to correlate with user satisfaction. However, it is not clear that they correlate with the throughput.

Instead, we propose a novel feedback-based simulation. This is a trace driven simulation, but using a semi-closed system model to play back the trace and generate the workload for the evaluation. The feedback reproduces the fine-grained interactions that naturally exist between the users and the system in reality. In particular, the simulation retains the logical structure of the workload — the users' behavior, as reflected by the think times, sessions, and dependencies between jobs. Moreover, schedulers that are capable of motivating their users to submit more jobs will actually cause the users to send their jobs faster, and therefore lead to higher throughput. This implies that schedulers will be evaluated with more realistic workloads and that they can be designed to improve user satisfaction directly, since their effect on productivity will be reliably evaluated.

To achieve this we suggest to divide each user's work into sequence of dependent batches. Then we model all the possible dependencies between batches. During the simulation we preserve these dependencies. This is done by simulating the submittal of a batch only when all its dependencies are satisfied. This creates the feedback affect described above, while preserving the characteristics of the workload, for example the order of the jobs per user and the job properties.

In order to use this feedback mechanism we need to model how a user would react to different performance levels, and in particular, when users will submit their jobs. For example, if a job is delayed in the system, the user model decides whether the user has the original think-time before the next job, or maybe he takes a break of a few hours for lunch or even a few days due to a weekend. Given the limited information contained in traces, there is no way to verify that a user model is correct. As a first step we therefore present alternative models of user behavior, which reflect expected behaviors and preserves different properties of the traces. We then compare the generated workloads with the original, and check the users' characteristics.

## II. Related Work

Traditionally, parallel system schedulers have been evaluated using simulations driven by traces or synthetic workloads based on statistical workload models (e.g. [9], [13], [4], [7], [10], [25]). In either case, the simulations typically follow an open systems model: the system receives jobs from some external population of users and processes these jobs. The job arrivals are independent of the system performance and state.

There have also been a small number of previous works, in different domains, who noted that a closed system model with feedback may be more realistic [5], [18], [17], [15], [11]. According to Spink, "feedback involves a closed loop of causal influences" [21]. In the context of systems performance evaluation, the idea is that poor performance may discourage users and cause them to submit fewer additional jobs; at the very least, they will delay the submittal of additional jobs until previous ones have terminated. Conversely, good performance may cause them to submit additional jobs at a more rapid pace. In either case the changes in user behavior affect subsequent system performance. In particular, reduced submittals when performance is poor contribute to system stability [18]. Sentiments such as these have been echoed in studies of networking and storage systems [6], [8].

In order to include feedback in evaluations one needs a model of how users react to load. While direct experimental evidence is rare [2], some works have considered user tolerance of delays and bandwidth limitations [16], [1], [14], [22], [23], [19]. And using such a model, Shmueli suggested a scheduler design that is specifically targeted to interact with user behavior [20]. The work closest to ours is that of Shmueli and Feitelson [18], [19], and we consider their user model in Section VII-B. The unique contributions of our work include a detailed analysis of job dependency structures, and new suggestions of how user behavior models may be extracted from workload traces. These lead to an improved match with traced workloads.

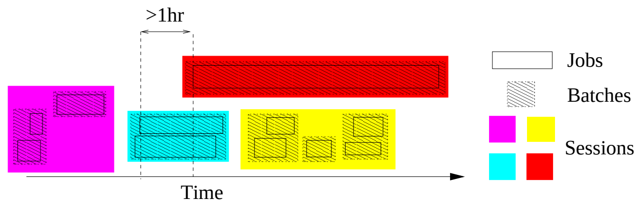

Fig. 1. Illustration of batches and sessions.

## III. Background

There are different types of parallel systems. Without loss in generality, we focus on the 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 or preemption support, so processors are allocated to jobs using a one-to-one mapping: one processor for every process of the job for the duration of the job's execution. 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 and heterogeneous node configurations.

The system's users submit their jobs by providing job descriptions to the scheduler. For our type of system, this typically includes two important attributes: the number of processors the job requires in order to execute, which is often referred to as the job's size, and an estimated upper bound on the runtime of the job, to enable the scheduler to plan ahead. In the evaluations we use jobs data from traces available in the Parallel Workloads Archive (www.cs.huji.ac.il/labs/parallel/workload).

## IV. Sessions and Batches

In order to create a feedback effect, we need to understand the structure of each user's work, and in particular the user's sessions. Intuitively, a session is a period of continuous work by a user. This does not mean that the user was active 100% of the session's time. A user may run a job to completion, think about the result, and then run another job, all within the same session.

The above description seems to imply sequential work, where jobs in a session never overlap. Empirical evidence from traces shows that this is clearly not always the case. Following Shmueli and our previous work [20], [24], we call a set of such overlapping jobs a *batch*, and treat them as a unit. Thus a session may contain several batches in sequence, and each batch may contain a number of jobs. The interval between batches is called the think-time (TT).

Finding the batches and sessions of the users is a basic requirement in order to understand and analyze their dynamics. However, activity logs do not contain explicit information about sessions. Thus our first goal is to estimate the batches and sessions based on data such as job submit and end times. We do not use a job's start running time because it reflects scheduling activity and not user activity. We define sessions and batches as follows (see [24] for justifications):

**Definition 1.** *A session is a maximal sequence of jobs of the same user such that the inter-arrival time between two successive jobs is up to one hour*

**Definition 2.** *For two jobs $j_1$ and $j_2$, such that $j_1.arrival \leq j2.arrival$, we say that $j_1$ and $j_2$ overlap if $j_1.end > j2.arrival$.*

**Definition 3.** *Consider a graph with jobs as nodes and edges connecting overlapping jobs provided they are in the same session. A batch is a connected component of jobs in this graph.*

The algorithm to derive sessions and batches according to these definitions is quite intuitive and simple. To produce sessions we scan all the jobs of a user according to their arrival time. If the inter-arrival time of the current job is longer than an hour, this represents a session break, and therefore this job starts a new session. Once the jobs are partitioned into sessions, we partition each such session into batches. We scan all the jobs in a session according to their arrival time and keep track of the latest end time seen so far. If the current job arrived after this time, the job starts a new batch.

An illustration of sessions and batches is presented in Figure 1. There are several interesting observations relating to overlapping sessions and batches:

**Definition 4.** *For two batches $b_1$ and $b_2$, we say that $b_1$ and $b_2$ overlap if there are two overlapping jobs $j_1$ and $j_2$, such that $j_1$ is in $b_1$ and $j_2$ is in $b_2$.*

**Observation 5.** *Two batches in the same session can't overlap. Hence all the jobs in one of them terminate before the first arrival of the second.*

The proof is simple. If two batches overlap, they have at least one pair of overlapping jobs, and therefore they will be in the same connected component of overlapping jobs in the session. By the batch definition, these batches should then actually be one batch.

**Observation 6.** *Two batches in different sessions may overlap.*

In Figure 1 we can see an example of this. The second session contains a batch with two long jobs. After sending these jobs, the user leaves the system and comes back after several hours. As a result, the next job he submit starts a different session and therefore a different batch, despite the fact that the previous jobs from the previous session are still running on the system.

**Observation 7.** *All the batches of a given user are in fact well ordered in a single sequence by their arrival times.*

By the definition of sessions, a session is a sequence of successive jobs, and the inter-arrival time between different

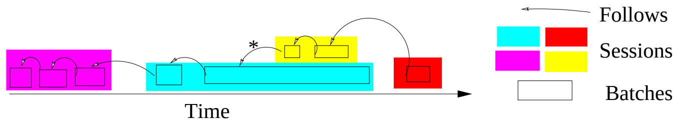

Fig. 2. Illustration of the follows relation.

sessions is at least one hour. Therefore, the sessions can be sorted according to the arrival time of their jobs, and this is a well defined order. Batches in turn are constructed by scanning the jobs in a session according to their arrival time, and associating each job to the current batch or starting a new batch. This implies that batches contain a sequence of successive jobs. Therefore, sorting the batches in a session according to the arrival time of the jobs is a well defined order. Combining these two results, we can conclude that all the batches of a user can be sorted according to the arrival times.

Based on this we can index each user's batches according to their place in this order. This means that for each two batches of a user $b_i$ and $b_j$, $i < j$ iff the jobs in $b_i$ were submitted before those in $b_j$. Using this order we can define a follows relation between batches:

**Definition 8.** *For each two batches $b_i$ and $b_j$ of the same user, we say that $b_j$ follows $b_i$ if $j = i + 1$.*

An illustration of the follows relation appears in Figure 2, where the batches are the nodes, and a directed edge from $b_j$ to $b_i$ means that $b_j$ follows $b_i$. Pay attention that the order is defined according to the arrival times, and therefore the edge are from the arrival of a batch to the last arrival of the previous batch.

## V. Shortcomings of Conventional Simulations

To understand the shortcomings of conventional simulations, we use the example presented in Figure 3. The top plot represents the trace. The Cyan user sent a job that couldn't run immediately, so it was delayed until enough processors became available. Meanwhile, the Red user sent a job. This job needed less processors and was scheduled immediately. Therefore it finished quickly. The user was still there and sent an additional job that depended on the results of the previous one. We can see the scheduler's signature on the workload: the fact that the scheduler uses backfilling caused the Red user to send the next jobs quickly.

In the middle graph we show a conventional simulation of a system with a different scheduler — FCFS without backfilling. All the jobs arrive according to their timestamps. But due to the scheduler policy, Red's jobs wait in the queue until Cyan's job starts to run. Then, they all start to run together, because they already arrived and there are enough processors available.

In contrast, we suggest that it is important to retain the dependency between Red's jobs. Each of Red's jobs depends on the previous job. Therefore the second job shouldn't arrive to the system until after the first one ends. The exact arrival time of the jobs is a difficult question we discuss in the next section. The simplest approach is to preserve the same think-times between the jobs. This leads to a simulation as presented in the third graph.

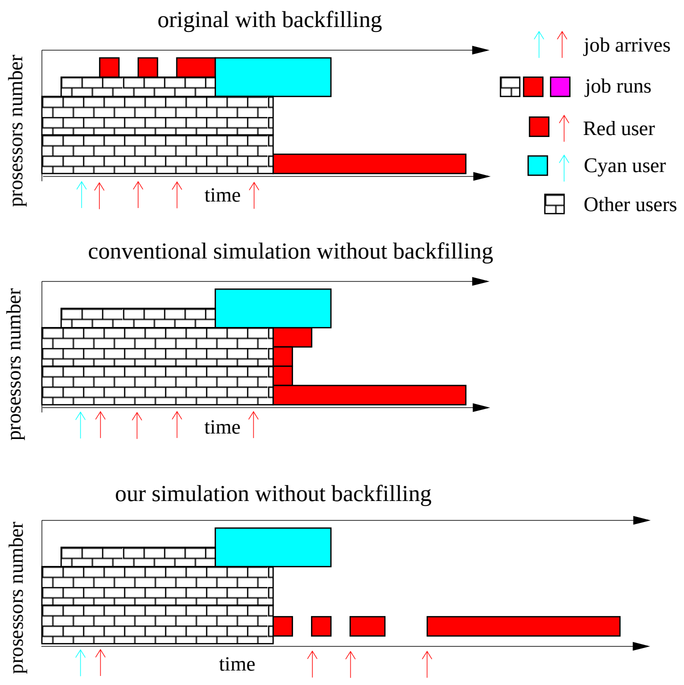

Fig. 3. Top: example schedule on a system that uses backfilling. This workload is then used to simulate a FCFS scheduler in a conventional simulation (middle) and a feedback based simulation (bottom).

This example demonstrates many problems of conventional simulations. Such simulations use the recorded trace without any change in order to preserve the properties of the workload. But in fact they preserve only a subset of properties, and sacrifice others. In particular, sticking to the original arrival times destroys the dependency structure and the think-times. *We argue that it is more important to preserve the logical structure of the workload, as embodied by dependencies and think times, and adjust the arrival times accordingly*. This is done by a feedback model that leads to changes to the arrival times of new jobs according to the terminations of previous jobs.

Second, using the workload trace as is may produce unrealistic performance measurements. In the conventional simulation, most of Red's jobs suffer from poor performances. This is because they arrived according to the original timestamps, without taking into account the state of the system. The jobs keep arriving despite the fact that the previous jobs haven't been finished yet. This causes the FCFS scheduler to have extremely poor performance. However, in reality users won't use a system with a problematic scheduler as they use a system with better scheduler. Rather, the users will slow down according to the performances and the system state. This is exactly what happens in the feedback simulation: the first of Red's jobs will indeed have poor performance, but the next job will arrive after this job finished, and therefore won't suffer from poor performance too. FCFS is still worse than EASY, but the difference is more realistic.

Third, we can see here that conventional simulations can't effect the throughput — despite using a worse scheduler in the simulation, all the users start at the same time and finish at the same time, which depends mostly on the timestamps.

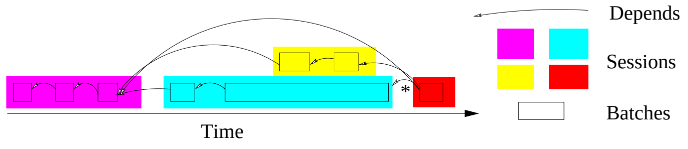

Fig. 4. Illustration of the depends on relation.

However, in our simulation, Red sensed the slow response (for his first job), and as a result, sent the rest of the jobs later. This caused him to finished later and his throughput to be lower.

It is worth to also consider the opposite situation — a trace created by a FCFS-based scheduler being used to simulate a scheduler that uses backfilling. If we assume the same user behavior, we would expect the recorded workload to be similar to the third graph. If we run a backfilling simulation using this workload, the first job would be scheduled immediately upon its arrival, and hence will finish much sooner. But if this is a conventional simulation, the rest of the jobs will arrive at the same times, and be scheduled at the same times as they were originally. This is again wrong, because the big gap between the first and the second jobs was due to the dependency between them and the delay in scheduling the first job. If it is not delayed, the following jobs should arrive earlier.

Moreover, we would expect that in a system with a better scheduler, the users will send more jobs. However, in conventional simulations, the users send jobs solely according to the timestamps. Therefore, using a workload from a poor scheduler in a conventional simulation of a better scheduler may lead to periods that the system is idle because there are not enough jobs to take advantage of the better scheduler. This may lead to unrealistically low wait times, response times, and slowdowns.

All these problems arise due to preferring timestamps over preserving the logical structure of the workload as reflected in job dependencies. The solution to these deficiencies is simulations that include internal feedback per user.

## VI. Mechanism of Feedback

Intuitively, when a user begins his interaction with the system, he sends several jobs (call them batch A). After a while he may send another batch of jobs (call them B). The jobs in B might depend on the results of A (we say B depends on A) and might not (they are independent). Our goal is to preserve this structure if it exists. In this section we describe how we do this, and in the next one we address the issue of setting arrival times.

Above we mentioned the definition of direct-dependency. One main property of general-dependency is the following:

**Observation 9.** *Dependency is a transitive relation. Meaning, if we have three batches $b_1$, $b_2$ and $b_3$ such that $b_3$ depends on $b_2$ and $b_2$ depends on $b_1$, than $b_3$ depends (indirectly) on $b_1$.*

This exists because if $b_3$ needs to wait for the results of $b_2$, and $b_2$ for $b_1$, then of course $b_3$ effectively waits for the results of $b_1$. In the future, what we say "dependency" we mean direct-dependency, and explicitly say "indirect dependency" when it is caused due to the transitive relation.

**Definition 10.** *The Dependency Graph is a directed graph, where the nodes are the batches, and there is an edge from $b_2$ to $b_1$ if $b_2$* depends on *$b_1$.*

**Observation 11.** *The Dependency Graph doesn't contain cycles.*

Now we will see how to handle dependencies in a simulation. For each batch we create a list of batches on which it depends. To preserve all the dependencies, we use the most conservative definition of dependencies. This means that the dependencies list of a batch will contain all the possible batches that this batch may depend on. The formal rule for dependency is the following:

**Definition 12.** *For each two batches $b_i$ and $b_j$ of the same user, we say that $b_i$* depends on *$b_j$ if one of the following exists:*

1) *$b_i$ and $b_j$ belong to the same session, and $b_i$* follows *$b_j$.*
2) *$b_i$ and $b_j$ belong to different sessions ($s_{b_i}$ and $s_{b_j}$ respectively), $b_i$ is the first batch in $s_{b_i}$, $b_j$ is the last batch in $s_{b_j}$, and all the jobs in $s_{b_j}$ finished before the arrival of the first job in $s_{b_i}$.*

Both rules for dependency are logical according to the definition of batches and sessions. An illustration of the Dependency Graph appears in Figure 4. Here we add an intuitive explanation:

1) The first rule means that in a session, except the first batch, each batch depends on the previous one. This reflects the assumption that the user sends the batch, waits for it to finish, and then sends the next batch. Due to the fact that dependency is transitive (Observation 9), it means that each batch depends indirectly on all the previous batches in the same session.
2) The second rule means that the first batch in each session depends on all the last batches in sessions that had finished before the arrival time of this batch. In order to explain when does batch B depends on A, when they're in different sessions, we expand the definition of dependence to sessions. Intuitively, $s_j$ may depend on $s_i$ ($j > i$) only if all the jobs in $s_i$ have finished before the beginning of $s_j$. Such a dependency between sessions will be preserved if the first batch in $s_j$ depends on the last batch of $s_i$.

Note that we have defined two distinct relations on batches: "follows" and "depends on". In many cases these relations overlap, for example with the sequence of batches in a session. But there are differences. To see this, compare Figure 4 and 8. Each has an edge marked with a * that doesn't appear in the other.

The goal of the feedback mechanism is not only to preserve both the depends on and the follows relations, in order to preserve both the dependencies and the unique jobs order created by each user [25]. The idea is to simulate the submittal of batches in the order defined by the follows relationship, subject to satisfying all the dependencies.

**Definition 13.** *The "next batch" is the batch that is next in line according to the* follows *order, meaning that all previous batches have been submitted already.*

At the beginning of the simulation, the user's first batch is initialized to be the next batch. Thereafter, when a batch is submitted the following batch becomes the next batch. However, the next batch should be delayed until all the batches it depends on finish. To maintain this information, we have a dependencies-list for each batch.

**Definition 14.** *A batch is called "available" if it is the next batch and its dependencies list is empty.*

Therefore, only the next batch may be available. When a batch is finished (after the termination of its last job), we delete it from the dependency lists of all the batches that depend on it. This may cause the next batch to become available, in which case it will be transmitted to the user behavior model to determine when it will be submitted. This is described in the next section.

The next batch mechanism preserves the follows relation by sending the next batch to the system (and deciding on its arrival time) only after the last arrival of the current batch. Therefore each user may have up to one batch in the simulation's queue of jobs that need to arrive in the future (meaning, its arrival time is decided, but the simulation didn't reach it yet). However, after the arrival of a batch, the next batch may be transmitted (if it is independent). That means that several batches of the same user may run in the simulation simultaneously.

As an example, consider the scenario presented in Figure 4, where we use the names $f_i$, $s_i$, $t_i$, to refer to the i-batch in the first, second and third session respectively. If we use only dependencies, after the end of $f_3$, two batches become independent: $s_1$ and $t_1$. Therefore, it is not clear that $s_1$ will be sent first. Moreover, to keep the original order, $s_2$ should be sent before $t_1$. But after $f_3$ the next batch is $s_1$, so the order is resolved as it was originally.

In summary, our mechanism has three main advantages:

1) The simulation never destroys a dependency between batches that existed in reality. That's because we use a very conservative definition of dependencies.
2) The simulation preserves the user's subtrace, due to the follows relation.
3) We grant importance to each job (except a few last jobs). If the job is a part of a batch that has at least one other batch depending on it, the performance of the job influences the arrival of at least one other batch (and most times of many batches). As a result, all the jobs, batches, and sessions are critical for the performance of a user.

## VII. The User Behavior Model

Up until now we described the mechanism for conducting a simulation with feedback. This is based on identifying batches and pacing them according to the behavior of each user. This is done by releasing a batch only when it becomes available, using the follows and depend on relations.

But there is another major open question before achieving a complete simulation: When a batch is released, what arrival times should be assigned to its jobs? This question in fact asks how the user responds to different delays. Does he take a break? Does he quit for the day? Of course, it is impossible to know the "correct" answer. We can only speculate how each user would react by using the data from the trace. We therefore suggest several different models for the user behavior. Each model preserves different characteristics of the real data. In the next subsections we will explain each model and show its results. Before that, we will give a little introduction here about the motivation of using think-times (TT) and inter-arrival times (IAT). We will also describe the presentation of our results.

In this section we will speak about times of a batch. The arrival of a batch is the arrival of the first job in this batch, and the termination of a batch is the termination of all the jobs in this batch. When we say that a batch is delayed by D or will arrive at T, we means that the first job will arrive at the current time + D or at T (respectively). The rest of the jobs in this batch have arrival times that preserve the original inter-arrival times between jobs belonging to the same batch.

When we speak about the arrival time of a batch, we refer to a batch that was affected by feedback. This means that the first batch of each user arrives at exactly the same time as in the original trace. This in addition to the fact that we use the same users, means that our simulation doesn't affect the throughput considerably. However, it can affect the throughput per user.

There are two main types of data that we use in all the user behavior models in order to set the arrival time of a batch.

- Think-time (TT) of a batch. This is the time during which the user thinks before sending off this batch. Therefore, this time is equal to the time from the termination of the batches that this batch depends on until the arrival of the first job in this batch. The reason for this definition of TT is that a user starts the TT only after the end of all the jobs that this batch is depended on.
  TT is used mainly when the next batch is this batch (the previous batch has arrived already), but the batch waited for the termination of the batches it depends on. In this case, the reason for the transmission of the batch at this time is the termination of all the dependencies of this batch. Therefore, the logical delay is to send off this batch after the original TT.
- Inter-arrival time (IAT). This is the time from the arrival of a batch to the last arrival of the previous batch. This is mainly used when the batch first became independent, and only after that the previous batch from the previous session has arrived to the system. Therefore the reason for the transmission of the batch at this time is the follows relation. In this case, the user doesn't need to think before sending the batch because it did not become available immediate after the fulfilling of the dependency. However, the batch should have a certain delay. Therefore, the intuitive step is using the original IAT as the delay.

An important observation is that if we simulate exactly the same scheduler, and we use the delays of the original TT and IAT, then all the jobs will arrive exactly at the same times. But with a different scheduler things may change. In the next subsections we present alternative user models, and compare simulations using these models with conventional simulations. This is done by recording the workloads as they unfold during each simulation and comparing them with each other. We also use graphs that present the distribution of some properties (e.g. average session length, number of sessions, etc.) across users, and the weekly and daily cycles by plotting the number of jobs that arrive in each hour during a week. To compare the throughput of different simulations, we present the distribution of the activity length per user. Due to the fact that each user sends the same jobs in all the simulations that use the same workload log, a longer activity length means that the user sends these jobs over a longer period, thus achieving a lower throughput, and vice versa. Unless stated otherwise, we use the EASY scheduler, which is probably the most commonly used backfilling scheduler [12], [3].

## A. *Adjusted User Model*

This basic model, which we have mentioned already, preserves the original TT and IAT of the batches. For example, if a batch finished later than originally by two hours and its termination caused another batch to be released, then the released batch will arrive two hours later too.

The biggest advantage of this model is that we use only the data from the trace. However, this leads to unrealistic behavior of the users — they will take the same break if the jobs finished during their normal working hours, at night, or on the weekend. As the simulated scheduler can make scheduling decisions that are completely different from the original one, this leads to a total destruction of the daily and weekly cycles, as shown in Figure 5. Moreover, due to the fact that the jobs distribute approximately equally during all the day and the hours, the performance is much improved as can be seen in Table I.

## B. *Distribution Based User Model*

This model is essentially the model of Shmueli and Feitelson in [20]. The idea is to categorize the users into four groups with different work patterns that are combinations of daytime vs. nightly work and weekdays vs. weekends work. Users are active only in their designated periods, and have dynamic session lengths that depends on the system's performance. However, due to several differences (for example, the definition of sessions and the fact that we retain the sequence of jobs for each user) several adaptations are required.

In the initialization, the model randomly gives each user several attributes. The first is whether the user is a daytime user (active between 7:30+RAND to 17:30+RAND) or nighttime user (17:30+RAND to 7:30+RAND) with probabilities of 0.7 and 0.3 respectively, where RAND is a random number between -1 hour to +1 hour that is chosen independently for each user. The second is whether the user is active during weekdays (Monday to Friday) or weekends (Saturday and Sunday) with probabilities of 0.8 and 0.2 respectively.

When a batch becomes available, we check if the current time is during the activity time of the user (working days and working hours). If it is not, the batch is delayed to the next active-time of this user. Otherwise, we apply the

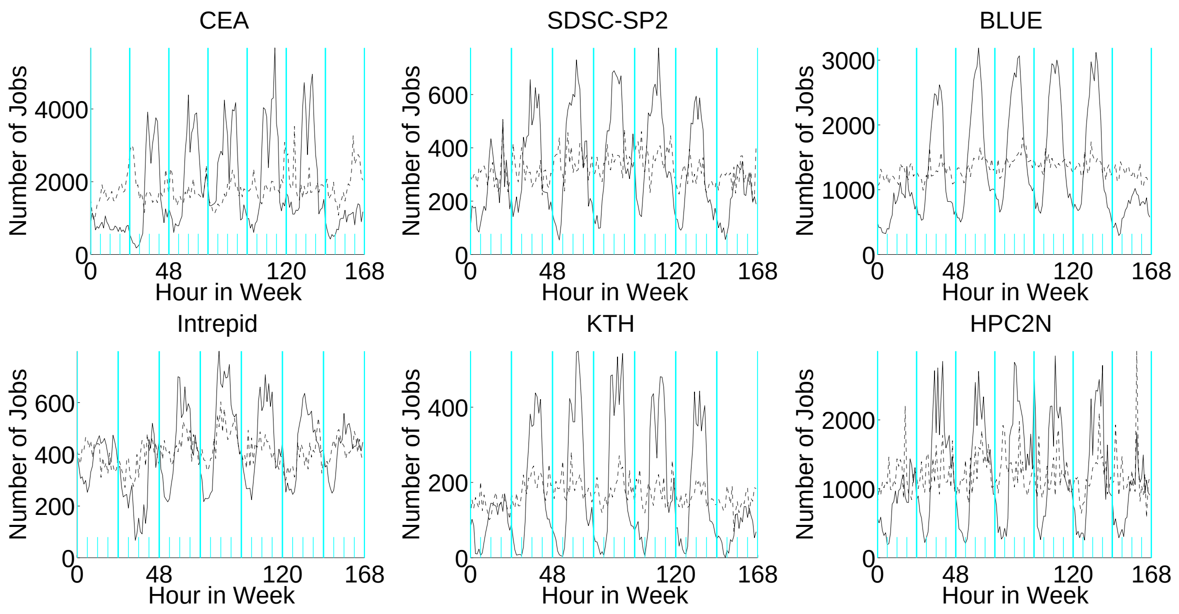

Fig. 5. Distribution of arrival times per week using the adjusted user model (dashed line) compared to the original trace (solid line).

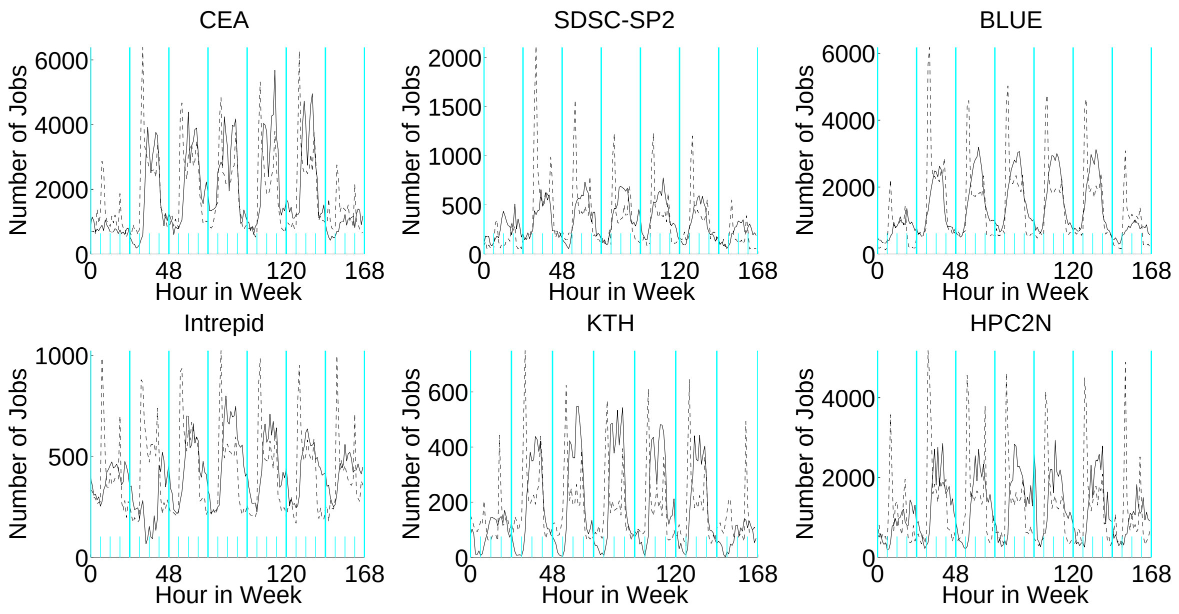

Fig. 6. Distribution of arrival times per week using the distribution based user model (dashed line) compared to the original trace (solid line).

| Trace | SDSC-DS | CEA | SDSC-SP2 | BLUE | Intrepid | KTH | HPC2N | CM5 |
|---|---|---|---|---|---|---|---|---|
| Conventional | 8353 | 6989 | 37363 | 8487 | 1242 | 8755 | 12591 | 20966 |
| Adjusted | 5005 | 5073 | 24731 | 7199 | 2307 | 8364 | 8167 | 3702 |
| Dist. based | 12976 | 6141 | 40579 | 47219 | 1369 | 11717 | 13347 | 3503 |
| Fluid | 5972 | 4395 | 18748 | 6826 | 1958 | 6389 | 10092 | 2614 |

TABLE I. THE AVERAGE WAIT TIME (IN SECONDS) OF THE DIFFERENT USER MODELS COMPARED TO CONVENTIONAL SIMULATIONS OF SEVERAL TRACES.

model of the probability to continue with the current session: $P_{\text{cont}} = \frac{0.8}{0.05*\text{RespTime}+1}$, which was defined in [19] based on data extracted from logs. If we continue the session, we choose a random TT or IAT between batches in the same session of this user, and the batch arrives at this time. Otherwise, this batch is delayed by a break. The break's length is a random TT or IAT between batches in different sessions that is smaller than eight hours. If after the break the batch starts after the working days/hours of this user, the batch will be delayed to the next activity of this user.

This model is based on empirical distributions, which has advantages and disadvantages. The main disadvantage is that we lose the connection with the real workload by having many

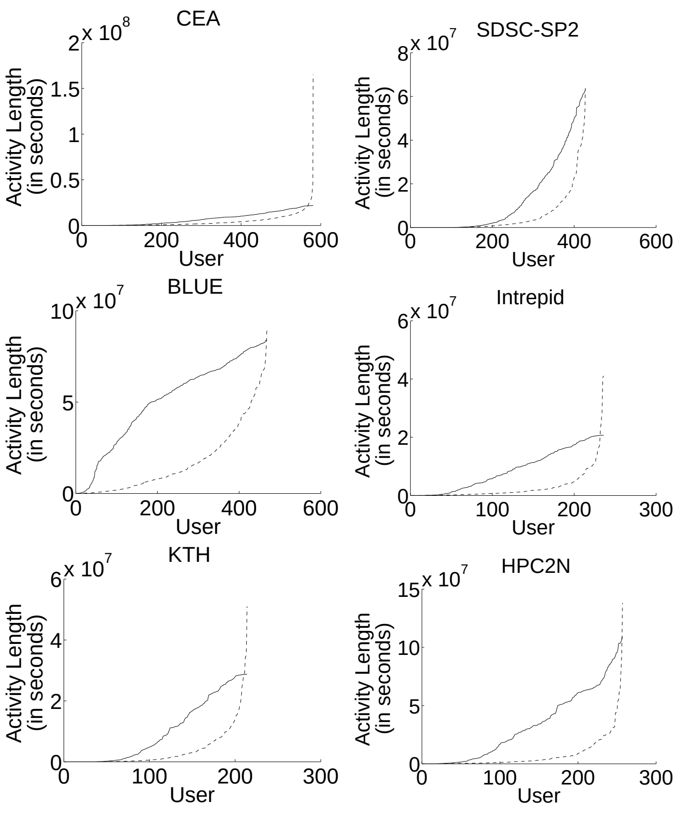

Fig. 7. Distribution of the activity length per user using the distribution based user model (dashed line) compared to the original trace (solid line).

assumptions. For example, assumptions include that the users don't take any long break (they work in each working-day), that night-time user don't send any job during the day, and that all users operate according to the same statistical model. For most of the workloads, this leads to high throughput per user (Figure 7) and long wait times (Table I). An apparent problem with the model is that all users arrive each day during two 2-hour slots, a large one in the morning and a smaller one in the evening. This creates sharp peaks of activity that do not exist in the original data, as seen in Figure 6.

An interesting artifact of this model is that the distributions of activity length in Figure 7 show that some few users are active for excessively long periods in the simulations. These were found to be hyperactive users (probably scripts) that submit many thousands of jobs in succession. The model blindly assigns them a user type, e.g. mandating that they only be active on weekdays at night. As a result they are forced to stop submitting jobs when their active time ends, so it takes them much longer to finish all their jobs. This is another example where departing from the original data leads to unwanted effects.

This model suggests that the probability to continue a session depends on the performance of the last job. This idea is central to our assumption, that feedback has an important effect, and therefore we will use it also in the next model.

### *C. Fluid User Model*

The idea of this model is to maintain the session times of the users. To do that, we keep the sessions' start and end points from the original workloads, and let the batches flow between the sessions according to the feedback effects. We will describe the algorithm assuming that the release reason is dependency. First we check if the current time is during a session of the user. If it is, the delay is chosen at random from the TT-distribution between batches in the same session of this user. Otherwise, the current batch can't continue the current session. Therefore we delay the batch to the beginning of the next session of this user. If the release reason is the follows relation, we do exactly the same, but we choose the delay from this user's IAT-distribution between batches in the same session.

One issue that this model needs to handle is what to do if the user's sessions are finished before all the user's batches have been simulated. Our solution to this situation is to recycle the sessions starting from the next week after the last session, maintaining the same days of the week and hours of the day.

The fluid model preserves the daily and weekly cycles, as can be seen in Figure 8. Also the throughput per user is close to the original workload in most of the traces (Figure 9). However, a few users have very long activity periods. The reason is that sessions may be skipped, but sessions are not added until the last session is finished. Therefore an user is likely to send less jobs during the same activity period. This also causes the wait time to be shorter (Table I).

## VIII. Comparing Feedback and Conventional Simulations

In the last section we described the feedback models in detail. Previously, we mentioned several motivations for using feedback. Here we compare the feedback based simulations to the conventional ones, and demonstrate briefly that our simulation really has the advantages mentioned there.

In order to compare the simulations, we used the conventional simulation and our simulation with the fluid user model to simulate the EASY scheduler, which is based on backfilling and may be expected to be better than the original scheduler, and FIFO which is much worse than the original scheduler. The results for the queue length distribution are presented in Figure 10. In our simulation, the queue when simulating FIFO tends to be a bit longer on average than when simulating EASY, but the difference is very small. The reason is that users adapt themselves to the lack of available resources, and skip sessions when their jobs haven't been handled yet. However, in the conventional simulations with FIFO, the simulation continues to receive jobs according to the original tempo, and the users don't adapt their behavior to the simulated system. As a result, for a very large fraction of the simulation, there are unreasonably huge queue lengths. In our simulations such long queues occurred only for Intrepid, and also that only for 2% of the time. Another interesting effect which is common to all the logs is that our simulation with EASY also has shorter queue lengths on average relative to conventional simulations. The reason is that in our simulations the users finish their jobs earlier on average than they did originally, and therefore they may submit the next jobs earlier.

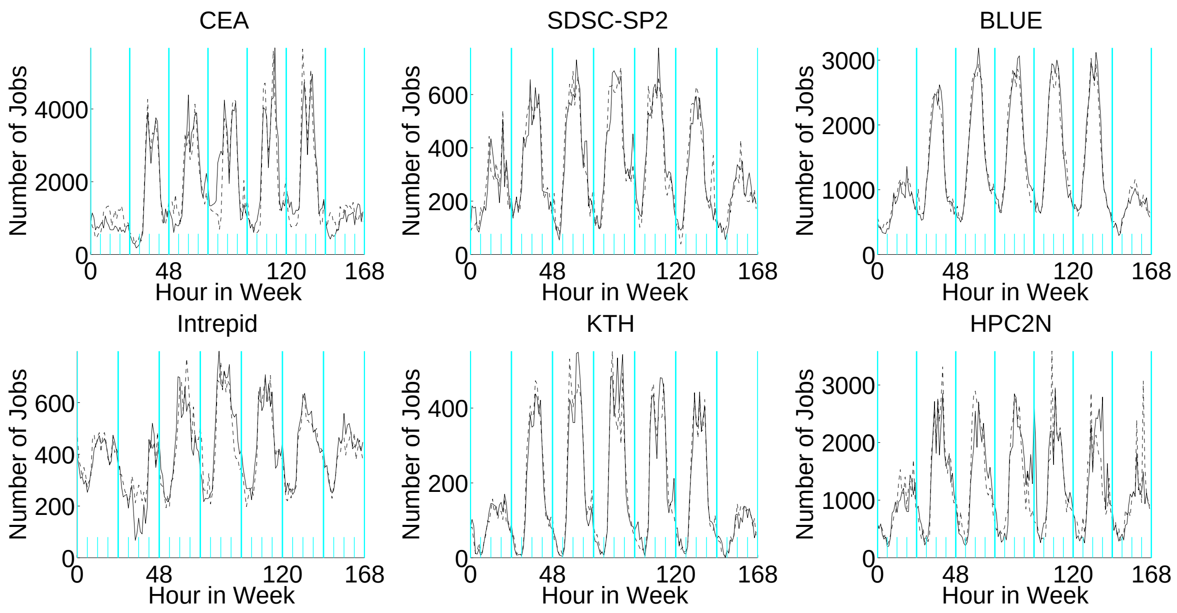

Fig. 8. Distribution of arrival times per week using the fluid user model (dashed line) compared to the original trace (solid line).

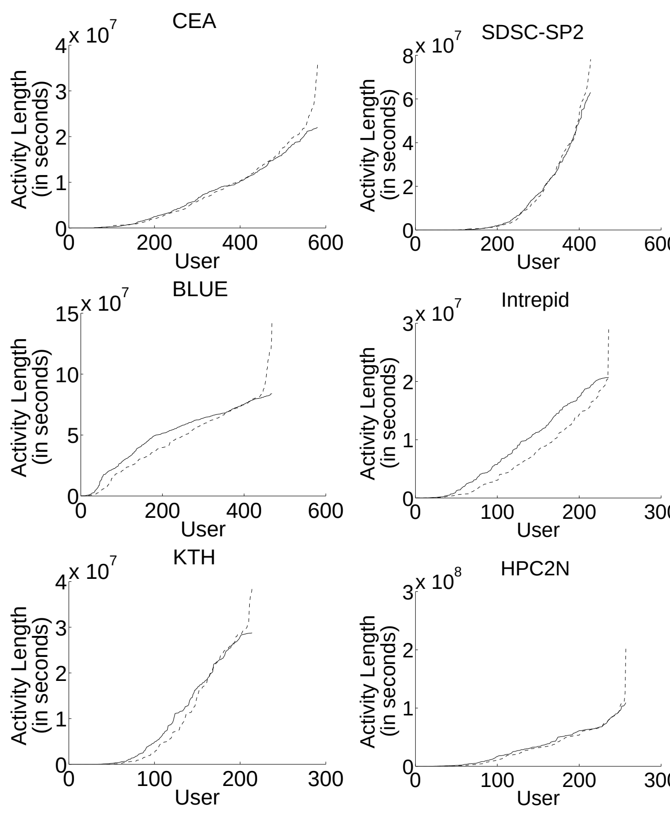

Fig. 9. Distribution of the activity length per user using the fluid user model (dashed line) compared to the original trace (solid line).

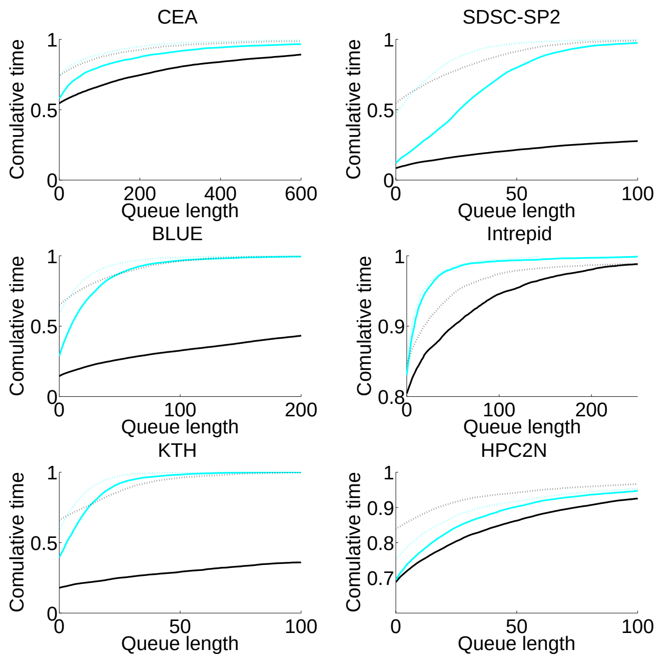

Fig. 10. Distributions of queue lengths during simulations: conventional simulation (solid line) is compared with feedback-based simulation with fluid user model (dashed line), using the EASY (Cyan) and FIFO (Black) schedulers.

## IX. Conclusions

To evaluate the performance of parallel system schedulers analysts typically use open-system trace-driven simulations. The reason for using traces is to be as close to real workloads as possible. However, this doesn't take into account the influence of the scheduler on the workload, effectively implying the assumption that users behave in exactly the same manner regardless of the scheduling policy. This contradicts the more reasonable assumption that users behave differently under different circumstances.

We argue that striving to preserve the details of traced workloads, and in particular the precise timestamps at which events occur, is misguided. In particular, it has the unintended consequence of breaking the logical structure of the workload.

Instead, we suggest that the logical structure of dependencies should be preserved, and the timestamps adjusted as needed. Using this approach also has two additional benefits. First, the feedback effect prevents excessive buildup of load when the system scheduler cannot keep up with the users. Second, it becomes possible to measure throughput, which is an important metric of performance.

The way to include feedback in trace-driven simulations is to first extract dependency information from the trace. The simulation then unfolds by simulating the submittal of only independent batches of jobs. When each batch finishes, and depending on its performances, the user model decides when to submit the next batch of this user. In fact we identify two types of dependency: one where batches depend on the termination of jobs in previous batches (the depends on relation), and another that maintains each user's sequence of jobs (the follows relation).

However, the question of how users react to different performance levels is extremely complicated. The goal is to understand user behavior and try to simulate it. We considered two simple models, and then suggested the fluid model which uses the sessions data from the trace instead of trying to simulate user session dynamics. Additional research on the behaviors of users and how it can be decoded from the workload data is necessary to improve the feedback model. Our goal in this paper is to supply the basic mechanism of feedback and to promote a new understanding of what it means to "be close to the original workload".

The proposed simulation simulates the feedback effect per user and uses the same jobs as in a recorded trace. Therefore, it may lead to different throughput of each user, but the global throughput will be approximately the same. An interesting future work is to develop a feedback based simulation where the feedback also affects the user population. For example, a user may leave the system due to poor performance or send more jobs if the performance is good. In such a simulation, the overall throughput may be expected to reflect the quality of the system.

*Acknowledgments*

This research was supported by the Ministry of Science and Technology, Israel. Heartfelt thanks to all those who contributed data to the Parallel Workloads Archive.

## References

[1] P. Cremonesi and G. Serazzi, "*End-to-end performance of web services*". In *Performance Evaluation of Complex Systems: Techniques and Tools*, M. C. Calzarossa and S. Tucci (eds.), pp. 158–178, Springer-Verlag LNCS vol. 2459, 2002, DOI:10.1007/3-540-45798-4_8.

[2] P. A. Dinda, G. Memik, R. P. Dick, B. Lin, A. Mallik, A. Gupta, and S. Rossoff, "*The user in experimental computer systems research*". In *Workshop Experimental Comput. Sci.*, Jun 2007, DOI: 10.1145/1281700.1281710.

[3] Y. Etsion and D. Tsafrir, *A Short Survey of Commercial Cluster Batch Schedulers*. Tech. Rep. 2005-13, Hebrew University, May 2005.

[4] D. G. Feitelson, "*Experimental analysis of the root causes of performance evaluation results: A backfilling case study*". *IEEE Trans. Parallel & Distributed Syst.* **16(2)**, pp. 175–182, Feb 2005, DOI: 10.1109/TPDS.2005.18.

[5] S. Floyd and V. Paxson, "*Difficulties in simulating the Internet*". *IEEE/ACM Trans. Networking* **9(4)**, pp. 392–403, Aug 2001, DOI: 10.1109/90.944338.

[6] G. R. Ganger and Y. N. Patt, "*Using system-level models to evaluate I/O subsystem designs*". *IEEE Trans. Comput.* **47(6)**, pp. 667–678, Jun 1998, DOI:10.1109/12.689646.

[7] F. Guim, I. Rodero, and J. Corbalan, "*The resource usage aware backfilling*". In *Job Scheduling Strategies for Parallel Processing*, E. Frachtenberg and U. Schwiegelshohn (eds.), pp. 59–79, Springer-Verlag LNCS vol. 5798, 2009, DOI:10.1007/978-3-642-04633-9_4.

[8] W. W. Hsu, A. J. Smith, and H. C. Young, "*The automatic improvement of locality in storage systems*". *ACM Trans. Comput. Syst.* **23(4)**, pp. 424–473, Nov 2005, DOI:10.1145/1113574.1113577.

[9] J. Jann, P. Pattnaik, H. Franke, F. Wang, J. Skovira, and J. Riodan, "*Modeling of workload in MPPs*". In *Job Scheduling Strategies for Parallel Processing*, pp. 95–116, Springer-Verlag LNCS vol. 1291, 1997, DOI:10.1007/3-540-63574-2_18.

[10] D. Klusáček and H. Rudová, "*Performance and fairness for users in parallel job scheduling*". In *Job Scheduling Strategies for Parallel Processing*, W. Cirne et al. (eds.), pp. 235–252, Springer-Verlag LNCS vol. 7698, 2012, DOI:10.1007/978-3-642-35867-8_13.

[11] D. Krishnamurthy, J. A. Rolia, and S. Majumdar, "*A synthetic workload generation technique for stress testing session-based systems*". *IEEE Trans. Softw. Eng.* **32(11)**, pp. 868–882, Nov 2006, DOI: 10.1109/TSE.2006.106.

[12] D. Lifka, "*The ANL/IBM SP scheduling system*". In *Job Scheduling Strategies for Parallel Processing*, pp. 295–303, Springer-Verlag LNCS vol. 949, 1995, DOI:10.1007/3-540-60153-8_35.

[13] U. Lublin and D. G. Feitelson, "*The workload on parallel supercomputers: Modeling the characteristics of rigid jobs*". *J. Parallel & Distributed Comput.* **63(11)**, pp. 1105–1122, Nov 2003, DOI:10.1016/S0743-7315(03)00108-4.

[14] R. Morris and Y. C. Tay, *A Model for Analyzing the Roles of Network and User Behavior in Congestion Control*. Tech. Rep. MIT-LCS-TR898, MIT Lab. Computer Science, May 2003.

[15] R. S. Prasad and C. Dovrolis, "*Measuring the congesion responsiveness of Internet traffic*". In 8th *Passive & Active Measurement Conf.*, pp. 176–185, Apr 2007, DOI:10.1007/978-3-540-71617-4_18.

[16] M. Satyanarayanan, "*The evolution of Coda*". *ACM Trans. Comput. Syst.* **20(2)**, pp. 85–124, May 2002, DOI:10.1145/507052.507053.

[17] B. Schroeder, A. Wierman, and M. Harchol-Balter, "*Open versus closed: A cautionary tale*". In 3rd *Networked Systems Design & Implementation*, pp. 239–252, May 2006.

[18] E. Shmueli and D. G. Feitelson, "*Using site-level modeling to evaluate the performance of parallel system schedulers*". In 14th *Modeling, Anal. & Simulation of Comput. & Telecomm. Syst.*, pp. 167–176, Sep 2006, DOI:10.1109/MASCOTS.2006.50.

[19] 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.*, pp. 274–280, Oct 2007, DOI:10.1109/MASCOTS.2007.67.

[20] E. Shmueli and D. G. Feitelson, "*On simulation and design of parallel-systems schedulers: Are we doing the right thing?*" *IEEE Trans. Parallel & Distributed Syst.* **20(7)**, pp. 983–996, Jul 2009, DOI: 10.1109/TPDS.2008.152.

[21] A. Spink and T. Saracevic, "*Human-computer interaction in information retrieval: Nature and manifestations of feedback*". *Interacting with Computers* **10(3)**, pp. 249–267, Jun 1998, DOI:10.1016/S0953-5438(98)00009-5.

[22] D. N. Tran, W. T. Ooi, and Y. C. Tay, "*SAX: A tool for studying congestion-induced surfer behavior*". In 7th *Passive & Active Measurement Conf.*, Mar 2006.

[23] S. Yang and G. de Veciana, "*Bandwidth sharing: The role of user impatience*". In *IEEE Globecom*, vol. 4, pp. 2258–2262, Nov 2001, DOI:10.1109/GLOCOM.2001.966181.

[24] N. Zakay and D. G. Feitelson, "*On identifying user session boundaries in parallel workload logs*". In *Job Scheduling Strategies for Parallel Processing*, W. Cirne et al. (eds.), pp. 216–234, Springer-Verlag LNCS vol. 7698, 2012, DOI:10.1007/978-3-642-35867-8_12.

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