# Improving Realism and Representativeness of Workloads to Achieve More Reliable Performance Evaluations

Thesis for the degree of
"Doctor of Philosophy"

By

**Netanel Zakay**

Submitted to the senate of the Hebrew University of Jerusalem
September 2017

This work was carried out under the supervision of Prof. Dror G. Feitelson.

# Acknowledgments

With the submittal of this thesis, I complete ten successive years of studying Computer Science at the Hebrew University of Jerusalem. From my Bachelor degree, through Master, and now Ph.D. The Journey was not a simple one, but interesting and challenging. It is much more than acquiring knowledge and experience - it changed the way I think.

I first want to express my appreciation to my parents. My parents are manual laborers. They did not even finish elementary school. When they raised their own children, however, they had an ambition that their children will not be the same as they were. They supplied me with unlimited support and aid whenever I needed. They always encouraged me to prioritize my education and study hard, and they definitely share every achievement that I will ever accomplish.

Dror Feitelson is one of the main reasons that I decided to expand my Master thesis into a Doctorate in the first place. He is much more than a supervisor. Dror is my mentor. Except being a source of knowledge and advice when I needed, Dror is incredibly patient and supportive. He made this journey much simpler and I will always thank him for that.

Finally, I want to thank my siblings Efrat, Liat and Sasi. Gil Zaharoni, my best friend, which was an integral part of the journey. My friends and colleagues. And the Hebrew University for hosting me for a decade.

# Abstract

## Improved Realism and Representativeness of Workloads to Improve Reliability of Performance Evaluations

שיפור המציאותיות והייצוגיות של עומסי עבודה בכדי להשיג הערכות ביצועים אמינות יותר

Student: Netanel Zakay

Supervisor: Prof. Dror G. Feitelson

When a new system design is proposed, 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 performance of a computer system is affected by the workload it handles. Reliable performance evaluations therefore require the use of representative workloads. As a result, traces of real workloads are often used to drive simulations of new system designs, because such traces obviously contain all the structure found in real workloads.

Replaying a trace provides only a single data point of performance for one workload. However, in many evaluations, several related workloads are needed. For example, in order to compute confidence intervals, one needs multiple instances of the same basic workload. The common way to satisfy this need is to create multiple synthetic workloads based on statistical workload models (which, in turn, are based on the traced data). While models provide the required variability and flexibility for evaluations, they also suffer from not necessarily including all the important features of the real workload.

To improve the representativeness of evaluation workloads we propose to *combine the realism of real traces with the flexibility of models*. This is done by modeling only the part of the workload that needs to be manipulated, and resampling from the real data to fill in the remaining details [2, 3].

Using such resampling, we can achieve the following: multiple similar workloads (used to compute confidence intervals), workloads with higher or lower average loads (used to investigate how load affects system performance), a much longer trace than the original (used to ensure convergence of evaluation results), and workloads in which rare events such as surges in activity are amplified (used to investigate the effect of such events).

Importantly, while the resampled workloads differ from the original in length, statistical variation, or load, they nevertheless retain important elements of the internal structure such as sessions and the relationship between the sessions and the daily work cycle. They are even found to have the same long-range dependence structure.

The simulations commonly used to evaluate the proposals of new system designs 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 are 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. Furthermore, performance in such simulations is measured by the average wait time and slowdown, under the fixed load and throughput conditions dictated by the trace. They cannot evaluate the system's effect on throughput and productivity.

Resampling alone does not change this. Using resampled 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.

For this purpose, we first need to understand the users' behavior. One important characteristic of user behavior is its temporal pattern. Human users may work for some time, but then they stop and do something else. The periods of continuous work are called sessions. Data about user behavior is contained in accounting logs. Unfortunately, these logs only include data about individual jobs. We analyze different approaches to identify and characterize the users' sessions from a recorded trace [1]. We used this understanding in all the rest of the papers in order to recreate the users' behavior.

Another important characteristic of user behavior is the workflow. Using dependency information extracted from traces [4], we show how a simulation can preserve these dependencies. This creates semi-open trace based simulations that include dynamic user activity and internal feedback from the system to the users. In these simulations, like in a real system, users adjust their job-submittal behavior in response to system performance. As a result, the simulations produce different loads and throughputs for different scheduling algorithms or parametrizations [5]. We also propose the User Priority Scheduler (UPS) that produces higher throughput when evaluated with semi-open simulations, but these improvements cannot be observed or analyzed using the conventional open-model simulations.

Open and closed systems are commonly used in analytical modeling for performance evaluations. However, semi open systems as proposed in [5] are less common, despite being more representative and appropriate for many systems. Moreover, in open systems the throughput is an input of the simulation, and cannot be evaluated. Currently the common solution is to use closed-system models, which are often unrepresentative. As an alternative, we define two models of semi open systems and analyze their characteristics. We describe their parameters, how to calculate the performance metrics, and present interactions between the different metrics (e.g. wait time and throughput) for both approaches. This increases the understanding of semi-open systems and shows their usefulness also from the analytical modeling aspect. This work, named *Models for Evaluating Throughput*, is an unpublished work yet, and is a part of the thesis as well.

## Bibliography

[1] N. Zakay and D. G. Feitelson, "*On identifying user session boundaries in parallel workload logs*". In J*ob Scheduling Strategies for Parallel Processing*, W. Cirne et al. (eds.), pp. 216–234, Springer-Verlag, 2012. Lect. Notes Comput. Sci. vol. 7698.

[2] N. Zakay and D. G. Feitelson, "*Workload resampling for performance evaluation of parallel job schedulers*". In *4th Intl. Conf. Performance Engineering*, pp. 149–159, Apr 2013.

[3] N. Zakay and D. G. Feitelson, "W*orkload resampling for performance evaluation of parallel job schedulers*". *Concurrency & Computation — Pract. & Exp*. 26(12), pp. 2079–2105, Aug 2014.

[4] N. Zakay and D. G. Feitelson, "*Preserving user behavior characteristics in trace-based simulation of parallel job scheduling*". In 22nd *Modeling, Anal. & Simulation of Comput. & Telecomm. Syst.*, pp. 51–60, Sep 2014.

[5] N. Zakay and D. G. Feitelson, "*Semi-open trace based simulation for reliable evaluation of job throughput and user productivity*". In 7th *IEEE Intl. Conf. Cloud Comput. Tech. & Sci.*, pp. 413– 421, Nov 2015.

# Letter of Contribution

This thesis is based on the following papers:

1. N. Zakay and D. G. Feitelson, "*On identifying user session boundaries in parallel workload logs* ". In J*ob Scheduling Strategies for Parallel Processing*, W. Cirne et al. (eds.), pp. 216–234, Springer-Verlag, 2012. Lect. Notes Comput. Sci. vol. 7698.
2. N. Zakay and D. G. Feitelson, "*Workload resampling for performance evaluation of parallel job schedulers* ". In *4th Intl. Conf. Performance Engineering*, pp. 149–159, Apr 2013.
3. N. Zakay and D. G. Feitelson, "W*orkload resampling for performance evaluation of parallel job schedulers* ". *Concurrency & Computation — Pract. & Exp*. 26(12), pp. 2079–2105, Aug 2014. This is an extended journal version of the original paper.
4. N. Zakay and D. G. Feitelson, "*Preserving user behavior characteristics in trace-based simulation of parallel job scheduling* ". In 22nd *Modeling, Anal. & Simulation of Comput. & Telecomm. Syst.*, pp. 51–60, Sep 2014.
5. N. Zakay and D. G. Feitelson, "*Semi-open trace based simulation for reliable evaluation of job throughput and user productivity* ". In 7th *IEEE Intl. Conf. Cloud Comput. Tech. & Sci.*, pp. 413– 421, Nov 2015.
6. N. Zakay and D. G. Feitelson, "*Models for Evaluating Throughput*", unpublished.

There are five papers and an unpublished work. In all of them, there are only two authors: Netanel Zakay and Dror G. Feitelson. In other words, there are no other co-authors or contributors.

# Contents

# Chapter 1

# Introduction

## 1.1 Performance Evaluations

Implementing a new system design is a time consuming and highly expensive project. Depending on the system, it usually requires several teams of engineers for a long term project. As a result, it is highly important that a new system design will be implemented only if it demonstrates dramatic improvements over the current design.

Moreover, due to the amount of time (and therefore also costs) that it takes to change the design after a first release of the product, it is important to choose the best design out of a few proposed designs in advance. This creates a need to evaluate a proposed system design before the actual implementation. This pre-product evaluations allows us to analyze the proposed system's improvements over the current designs, and to improve the system design by considering multiple options, evaluating them, and choosing the best. This domain, named performance evaluations, is essential and important part of system designs. It deals with the challenges of faithful evaluations, which reflect all the complexities of the real system.

The first approach suggests to use analytical modeling for computer systems evaluations. These systems designers mathematically model the system, stochastically characterize the workloads and performance goals, and then analytically derive the performance of the system as a function of workload and input parameters. This is based on queuing-theoretic modeling and analysis, including Operational Laws, Markov Chains (discrete-time and continuous), $M/M/1$, $M/M/k$, $M/M/k/k$, $M/M/\infty$, MVA for closed system analysis, Burke's Theorem and Jackson Product Form [40, 45]. The approach of analytical modeling has two main drawbacks. The first is the difficulty to model current, complicated systems in this way. The second is that some of the common assumptions are incorrect, for example Poisson distribution of the jobs' arrivals.

As a result, people tend to use simulations. Simulations are widely used in various areas and domains in order to evaluate the performance of new system designs. They are common for decades in both the industry and the academy. Therefore, improving the reliability of simulations has big impact and leads to better designs and reduced expenses in many domains. Understanding this, people created new simulations which are adapted to their own domain. For example, *Parsec* is a parallel simulation environment for complex systems [4], *NanoNS* [31], *ns2-MIRACLE* [5], *OverSim* [7], and *Planetsim* [29] were proposed as network simulators, *CloudSim* [10], *CloudGrid* [12], and *Networkcloudsim* [30] for clouds, *Alea* [36], *GroudSim* [47], and *Gridsim* [9] for grid scheduling and resouce managment, *MDCSim* [41] for data centers, *Cheddar* [57] for general scheduling framework, *Multi2Sim* [61] for CPU-GPU comupting, *SensorSim* [48] for sensor networks, *Simflex* [64] for server architecture, *Simics* [44] for embedded systems, and *Feedback control real-time scheduling* for real-time system. In addition, *SimGrid* is a framework that is capable to do simulations and emulation for many fields including: HPC, Cloud, Grid, and Scheduling. Relaying on the strong validation of its network and CPU models, may simulator are built on top of SimGrid, including some that are designed to study parallel system scheduling.

Modern simulations are complex and composed of several components. We will focus on a single aspect of this complexity — the workload that the system receives. While other components of the simulations are researched, developed, and improved, the workload often went under the radar. The performance of a computer system is affected by the workload it handles. This means that a system design may be highly efficient for certain cases (workloads) but inefficient for typical usage of the system. Therefore, reliable performance evaluations require the use of representative workloads. The goal is to analyze the performance in the typical cases, using representative workloads scenarios, rather than the worst case scenario. This aims to provide an estimation of the expected performance of the system when it will be finally deployed.

## 1.2 Parallel Systems

Our goal is to improve the workload used in simulations to be more representative and flexible. While our goal is to improve the methodology of general simulations in any domain, in order to achieve concrete results based on real experiments, we needed to focus on a certain domain. Therefore, we focus on simulations that evaluate the performance of the scheduler of a parallel system.

Parallel computing is a type of computation in which many calculations or the execution of processes are carried out simultaneously. Large problems can often be divided into smaller ones, which can then be solved at the same time. Parallelism has been employed for many years, mainly in high-performance computing, but interest in it has grown lately due to the physical constraints preventing frequency scaling. As power consumption (and consequently heat generation) by computers has become a concern in recent years, parallel computing has become the dominant paradigm in computer architecture, mainly in the form of multi-core processors.

There are thousands of parallel supercomputers across the world. They are owned by governments (such as United States, China, Japan, and more), organizations and private companies (such as Intel, IBM, Google and Amazon). The TOP500 list ranks and details the 500 most powerful non-distributed computer systems in the world. For example, the most powerful supercomputer in the world now is Sunway Taihulight owned by China. It employs a total of 10,649,600 CPU cores and supplies up to 93.01 PFLOPS. Next there is Tianhe-2 (China, 33.86 PFLOPS), Cray Titan (USA, PFLOPS), and Sequoia (IBM, 17.17 PFLOPS). Usually, these types of system are used primarily for nuclear weapons simulation, security simulations, as well as scientific purposes such as astronomy, energy, lattice QCD, study of the human genome, and climate change. These supercomputers cost dozens of millions of dollars, and their maintenance costs millions of dollars per year (only the energy consumption alone costs a few millions per year). Therefore it is clear that improving their algorithms, utilization, and power consumption is highly desired. However, this field doesn't belong only to governments and corporations. Currently more and more companies build their own parallel system, and use it for internal uses. For example, running massive tests on programs or hardware.

In parallel computing, a computational task (job) is typically broken down into several, often many, very similar sub-tasks that can be processed independently, and whose results are combined afterward, upon completion. When a user of a parallel system wants to execute a job on the system, he send it to a certain computer that serves as the gateway of the system. Each job contains information regarding the required processors number, the estimated run time, user id, and in some cases also memory demands. The Scheduler is the component that receives a job, and decides whether the job will run immediately (and allocate the required processors for the jobs' processes) or it will wait until some conditions are met and the job can run (e.g. another job terminated and there are enough avaible processors to run the job). These systems usually does not use preemption. Therefore, in order to run the job, there is a need to allocate a distinct processor for each process.

The most basic scheduler is First Come First Serve (FCFS). It serves the jobs according to their arrival order. FCFS schedules the longest-waiting job if it can run (e.g. there are enough available processors). FCFS is usually implemented by

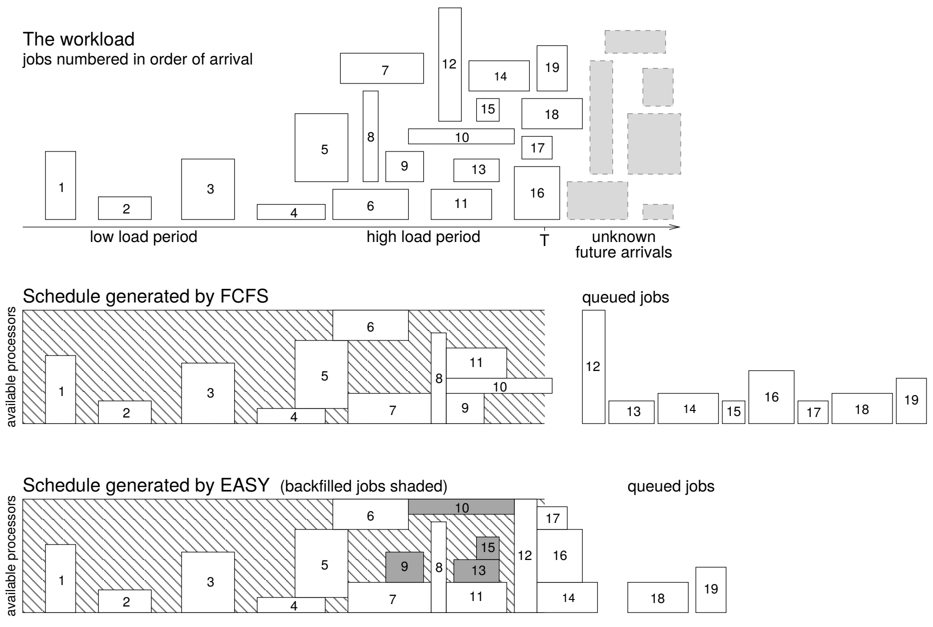


Figure 1.1: Illustration of a sequence of parallel jobs (the workload) and how it would be scheduled by FCFS and EASY up to time T.

a queue of waiting jobs, enqueuing new jobs that arrive to the system and can't run immediately, and dequeuing jobs when they are scheduled. The problem with FCFS is usually low utilization and long wait times. For example, if the first arrived job requires many processors and can't run currently, perhaps another job can be scheduled meanwhile? This methodology is named backfilling — filling holes in the first-come first-served order by scheduling jobs from the back of the waiting jobs queue. This packs the jobs more tightly and optimizes response time and slowdown. EASY is a common scheduler algorithm, which backfills a job only if it does not delay the first waiting job's estimated start time. Figure 1.1 demonstrates the differences between FCFS and EASY.

We demonstrate our concepts and methodology in a simulation that evaluates the performance of a new scheduler design for a parallel system. To produce representative workloads, we used the Parallel Workload Archive, which contains recorded traces from large-scale systems around the world. Some logs that we used often are listed in Table 1.1. Each log contains tens of thousands jobs, with all the relevant information, including: arrival time, start time, run time, number of processors, requested run time, user ID, and more. Each log was recorded during six months to a few years. The logs were analyzed in details and cleaned from

| Log name | File | Period | PEs | Users | Jobs |
|---|---|---|---|---|---|
| LANL-CM5 | LANL-CM5-1994-3.1-cln | 10/94–09/96 | 1024 | 213 | 122,060 |
| SDSC-Par | SDSC-Par-1995-3.1-cln | 12/94–12/95 | 400 | 98 | 53,970 |
| CTC-SP2 | CTC-SP2-1996-2.1-cln | 06/96–05/97 | 338 | 679 | 77,222 |
| KTH-SP2 | KTH-SP2-1996-2 | 09/96–08/97 | 100 | 214 | 28,489 |
| SDSC-SP2 | SDSC-SP2-1998-3.1-cln | 04/98–04/00 | 128 | 437 | 59,725 |
| OSC-cluster | OSC-Clust-2000-3.1-cln | 01/00–11/01 | 178 | 253 | 36,097 |
| SDSC-BLUE | SDSC-BLUE-2000-3.1-cln | 04/00–01/03 | 1152 | 468 | 243,314 |
| HPC2N | HPC2N-2002-1.1-cln | 07/02–01/06 | 240 | 257 | 202,876 |
| SDSC-DS | SDSC-DS-2004-1 | 03/04–04/05 | 1664 | 460 | 96,089 |
| ANL-Intrepid | ANL-Intrepid-2009-1 | 01/09–09/09 | 163,840 | 236 | 68,936 |
| PIK-IPLEX | PIK-IPLEX-2009-1 | 04/09–07/12 | 2560 | 225 | 742,965 |
| CEA-Curie | CEA-Curie-2011-2.1-cln | 02/12–10/12 | 93,312 | 582 | 312,826 |

Table 1.1: Logs from the Parallel Workloads Archive (www.cs.huji.ac.il/labs/parallel/workload/) that were used in this study.

anomalies [25, 26]. This is one of the biggest, most-intensively used workloads archive for parallel systems.

## 1.3 Conventional Simulations' Workloads

The performance of a computer system is affected by the workload it handles. Reliable performance evaluations therefore require the use of representative workloads. This means that the evaluation workload should not only have the same marginal distributions as the workloads that the system will have to handle in production use, but also the same correlations and internal structure. As a result, traces of real workloads are often used to drive simulations of new system designs, because such traces obviously contain all the structure found in real workloads.

However, replaying a trace only provides a single data point of performance for one workload. But in many evaluations, several related workloads are needed. For example, in order to compute confidence intervals, one needs multiple instances of the same basic workload. Moreover, we often would like to evaluate the system under different conditions. The common way to satisfy this need is to create multiple synthetic workloads based on statistical workload models (which, in turn, are based on the traced data) [34, 6, 43, 58, 62]. While models provide the required variability and flexibility for evaluations, they also suffer from not necessarily including all the important features of the real workload [24, 2]. In fact, they include only those of which the modeler was aware.

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 reflect the activity of real users in this system, but on the original system.

In open-system models, the throughput and utilization of the system being evaluated are dictated by the timestamps. The inability to measure the throughput and utilization creates a need 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, But throughput is a more direct measurement of the user productivity.

## 1.4 Our Research

Our research's goal is to improve the evaluations of new system designs, by improving the workload that the simulation handles. This is expected to lead to better designs at reduced costs. We suggest a new novel ways to create workloads, based on recorded traces, that provide variability, flexibility, and representativeness of real users.

The current existing ways to obtain workloads for a simulation are to use either a recorded trace or statistical workload models. This means that the system designer needs to decide whether to have a single data point that represents realistic work, or multiple data points such that their representativeness is in doubt. Currently there is no solution for workloads that are representative on one hand, and allow variability and flexibility for evaluations on the other hand.

As a solution we propose in Chapter 3 a new approach for creating workloads named Resampling. Resampling combines the realism of real traces with the flexibility of models. This is be done by modeling only the part of the workload that needs to be manipulated, and resampling from the real data to fill in the remaining details. Technically this is done by partitioning workload traces into their basic components, which proved to be users, and regrouping them in different ways to achieve the desired effects.

Other problems of the current workloads arise by the use of an open system model. 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 dont take into account that traces contain a signature of the scheduler that was used on the traced system [54]. 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 this purpose, we first needed to understand the users behavior. An important characteristic of user behavior is its temporal pattern. Human users may work for some time, but then they stop and do something else. The periods of continuous work are called sessions. Data about user behavior is contained in accounting logs. Unfortunately, these logs only include data about individual jobs. In Chapter 2, we analyze different approaches to identify and characterize the users' sessions from a recorded trace.

Using this understanding, in Chapter 4 we propose a novel feedback-based simulation. We integrate a feedback from the system to the users with trace-based simulations, by extracting dependencies from the trace. Given the dependencies, we adapt the arrival times in the simulations to reflect the users' reactions to system performance, while preserving all other properties of the recorded trace, including the users sessions and think times. The feedback reproduces the fine-grained interactions that naturally exist between the users and the system in reality. This creates a new workload, that reflects the users' activity on the simulated system. 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. Pay attention that these are not preserved in the conventional simulations.

In Chapter 5, we present the combination of Resampling (Chapter 3) and Feedback (Chapter 4) into a single complete simulation: the Trace Based User Oriented Simulation (TBOUS). TBOUS inherits the properties of both its parts: variability and flexibility while preserving the recorded data of Resampling, as well as simulating correctly the users responses to the simulated system design of Feedback. Feedback alone simulates a feedback between the user's jobs. However, the workload contains all the recorded jobs and runs for the same duration. As a result, when using Feedback alone the throughput is fixed, regardless of the system design. The combination between Resampling and Feedback creates a semi-open system, with dynamic throughput. This means that 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. We also propose a new scheduler, the User Priority Scheduler (UPS), that improves the throughput when evaluated with TBOUS, but can not be analyzed correctly using conventional simulations.

Open and closed systems are commonly used in analytical modeling for performance evaluations. However, semi open systems as we proposed in simulations (e.g. Chapter 5) are less common, despite they might be more representative and appropriate for many systems. Moreover, in open systems the throughput is the input of the simulation, and can not be evaluated. Currently the common solution is to use closed-system models, which is often unrepresentative. As an alternative, in Chapter 6, we define two models of semi open systems and analyze their characteristics. We describe their parameters, how to calculate the performance metrics, and present interaction between the different metrics (e.g. wait time and throughput) for both approaches. This increases the understanding of semi-open systems and shows their usefulness also from the analytical modeling aspect.

## 1.5 Related Work

This work touches upon three distinct concerns: the workloads used to evaluate parallel job schedulers, the simulation methodology, and the policy considerations of the schedulers.

In terms of workloads, the two most common approaches have been to replay job traces directly, (e.g. [23, 42, 51]), or else to create statistical models based on job traces ([34, 43]). Models facilitate the creation of multiple similar workloads, potentially with controlled variations such as different loads, but they suffer from not necessarily including all the important features of the real workload [24, 2]. Resampling improves the representativeness of evaluation workloads by modeling only the parts that need to be manipulated, and using real data to fill in the remaining details [68].

Resampling is a powerful technique for statistical reasoning in such situations, when not enough empirical data is available [20, 21]. The idea is to use the available data sample as an approximation of the underlying population, and resample from it. This enables multiple, quasi-independent samples to be created, which are then used to compute confidence intervals or other metrics of interest that depend on the unknown underlying distribution.

We note that while we believe such resampling to be relatively novel in the context of computer workloads and performance evaluation, analogies from other fields of computer science do exist. One analogy comes from computer graphics, where texture mapping is often done by replicating a small patch of texture, with certain variations to give an impression of perspective, conform to lighting conditions, and avoid an obvious tiling effect [39]. More relevant to our work on workloads, such replication, modification, and patching together has also been done for temporal signals, such as movement specification [37] and sound [18]. Another analogy comes from the joint time-space analysis of video. Here the idea is to partition a video into patches, and then replace certain patches with others, e.g. to reconstruct missing frames or add or remove objects [65]. This technique can also be used for anomaly detection: if a piece of a new video cannot be reconstructed from snippets that exist in the system's database, then it is anomalous [8].

To the best of our knowledge resampling-based workload manipulations as we propose here have been used only in few isolated cases, and that in a very limited manner. The closest related work we know of is the Tmix tool, used for the generation of networking traffic. This tool extracts communication vectors describing different connections from a traffic log (sequences of $\langle \text{requestBytes}, \text{responseBytes}, \text{thinkTime} \rangle$) and then replays them subject to feedback from the simulated system's performance [63]. A subsequent paper also mentions the possibility of resampling traces to create diverse load conditions [32], but their approach is simpler than ours as they do not use the concept of sessions nor retain phenomena like the daily cycle.

A similar construction was proposed by Krishnamurthy et al. in the context of evaluating e-commerce systems [38]. In this case they reuse sequences of user operations in order to ensure that illegal sequences are not generated by mistake by the workload modeling procedure. In the domain of parallel job scheduling, Ernemann et al. resize and replicate jobs in order to make a trace suitable for simulations with a larger machine [22]. Our goal, in contrast, is to use the resampled traces to perform better and more comprehensive evaluations. Kamath et al. have suggested to merge several traces and simulate a queueing mechanism in order to increase load [35]. However, this is limited to load values that are the sums of loads from existing traces. Ebling and Satyanarayanan created micro-models of application file behaviors based on a trace, and then combined them stochastically to create test workloads [19]. Again this is similar in concept; the difference from our work is that we use snippets of the traced data directly as the elements of workload being resampled, whereas they create models that risk losing important details. Finally, Chen et al. use a sequence of short samples of MapReduce workloads to reduce the volume of a large workload [15, 14]. This sort of sub-sampling makes no attempt to mimic the processes that generate the workload, and may destroy internal structures, especially if the sample lengths are too short.

There are fields in which timestamps are meaningless when replaying them and therefore they tend to use Time Independent Traces (TIT) files. An initial trace is collected with timestamps and later such timestamps are removed from the original trace. The resulting TIT files are replayed on top of another system configuration with different computing and network models. This technique is mostly used to evaluate MPI applicaions on new hypothetical HPC machines, For example in [11]. However, this approach is much more aggressive comparing to ours. while it allows to completely control the jobs/events that are injected in the simulation-system, it does not attempt to preserve the users' characteristics such as sessions, think-times and daily and weekly cycles.

The simulations typically follow an open systems model, where jobs are submitted by some external population of users. Therefore job arrivals are independent of system performance and state. But a closed (or partially closed) system model with feedback may be more realistic [27, 54, 52, 50, 38], as poor performance may delay the submittal of additional jobs until previous ones have terminated, and perhaps even discourage users and cause them to submit fewer additional jobs. According to Spink, "feedback involves a closed loop of causal influences" [59]. Such effects have been discussed in several different areas. In a database context, Hsu et al. claim that replaying timestamps from a trace loses feedback [33]. Ganger and Patt recognize the influence of the lack of feedback in simulations of storage subsystems as part of a larger system, and suggested giving higher priority to critical requests even if this degrades the performance of the storage system by itself [28]. In evaluations of networks, the feedback is important in shaping the traffic. For example, TCP congestion control is highly dependent on current conditions, so using traced timestamps for packets in simulations is wrong [66, 27]. Also, several papers dealt with a human user's feedback effects on the performance of applications [46, 60, 50, 67]. For example, Yang and Veciana modeled users that aborted their downloads due to poor performance. Synchronous protocols were shown to naturally throttle load due to feedback [53, 1]. Finally, in cloud platforms Chen et al. [13, 3] suggest a new simulation methodology that enforces dependency preservation in the users' workflows, which is predefined and preserving it is essential in order to produce the required results.

In order to include feedback in evaluations one needs a model of how users react to load. While direct experimental evidence is rare [17], some works have considered user tolerance of delays and bandwidth limitations (e.g. [16, 46, 60, 67]). Shmueli used synthetic workloads with a feedback model and a user-aware scheduler designed to exploit this feedback effect [54, 55]. Our work extends those results in two significant ways. First, we show how to conduct more realistic simulations (TBUOS) using both feedback and resampling. This retains all the structure that exists in workload traces, and avoids any potential over-simplifications that may exist in synthetic workloads. Second, we propose a user-aware scheduler (UPS) that is not based directly on the feedback model being used. This generalizes the results and eliminates the risk that they depend on prior knowledge. The UPS scheduler, based on EASY, has a similar basic structure to Shmueli's CREASY scheduler [56], but introduces user-based prioritization.

# Chapter 2

# On Identifying User Session Boundaries in Parallel Workload Logs

**Published**. In Job Scheduling Strategies for Parallel Processing, pp. 216-234, Springer-Verlag, 2012. Lecture Notes in Computer Science Vol. 7698.

Sections:

# On Identifying User Session Boundaries in Parallel Workload Logs

Netanel Zakay Dror G. Feitelson

School of Computer Science and Engineering
The Hebrew University of Jerusalem
91904 Jerusalem, Israel

**Abstract.** The stream of jobs submitted to a parallel supercomputer is actually the interleaving of many streams from different users, each of which is composed of sessions. Identifying and characterizing the sessions is important in the context of workload modeling, especially if a user-based workload model is considered. Traditionally, sessions have been delimited by long think times, that is, by intervals of more than, say, 20 minutes from the termination of one job to the submittal of the next job. We show that such a definition is problematic in this context, because jobs may be extremely long. As a result of including each job's execution in the session, we may get unrealistically long sessions, and indeed, users most probably do not always stay connected and wait for the termination of long jobs. We therefore suggest that sessions be identified based on proven user activity, namely the submittal of new jobs, regardless of how long they run.

## 1 Introduction

There has recently been increased interest in user-based workload models for parallel supercomputers [16, 11–13]. Such models are generative in nature. This means that instead of modeling the statistical properties of the workload, as was done for example by Jann et al. and Lublin and Feitelson [3, 5], they model the process by which the workload is generated. As jobs are submitted by users, this implies the need to model user behavior.

The motivation for using generative user-based workload models is that such models enable us to include feedback effects in performance evaluations. The stream of jobs submitted to a parallel supercomputer is the result of an interaction between the system and its users. If the system is responsive, users will submit more jobs. If it performs poorly, users may depart in frustration and refrain from submitting more jobs. When we evaluate the performance of a new scheduler design, we need to include such feedback and its effect on user behavior [11, 13].

An important characteristic of user behavior is its temporal pattern. Human users may work for some time, but then they stop and do something else. The periods of continuous work are called *sessions*. There are usually many more user sessions during the day than during the night or weekend, leading to the creation of an overall daily and weekly cycle of activity. Understanding how such patterns are generated is a basic component in defining a generative workload model.

Data about user behavior is contained in accounting logs from existing parallel machines, such as those that are available in the Parallel Workloads Archive [9]. Unfortunately, these logs only include data about individual jobs: when the job was submitted, when it started to run and when it terminated, how many processors it used, etc. Importantly, we usually also know the identity of the user who submitted the job (or at least anonymized identity, in the interest of preserving privacy). But we do not know when the user started or ended each session. If we want to characterize this behavior, we need to glean this data based on the pattern of job submissions.

The common approach to extracting session data is based on the assumption that users typically wait to see the results of their jobs, and then submit additional jobs. Thus the user session extends from the submittal of some job till the termination of that or some later job. Zilber at al. have suggested that breaks of 20 minutes or more between successive jobs indicate a session break [16], and others have followed this definition [12].

The problem with this definition is that parallel jobs may be very long. In some logs we even observe jobs that run for multiple days. Obviously it is unreasonable to expect the user to remain active for such a long time waiting for the job to terminate. And indeed we find that sessions defined according to the above definition may be much longer than is reasonable. We therefore suggest an alternative approach, whereby sessions are defined based on only explicit user activity, namely the submittal of new jobs. The times at which the jobs terminate are ignored.

A basic problem with this line of research is that ground truth is not available for comparison. In other words, we do not really know when users started or ended their sessions. We therefore need to make qualitative judgments. Our main criterion is to look at the distribution of session lengths that is generated by the analysis, and to reject methods that lead to distributions with obvious deficiencies (such as sessions that extend over more than a week).

The next section describes the technical details of how sessions may be defined according to different approaches. Section 3 discusses the selection of threshold values used to identify session breaks. Section 4 shows how we can use the distribution of generated sessions to select among two competing approaches for how to apply the threshold. Section 5 identifies some problems that occur when using inter-arrival times rather than think times. Finally, Section 6 introduces the notion of using the generated session lengths as a criterion for accepting or rejecting different approaches.

## 2 Definition of Sessions and Batches

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 parallel supercomputer job logs shows that this is clearly not always the case, and jobs may overlap. Given such an overlap, the later job cannot depend on the earlier one. Following Shmueli, we call a set of such independent, overlapping jobs a *batch* [11]. 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*, or TT.

Finding the batches and the sessions of the users is a basic requirement in order to understand and analyze their behavior. However, activity logs do not contain explicit information about neither the sessions nor the batches. Therefore, we need to estimate them based on the data that the logs do contain. The most relevant information is the job arrival times (also called submit times) and the job end times. For job $i$, we will denote these as $J[i].arr$ and $J[i].end$.

## 2.1 Definitions Based on Think Times

Assume we scan the jobs in a log one by one. As each job is considered, the question is whether it belongs to the previous session or batch, or starts a new session or batch. The simplest and most commonly used approach makes this decision based on the think time, namely the interval from the termination of one job to the submittal of the next[1]:

1. If the think time is negative, the job overlaps the current batch and therefore belongs to this batch.
2. If the think time is positive but below the *session threshold*, the job starts a new batch in the same session as the previous batch. (We discuss the value of the session threshold in Section 3.)
3. Otherwise, the job starts a new batch in a new session.

Note, however, that we need to be precise regarding how we measure the think time, and in particular, exactly what job end time do we use as a reference point. There are two possibilities:

– The end time of the last job that was submitted. With this approach, the think time of job $i$ will be calculated as

$$TT_{Last} = J[i].arr - J[i-1].end$$

Hereafter we denote this approach by Last.

– The maximal end time among all previous jobs. In this case, the think time is calculated as

$$TT_{Max} = J[i].arr - \max_{j<i} J[j].end$$

This approach will be denoted by Max.

[1] Recall that the conceptual model is that the user submits a job, waits for it to terminate, and then thinks about the result before submitting the next job.

To appreciate the difference, consider a sequence of 3 jobs. Job 1 is very long. Job 2 is short and ends much before job 1 ends. Job 3 arrives after job 2 ended, but still overlaps job 1. In this situation all 3 jobs will be in the same batch based on Max, but job 3 will start a new batch based on Last. This is illustrated in Figure 1.

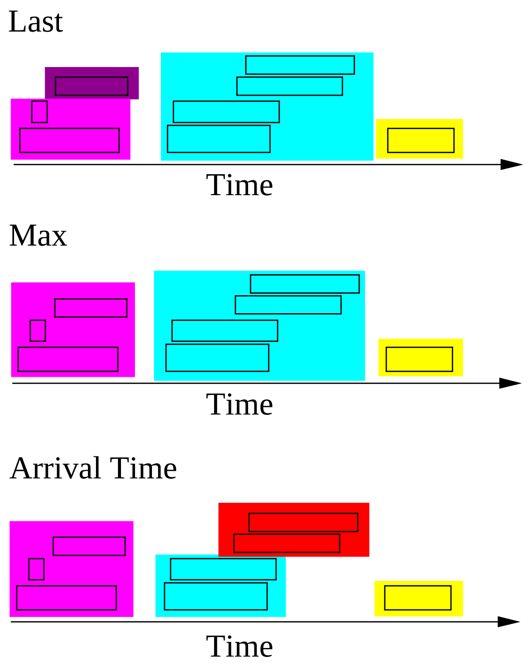


**Fig. 1.** Batches according to the three approaches: Last, Max, and Arrival.

### 2.2 Definition Based on Inter-Arrival Times

Another approach to define sessions is according to the arrival times of the jobs, or rather, the inter-arrivals (to be denoted by Arrival). In this approach, the current job would belong to the same session as previous jobs if it arrives up to the session threshold time after the arrival of the previous job in the session. In other words, if the inter-arrival time is longer than the session threshold, we decide that this represents a session break. Once the jobs are partitioned into sessions using this approach, we partition each such session into batches according to the Max approach.

An example showing the effect of this procedure is shown in Figure 1. The four jobs in the middle all overlap, and are considered to be the same batch by both Last and Max. But there is a relatively large gap between the arrival of the first pair and the arrival of the second pair. If this gap is bigger than the session threshold, the two pairs will be in different sessions according to Arrival, and as a result also in different batches.

The reason for using Max to partition a session into batches is as follows. Consider how the end of a batch is defined. If batch A comes a certain TT after batch B according to Max, then it will start only TT time after *all* the jobs in B are finished. But according to Last, it will start TT time after the last job in B has finished, while other jobs from B may still be running. Shmueli indeed used the last job as the critical one [13]. However, this definition is problematic, because it means that the future activity of the user depends only on the last job in each batch, while the other jobs don't effect the future activity at all. This seems very unrealistic. A simple example of the problem is that it is very easy to create a scheduler that reduces both the user's wait-times and the overall system utilization by running the last job of each user last, thereby causing the user to wait a long time before submitting more jobs. Alternatively, one can construct a scheduler that would increase both the wait-times and the utilization by handling the last job of each user first. To avoid such problems, we prefer Max.

We note that Max creates a sequence of batches with no overlaps. In Last they may overlap, but the dependencies between batches are still a linear sequence. In Arrival a batch may depend on multiple earlier batches.

In the area of parallel supercomputer workloads, the common way to define sessions uses the end time (meaning Last or Max). For example Zilber et al. and Shmueli use Last [16, 12]. But in other areas, where job durations are extremely short, it is more common to define sessions based on arrivals. An example is interactive web use (surfing, searching, or e-commerce) [1, 2, 4, 7, 8, 10, 15]. Of course, due to the very short time it typically takes to process a request on the web, requests never overlap. Therefore Last, Max, and Arrival are actually equivalent in this case.

In the next sections we will discuss the session threshold for each approach and the influence of the choice of this unique value. Additionally, we will present a comparison between the Max and Last approaches. Later, we will investigate the session lengths produced by the different approaches, and conclude that Arrival is the best approach to use.

## 3 Selecting a Session Threshold Value

The dominant methodology to extract session data from activity logs is to postulate a certain threshold value, and assume that breaks in activity which are longer than this threshold represent a division between separate sessions. Such a threshold exists in all three approaches: Last, Max, and Arrival. The main difference between these definitions is the time interval that we compare to the threshold. In Arrival this interval starts at the arrival of the last job, in Last at the end of the last job, and in Max at the maximal end time among previous jobs. The threshold value that is chosen may have a strong effect on the resulting session properties [1]. In this section we will consider how to select the threshold value for each approach, and consider its influence on the sessions.

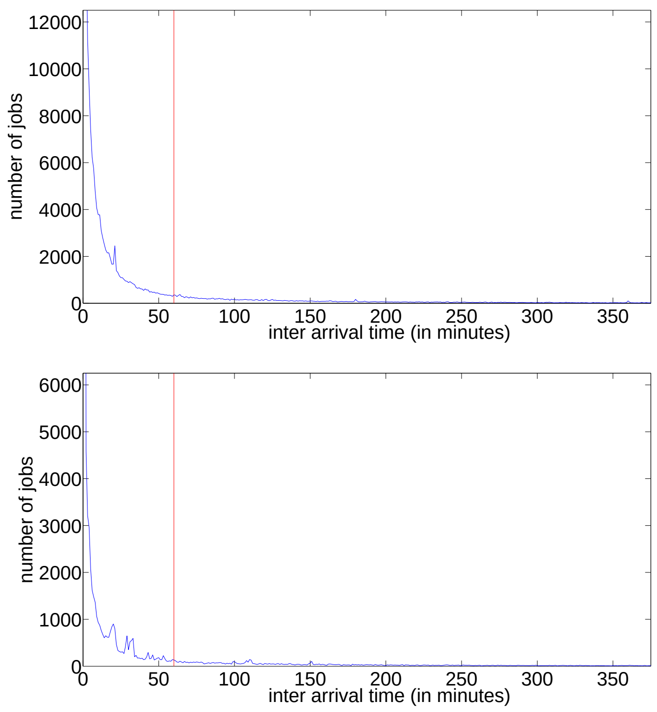


**Fig. 2.** The distribution of inter-arrival time in the SDSC-BLUE and SDSC-DS logs.

As mentioned above, Last and Max are both popular approaches in this area. Therefore, many previous works have considered the selection of the the threshold value for them. The commonly used value is 20 minutes, because this seems to capture the majority of think times. For example, Zilber et al. and Shmueli [16, 12] used this threshold value.

As far as we know, there has been no previous work concerning the selection of a threshold on inter-arrival times for parallel workloads. Several different values have been used in the context of web workloads, including 30 minutes [2, 7], an hour [14], and even two hours [8]. To find what value would make a suitable threshold for our parallel workloads, we calculated the distribution of inter-arrival times for different logs available from the Parallel Workloads Archive [9]. Thus, for each user we found the difference between the arrival times of each pair of successive jobs. We ignored values that were above a day (1440 minutes), because such long intervals obviously defy the notion of a single session. Examples of the resulting distributions are shown in Figure 2. CDFs[2]

[2] The Cumulative Distribution Function (CDF) is the integral of the probability density function (pdf). For each value $x$, it gives the probability of observing values that are smaller than or equal to $x$. In the case of empirical data, it is the fraction of samples that are smaller than or equal to $x$.

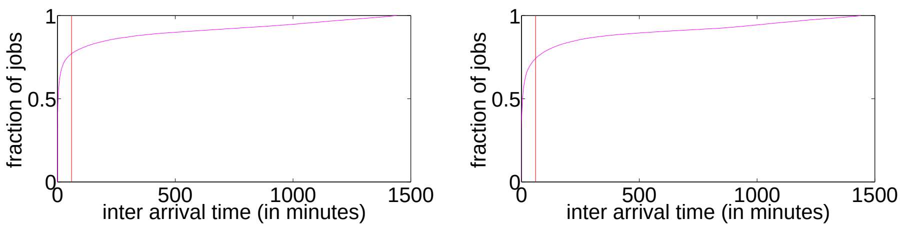


**Fig. 3.** CDFs of inter-arrival times in the SDSC-SP2 and KTH-SP2 logs.

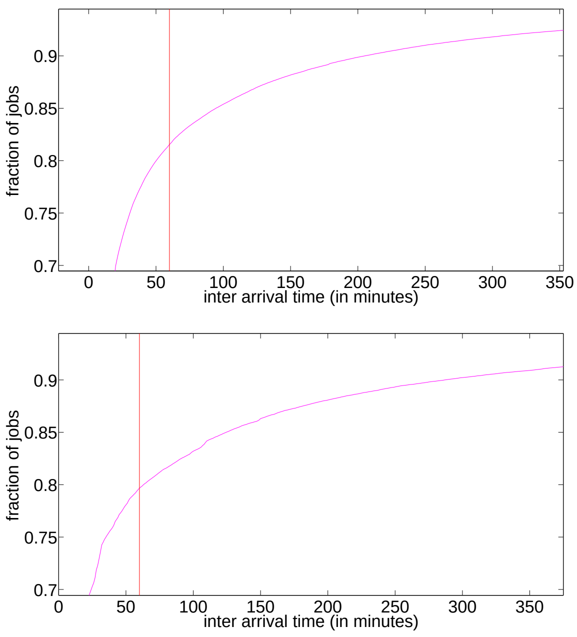


**Fig. 4.** Zoom in on the CDFs of inter-arrival times for the SDSC-BLUE and SDSC-DS logs.

are shown in Figure 3 and Figure 4. In all these figures we added a vertical line at 60 minutes (1 hour), which is the threshold we eventually chose.

Our goal was to find the point in the distribution where the derivative doesn't changed much any more. From Figure 2 it appears that any value between approximately 25 minutes and 200 minutes will be logical. However, for values below 40 one can still observe an obvious drop in the distribution. This is even clearer in the CDF (and especially in the figures with zoom in). In the range of 100 to 200 minutes the slope is already very low, and therefore we would prefer a lower value for the threshold. We concluded that the value ought to be between 40 minutes to 100 minutes. We chose 60 minutes as it is in the middle of this range and is a round value (one hour). We do not claim this is necessarily the best value, but it seems that there is no other value that is obviously better.

Selecting a session threshold has a strong impact on the resulting analysis. If we were to select a higher threshold, jobs with longer intervals will nevertheless be grouped together. As a result the number of jobs in each session would grow and the number of sessions would decrease. In the following sections we provide an in-depth analysis of the implications of the selected session threshold values, mainly in terms of the distribution of session lengths.

## 4 Comparing Last and Max Using the Think-Time Distribution

As we mentioned above, the most common definition of sessions is based on think times, using Last or Max. In this section we investigate which definition leads to a more reasonable think time distribution. The problem is that we do not know what the think time distribution should be. We circumvent this problem as follows. First, we identify the batches according to both approaches separately, and calculate the think times. Then we create a list that contains the common batches (batches that are exactly the same according to both approaches). Based on this list, we extract the think times following these common batches. This provides us with two lists of think times: *CommonTTMax* and *CommonTTLast*. Note that the common batch think times may be different because the think times are defined differently in each approach. In Last, the think time is measured from the end of the last job, whereas in Max it is the maximal end time of all jobs. But we expect the distributions to be close, which indeed they are.

Given the agreement on the common batches, and the similarity of their think time distributions, we take this to represent the "real" distribution of think times. The remaining think times, that we didn't put in the common lists, represent the differences between think times of batches that were created according to Last and Max. Therefore, we would prefer the approach for which the distribution of unique think times is similar to the distribution of common think times.

The resulting distributions are shown in Figure 5. The first obvious conclusion from the graphs is that our expectation that the distributions for common batches shared by Max and Last would be very close to one another was correct. It is also quite clear that the distribution for unique batches as identified by Last is much closer to the common distributions than the distribution for unique batches as defined by Max. It is true that the distribution for Max is closer for larger values, but in most of the range, Last is a lot closer. We concluded that Last creates a more realistic distribution of think times. This supports the use of Last by Zilber et al. [16], Shmueli [12], and others.

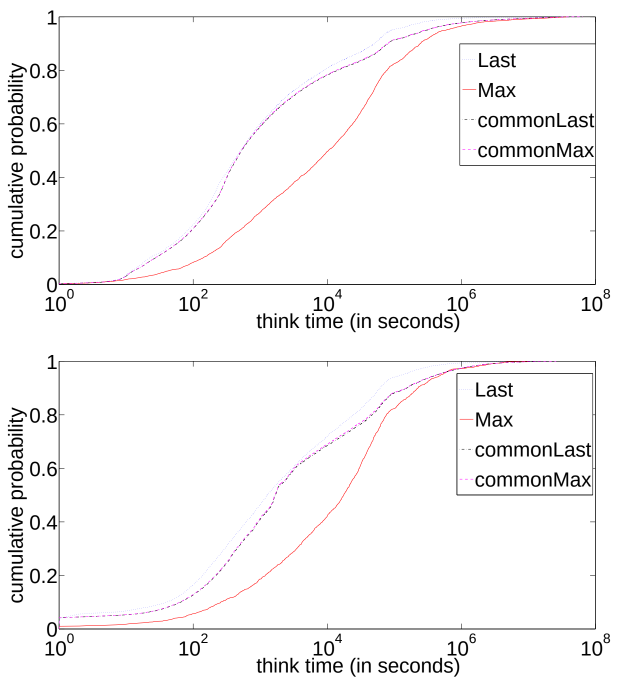


**Fig. 5.** Comparison between distributions of think times in the SDSC-BLUE and SDSC-DS logs.

## 5 Artifacts in the Distribution of Session Lengths with Arrival

According to the work of Mehrzadi, using the Arrival approach may lead to artifacts in the distribution of session durations [6]. Specifically, he shows that in web search data the distribution of session lengths exhibits a pronounced drop exactly at the threshold value that was used to define the sessions. In order to check this, we examined the distribution of session durations for each of the three approaches. Due to the large number of very short sessions, we ignore sessions of up to 2 minutes. The results are shown in Figure Figure 6 for Last, Figure 7 for Max, and 8 for Arrival.

Upon examination of the graphs, we found that Last and Max behave very similarly, but Arrival is indeed different. According to the Max and Last approaches, the distribution of session lengths is essentially the same for different threshold values. The difference in heights is due to the fact that larger thresholds lead to a smaller number of sessions, but the behavior of each graph is the same. In addition, for all values, there are no obvious discontinuities.

In contrast, with the Arrival approach it is easy to notice a sharp drop in the distribution at the threshold value, exactly as had occurred in Mehrzadi's

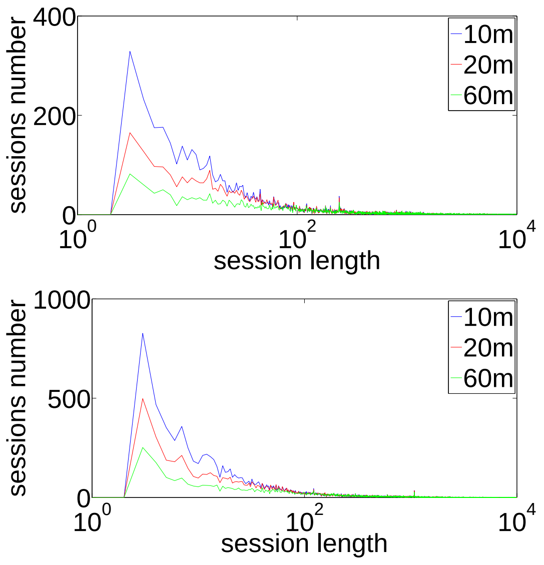


**Fig. 6.** Distribution of session lengths as created by Last, for the KTH-SP2 and SDSC-SP2 logs.

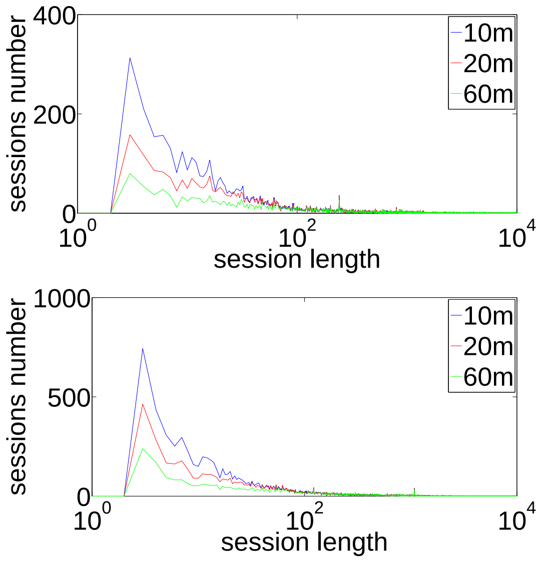


**Fig. 7.** Distribution of session lengths as created by Max, for the KTH-SP2 and SDSC-SP2 logs.

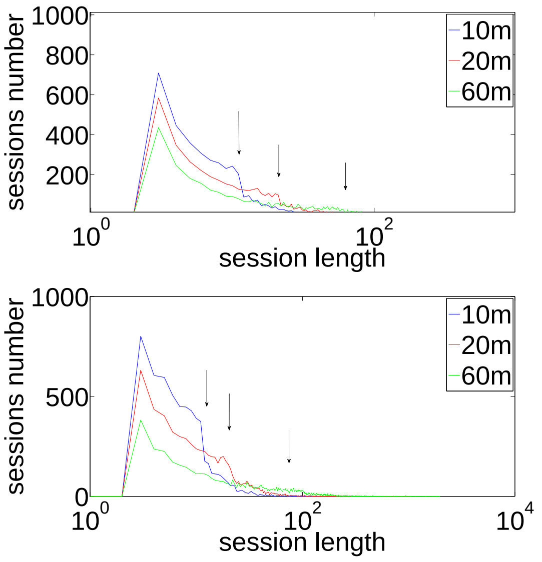


**Fig. 8.** Distribution of session lengths as created by Arrival, for the KTH-SP2 and SDSC-SP2 logs. Arrows denote threshold values.

data. In particular, each specific threshold value changes the distribution of session lengths in a different way (see arrows). However, this effect is reduced when we increase the value of the threshold. This is more evident in Figure 9. In this figure we ignored all session lengths below 9 minutes, and present the histogram without any connecting lines for clarity. With a 10 minute threshold the discontinuity is very significant. For 20 minutes the discontinuity is smaller (but still noticeable). With 60 minutes the drop becomes a step. It is worth mentioning that in some logs (although not in most) there is a clearer drop for 60 minutes, yet rather less dramatic than for 20.

In conclusion, we find that the Arrival approach is sensitive to the threshold value, although with large values (like the one we chose) the effect is rather small.

## 6 The Problem of Very Long Sessions

The graphs in Figures 6 and 7 show that with Last and Max most sessions are short, and few sessions are very long, possibly unrealistically so. However, it is impossible to see the details. In order to emphasize the long sessions we calculated the survival function[3], and present the results in Figure 10. This shows that when using the Last approach, approximately 13.5% of the session

[3] The survival function is the complement to the CDF: for each value $x$, it gives the probability of observing values that are larger than $x$.

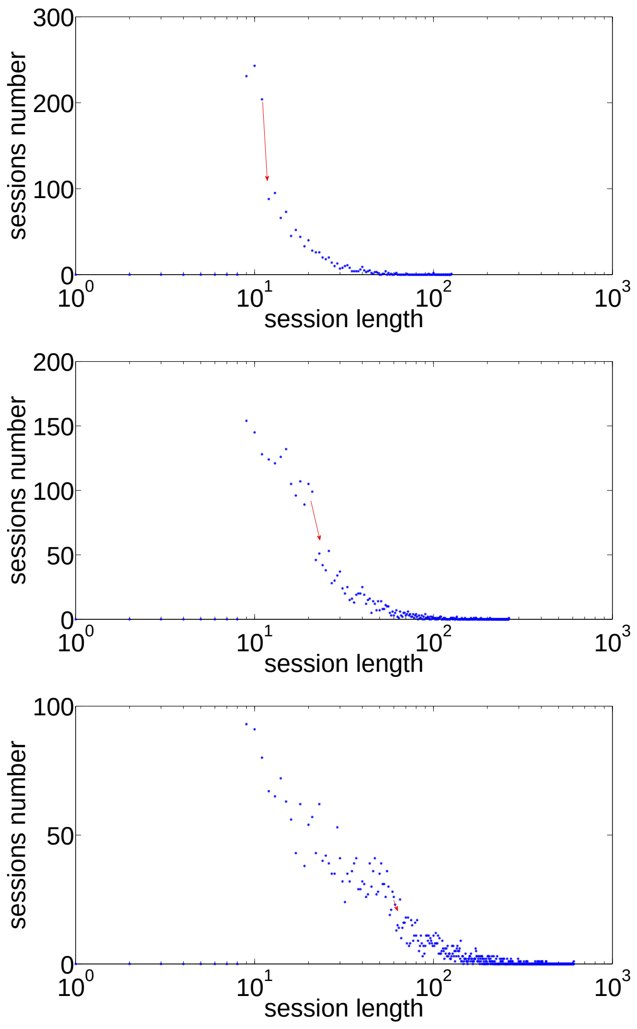
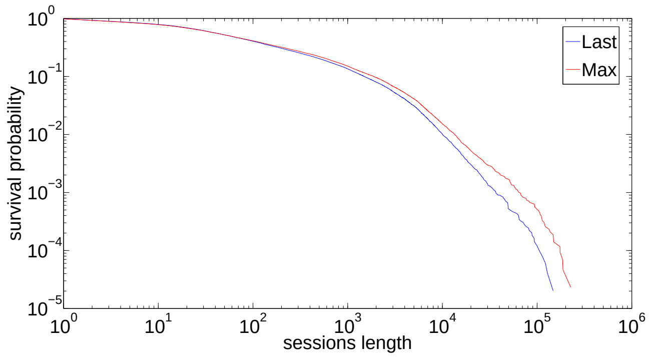


**Fig. 9.** Detailed view of discontinuities in the histogram of session lengths, using the Arrival approach, with thresholds of 10m (top), 20m (middle), and 60m (bottom), for the KTH-SP2 log.

lengths are longer than $10^3$ minutes (16 hours) in the SDSC-BLUE log, and 17.7% are longer than this value in the SDSC-DS log. With the Max approach, the percentages are a little higher: 15% in SDSC-BLUE and 19.5% in SDSC-SP2. In addition, one may notice that the maximum session length is above $10^5$ minutes (approximately 70 days) in both logs.

Recall that a session is supposed to represent the time period when the user is active at the computer (the interval from when the user begins to work and until



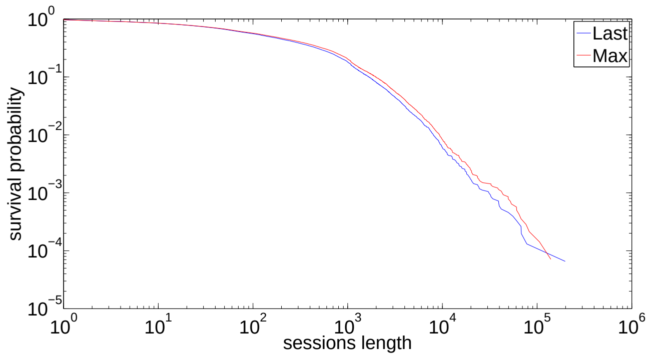


**Fig. 10.** The survival function of session lengths in the SDSC-BLUE and SDSC-SP2 logs, using log-log axes.

he is done working). Therefore, session lengths above $10^4$ minutes are impossible. In addition, even sessions of 16 hours are not reasonable. It should be very rare that a user would work this long continuously, yet still the results show that more than 13% of the sessions are that long. This is very unlikely.

The reason both approaches create sessions that are far too long so many times is that both are based on a wrong assumption. This is the assumption that all work is interactive. With interactive work, it is reasonable to assume that the users wait for the termination of each job, think for a while, and then send the next jobs. But on parallel supercomputers at least some of the work is not interactive. In particular, this is the case for very long jobs that run for many hours or even days. Including these very long jobs within the session, as is done by both Last and Max, then leads to unrealistically long sessions. For example, if a user sends out a job that takes 5 days, and after 3 days sends another job to the system, both approaches will put these two jobs into the same session, although the user most probably wasn't active in the system this whole time. The same problem may also occur on a smaller scale of a hew hours. If there are jobs that run during a break in the user's activity in the middle of the day (for example, during meetings or lunch), these jobs may overlap new work done after the user returns. Therefore, instead of a few short sessions of a couple of hours scattered along the day, we would get one long session — from the first job the user submits in the morning until after he goes home at night.

In order to avoid such problems, we suggest alternative versions of Last and Max which we call Last+Cut and Max+Cut. In these versions we define a new threshold value, called the *Cut*. Then, we use each job's arrival time plus Cut as its effective end time, instead of using the real end time, provided it is shorter. This means that if a job ends within Cut time from its arrival, we measure the think time from its end time without change. Otherwise, we use its arrival time + Cut as the start of the think time. Assuming a session threshold of $T$ minutes we then have:

– Last+Cut: A job will belong to the current batch if it arrives before the arrival time of the last job + Cut + session-threshold (or the end time + session-threshold):

$$J[i].arr \leq \min\{\ J[i-1].end,\ J[i-1].arr + Cut\ \} + T$$

– Max+Cut: A job will belong to the current batch if it arrives before the maximum of the arrival times of all the jobs in the batch + Cut + session-threshold (or with end times):

$$J[i].arr \leq \max_{j<i}[\ \min\{\ J[j].end,\ J[j].arr + Cut\ \}\ ] + T$$

The results of using these approaches are shown in Figure 11. We checked three different values for Cut: 30 minutes, 1 hour, and 2 hours. (Last, Max, and Arrival are also included for comparison.) As expected, in all 3 cases the problem of overly long sessions is largely eliminated. Also, the difference between Max+Cut and Last+Cut with the same threshold is very marginal. Therefore we will distinguish between the Cut approaches only according to the threshold. In the SDSC-BLUE log, the fraction of sessions longer than $10^3$ is a little more than $10^{-3}$ with a large Cut value of 2 hours, but with 1 hour or 30 minutes this fraction is only a little higher than $10^{-4}$ (approximately $10^{-3.9}$). In the SDSC-SP2 log, this fraction is approximately $10^{-3.4}$ with a 2 hours Cut, $10^{-3.7}$ with 1 hour, and $10^{-3.9}$ with 30 minutes. The value of the maximum session length is also dramatically decreased in the Cut approaches: down to 2600 minutes (43 hours) in the SDSC-BLUE log and less than 2000 minutes (33 hours) in the SDSC-SP2 log.

The conclusion is that the Cut approach creates more realistic session lengths. The longest sessions still seem to be too long, lasting nearly 2 days, but still this is much better than the sessions that last for more than 2 months we had before. While unreasonable for humans, such long sessions may be due to a short script or a number of people who might have replaced each other on the computer, sending the jobs through the same user name. In addition, the percentage of long sessions has dropped. Only a very small percentage of the sessions were more than 1000 minutes (16 hours) long, in comparison to 13% or more with the original Last and Max.

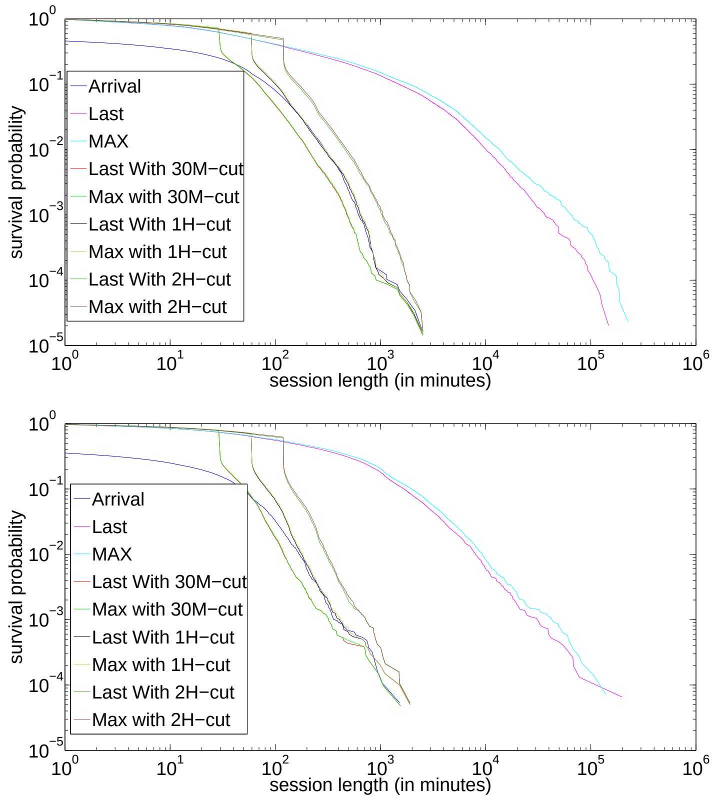


**Fig. 11.** The survival function of session lengths according to all the different approaches, for the SDSC-BLUE log and the SDSC-SP2 log.

However, although the length of the sessions in the Cut approaches are more realistic, the effect of the Cut value on the distribution is enormous: There is a very sharp drop in the graph at the point of the Cut value. In order to examine this effect, we created histograms of the session lengths generated by Last+Cut and Max+Cut. These are presented in Figure 12. It is easy to see that the Cut values produce a very significant mode in the distributions. The reason for these modes is as follows. For all the sessions with one job, if the job ends before the Cut value, the length will be the end time minus the arrival time. This part of the distribution will be continuous. But if the job ends after the Cut Value, the length will be the equal to the Cut value. Therefore, many sessions will receive the Cut value length.

The bottom line is that Last and Max remain problematic. In the original version, they create sessions that are way too long. Introducing the Cut heuristic leads to a strong artifact in the distributions of session lengths. Hence, the only logical approach is to use the Arrival approach. It is equivalent to the Cut ap-

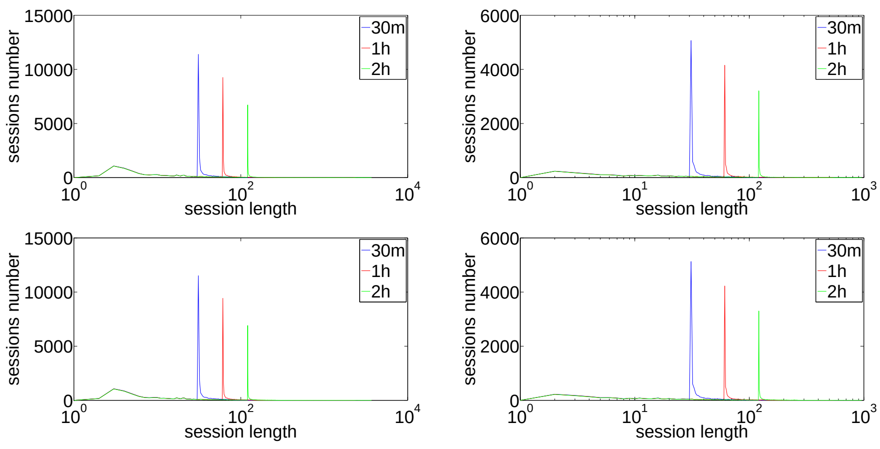


**Fig. 12.** Histograms of session lengths generated by Last+Cut (top) and Max+Cut (bottom) using the SDSC-DS and KTH-SP2 logs (left and right).

proach, where Cut=0, and with a larger session-threshold (60m instead of 20m). (Note that if the Cut value is 0, then Max+Cut is equivalent to Last+Cut.)

The Arrival approach produces realistic session lengths similar to the Cut approaches, But in addition, the distribution is smooth with no modes that depend on parameter values. Therefore, it seems that this approach creates the most sensible distribution of session lengths. We conclude that the Arrival approach, especially with a relatively long session threshold of 1 hour, is the most promising approach to delimit sessions.

## 7 Results with the Arrival Approach

Due to the fact that it is innovative and uncommon to use the Arrival approach to define sessions in parallel workloads, we present a few details and distributions of sessions and batches.

First, we present the number of jobs, batches, and sessions in Table 1. In all of the logs the ratios are very similar. On average, the number of jobs is a little less than twice the number of batches, and the number of batches is a little less

| Log | Jobs | Batches | Sessions |
|---|---:|---:|---:|
| SDSC-SP2 | 54,051 | 32,614 | 18,730 |
| SDSC-DS | 85,003 | 41,679 | 24,294 |
| KTH-SP2 | 28,489 | 16,488 | 10,303 |
| SDSC-BLUE | 223,407 | 136,460 | 58,311 |

**Table 1.** Number of jobs, batches, and sessions in the main logs.

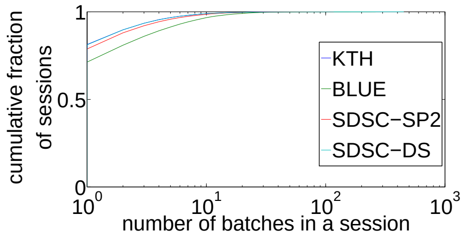


**Fig. 13.** CDF of the number of batches in a session.

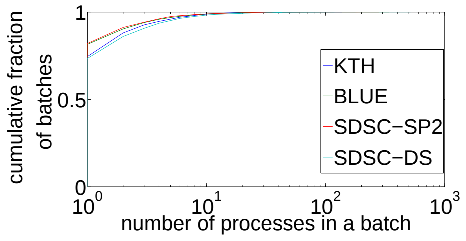


**Fig. 14.** CDF of the number of jobs in a batch.

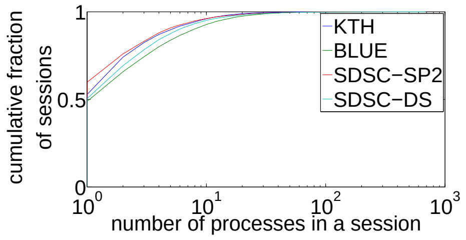


**Fig. 15.** CDF of the number of jobs in a session.

than twice the number of sessions. Additional data on batches and sessions are presented in Figure 13, Figure 14, and Figure 15. A very important observation is that generally more than 50% of the sessions and 75% of the batches contain only one job. This means that when users work with supercomputers, most of the time they send out a single job and then stop their interaction with the computer for a while. However, it is important to note that some sessions have very many jobs, so the distribution is skewed, and most jobs do not constitute single-job sessions.

## 8 Conclusions

A summary of the methods that can be used to identify sessions when analyzing parallel workloads is given in Table 2.

| *Approach* | *Issues* |
|---|---|
| Last, Max | excessively long sessions |
| Last+Cut, Max+Cut | strong peak at cut value |
| Arrival | peak at threshold value<br>many zero-length sessions |

**Table 2.** *Summary of approaches and their effect on the session length distribution.*

The most common approach is to use the Last and Max approaches. These approaches are based on setting a threshold on think times: if the think time is long, this is assumed to be a break between sessions. However, these approaches occasionally cause extremely long sessions, due to the fact that some of the jobs running on such systems are extremely long.

A possible improvement is to use Last+Cut or Max+Cut. This eliminates the very long sessions, at the price of producing a strong peak in the distribution of session length at the value of the cut threshold being used. This is also undesirable.

The alternative is to use the Arrival approach, as in commonly done in other domains, such as the analysis of web workloads. In this approach, inter-arrival times are used. If the inter-arrival is longer than some threshold, a session break is assumed. The main problem with this approach is that long sessions may not be identified correctly, and again a peak in the distribution is created at the value of the threshold being used. However, the size of this peak decreases with increasing threshold values. We suggest to use a threshold of 1 hour. With such a threshold the peak in the distribution of session lengths is very small.

The obvious deficiency with the above is that it is based on common sense, not on data. A desirable avenue for future work is therefore to conduct a user study in which the actual activity patterns of users are followed, and this is correlated with their job submittal patterns.

**Acknowledgments**

Many thanks to all those who have made their workload data available through the Parallel Workloads Archive.

## References

1. M. Arlitt, "*Characterizing web user sessions*". *Performance Evaluation Rev.* **28(2)**, pp. 50–56, Sep 2000.
2. D. Downey, S. Dumais, and E. Horvitz, "*Models of searching and browsing: Languages, studies, and applications*". In 20th *Intl. Joint Conf. Artificial Intelligence*, pp. 1465–1472, Jan 2007.

3. J. Jann, P. Pattnaik, H. Franke, F. Wang, J. Skovira, and J. Riodan, "*Modeling of workload in MPPs*". In *Job Scheduling Strategies for Parallel Processing*, pp. 95–116, Springer Verlag, 1997. Lect. Notes Comput. Sci. vol. 1291.
4. B. J. Jansen, A. Spink, C. Blakely, and S. Koshman, "*Defining a session on web search engines*". *J. Am. Soc. Inf. Sci. & Tech.* **58(6)**, pp. 862–871, Apr 2007.
5. 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.
6. D. Mehrzadi and D. G. Feitelson, "*On extracting session data from activity logs*". In 5th *Intl. Syst. & Storage Conf.*, Jun 2012.
7. D. A. Menascé, V. A. F. Almeida, R. Riedi, F. Ribeiro, R. Fonseca, and W. Meira Jr., "*A hierarchical and multiscale approach to analyze E-business workloads*". *Performance Evaluation* **54(1)**, pp. 33–57, Sep 2003.
8. A. L. Montgomery and C. Faloutsos, "*Identifying web browsing trends and patterns*". *Computer* **34(7)**, pp. 94–95, Jul 2001.
9. "*Parallel workloads archive*". URL http://www.cs.huji.ac.il/labs/parallel/workload/.
10. 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.
11. 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.
12. 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.
13. 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.
14. E. Shriver and M. Hansen, *Search Session Extraction: A User Model of Searching.* Tech. rep., Bell Labs, Jan 2002.
15. C. Silverstein, M. Henzinger, H. Marais, and M. Moricz, "*Analysis of a very large web search engine query log*". *SIGIR Forum* **33(1)**, pp. 6–12, Fall 1999.
16. J. Zilber, O. Amit, and D. Talby, "*What is worth learning from parallel workloads? a user and session based analysis*". In 19th *Intl. Conf. Supercomputing*, pp. 377–386, Jun 2005.

# Chapter 3

# Workload Resampling for Performance Evaluation of Parallel Job Schedulers

**Published**. In Concurrency and Computation – Practice and Experience 26(12), pp. 2079–2105, Aug 2014.

Sections:

**Note: This is an invited extended journal version** of N. Zakay and D. G. Feitelson, Workload resampling for performance evaluation of parallel job schedulers. In 4th International Conference Performance Engineering (ICPE), pp. 149-159, Apr 2013.

CONCURRENCY AND COMPUTATION: PRACTICE AND EXPERIENCE
*Concurrency Computat.: Pract. Exper.* 2013; **00**:1–27
Published online in Wiley InterScience (www.interscience.wiley.com). DOI: 10.1002/cpe

# Workload Resampling for Performance Evaluation of Parallel Job Schedulers

Netanel Zakay Dror G. Feitelson*

*The Rachel and Selim Benin School of Computer Science and Engineering*
*The Hebrew University of Jerusalem, 91904 Jerusalem, Israel*

SUMMARY

Evaluating the performance of a computer system is based on using representative workloads. Common practice is to either use real workload traces to drive simulations, or else to use statistical workload models that are based on such traces. Such models allow various workload attributes to be manipulated, thus providing desirable flexibility, but may lose details of the workload's internal structure. To overcome this, we suggest to combine the benefits of real traces and flexible modeling. Focusing on the problem of evaluating the performance of parallel job schedulers, we partition the trace of submitted jobs into independent subtraces representing different users, and then re-combine them in various ways, while maintaining features like long-range dependence and the daily and weekly cycles of activity. This facilitates the creation of longer workload traces that enable longer simulations, the creation of multiple statistically similar workloads that can be used to gauge confidence intervals, the creation of workloads with different load levels, and increasing the frequency of specific events like large surges of activity. Copyright © 2013 John Wiley & Sons, Ltd.

Received . . .

KEY WORDS: Workload trace; Resampling; Simulation

## 1. INTRODUCTION

The performance of a computer system is affected by the workload it handles. Reliable performance evaluations therefore require the use of representative workloads. This means that the evaluation workload should not only have the same marginal distributions as the workloads that the system will have to handle in production use, but also the same correlations and internal structure. As a result, traces of real workloads are often used to drive simulations of new system designs, because such traces obviously contain all the structure found in real workloads.

Replaying a trace provides only a single data point of performance for one workload. But in many evaluations, several related workloads are needed. For example, in order to compute confidence intervals, one needs multiple instances of the same basic workload. The common way to satisfy this need is to create multiple synthetic workloads based on statistical workload models (which, in turn, are based on the traced data) [23, 2, 31, 39, 43]. While models provide the required variability and flexibility for evaluations, they also suffer from not necessarily including all the important features of the real workload [18, 1] — in fact, they include only those of which the modeler was aware.

To improve the representativeness of evaluation workloads we propose to *combine the realism of real traces with the flexibility of models*. This will be done by modeling only the part of the workload that needs to be manipulated, and resampling from the real data to fill in the remaining

*Correspondence to: feit@cs.huji.ac.il

Table I. Logs from the Parallel Workloads Archive (www.cs.huji.ac.il/labs/parallel/workload/) that were used in this study.

| Log name | File | Period | PEs | Users | Jobs |
|---|---|---|---|---|---|
| LANL-CM5 | LANL-CM5-1994-3.1-cln | 10/94–09/96 | 1024 | 213 | 122,060 |
| SDSC-Par | SDSC-Par-1995-3.1-cln | 12/94–12/95 | 400 | 98 | 53,970 |
| CTC-SP2 | CTC-SP2-1996-2.1-cln | 06/96–05/97 | 338 | 679 | 77,222 |
| KTH-SP2 | KTH-SP2-1996-2 | 09/96–08/97 | 100 | 214 | 28,489 |
| SDSC-SP2 | SDSC-SP2-1998-3.1-cln | 04/98–04/00 | 128 | 437 | 59,725 |
| OSC-cluster | OSC-Clust-2000-3.1-cln | 01/00–11/01 | 178 | 253 | 36,097 |
| SDSC-BLUE | SDSC-BLUE-2000-3.1-cln | 04/00–01/03 | 1152 | 468 | 243,314 |
| HPC2N | HPC2N-2002-1.1-cln | 07/02–01/06 | 240 | 257 | 202,876 |
| SDSC-DS | SDSC-DS-2004-1 | 03/04–04/05 | 1664 | 460 | 96,089 |
| ANL-Intrepid | ANL-Intrepid-2009-1 | 01/09–09/09 | 163,840 | 236 | 68,936 |
| PIK-IPLEX | PIK-IPLEX-2009-1 | 04/09–07/12 | 2560 | 225 | 742,965 |
| CEA-Curie | CEA-Curie-2011-2.1-cln | 02/12–10/12 | 93,312 | 582 | 312,826 |

details. Technically this is done by partitioning workload traces into their basic components and re-grouping them in different ways to achieve the desired effects.

The domain of our work is parallel job scheduling. Parallel systems are increasingly relevant today, with the advent of multi-core processors (parallelism on the desktop), clusters and blade servers (parallelism at the enterprise level), and grids and clouds (parallelism across multiple locations). The jobs that run on parallel systems are composed of multiple processes that need to run on distinct processors (in large clusters and supercomputers the number of processes and processors can be in the thousands). When a job is submitted the user specifies how many processors are needed, and often also for how much time. The scheduler then determines the order in which jobs will be executed, and which processors will be allocated to each one. Accounting logs from large-scale systems are available in the Parallel Workloads Archive [20], and provide data about the workloads they served. In particular, logs typically contain information about the submit time of each job, it's runtime and number of processes, the user who submitted it, and more. These logs can therefore be used to simulate the behavior of new scheduler designs and compare them with each other. The logs we use in this work are listed in Table I. Most of the results that we present in this paper use the more recent and relevant logs. However, two logs (SDSC-Par and PIK-IPLEX) do not have an estimated running time, and are therefore excluded from simulations where it is needed. OSC-cluster is also often skipped due to its very low load.

In the context of parallel job scheduling, we suggest that the resampling be done at the level of users. We first partition the workload into individual subtraces for the different users, including all the jobs submitted by each user throughout the tracing period. We then sample from this pool of users to create a new workload trace. Using such resampling, we can achieve the following:

- Create a much longer trace than the original, and use it to ensure convergence of evaluation results.
- Create multiple similar workloads, and use them to compute confidence intervals.
- Create workloads with higher or lower average loads, by using more or less concurrently active users, and use them to investigate how load affects system performance.
- Create workloads in which rare events such as surges in activity are amplified, and use them to investigate the effect of such events.

Importantly, while the resampled workloads differ from the original in length, statistical variation, or load, they nevertheless retain important elements of the internal structure such as sessions and the relationship between the sessions and the daily work cycle. They are even found to have the same long-range dependence structure.

Workload manipulations are an important tool in the performance analyst's toolbox, that has not received its due attention in terms of methodological research. As a result, inappropriate manipulations are sometimes used, which in turn has led to some controversy regarding whether any manipulations of real workloads are legitimate. By increasing our understanding of resampling-based manipulations we hope to bolster the use of this important tool, allowing new types of manipulations to be applied to workload traces, and enabling researchers to achieve better control over their properties, as needed for different evaluation scenarios.

In the rest of this paper we describe this promising approach to using workload traces and demonstrate its effectiveness. The next section further explains the motivation for using resampling. In Section 3 we consider different resampling granularities, and justify the decision to do so at the level of all the activity of each user. Section 4 explains how the resampling is done in considerable detail, including the proposed distinction between long-term and temporary users, and Section 5 validates the process by showing that the generated workloads have the same statistical properties as the original. Section 6 then demonstrates the use of resampling to achieve the objectives listed above, and also suggests some additional potential uses, and we conclude in Section 7.

This paper extends a previous conference version [47]. The main additions are the verification that resampling retains the self-similarity of the workloads, the use of resampling to over-sample rare events such as flash crowds and evaluate their impact, and the addition of more examples to the experimental results including the use of two new recent workload logs.

## 2. WHY USE RESAMPLING

The Parallel Workloads Archive includes more than 20 workload traces from different systems, but this may not always suffice. Some of the traces may not be appropriate for certain system types (for example, throughput-oriented systems often allow only serial jobs). Some traces are dated and may not represent present practices. Evaluations may require certain attributes that are not available in the archive, e.g. a series of workloads whose loads differ by 5%. Even if one has access to a real system one cannot force the workload on it to conform to a desired configuration.

Resampling is a powerful technique for statistical reasoning in such situations, when not enough empirical data is available [11, 12]. The idea is to use the available data sample as an approximation of the underlying population, and resample from it. This enables multiple, quasi-independent samples to be created, which are then used to compute confidence intervals or other metrics of interest that depend on the unknown underlying distribution.

Our ideas for workload manipulation are analogous to this. We have a workload trace at our disposal. The problem is that this provides a single data point, whereas our evaluation requires the use of several (maybe many) workloads with certain variations. The proposed solution is to partition the given workload into its constituents, and re-group them in different ways to create new workloads. The simplest approach is to partition the workload into its most basic components (e.g. jobs), and resample at random. This is similar to just using the empirical distribution as a model. Our proposal is to extend this in two ways:

1. We consider different definitions of what constitutes the basic elements of the workload. For example, they could be individual jobs, batches of related jobs, complete user sessions, or even the sequence of all the sessions by each user.
2. Resampling may not be random, but guided by some specific manipulation that we want to apply to the workload, and also subject to constraints such as maintaining system stability.

The notion that this is a useful device is our working hypothesis; examples and evidence supporting this notion are given below.

We note that while we believe such resampling to be relatively novel in the context of computer workloads and performance evaluation, analogies from other fields of computer science do exist. One analogy comes from computer graphics, where texture mapping is often done by replicating a small patch of texture, with certain variations to give an impression of perspective, conform to lighting conditions, and avoid an obvious tiling effect [27]. More relevant to our work on workloads, such replication, modification, and patching together has also been done for temporal signals, such as movement specification [25] and sound [9]. Another analogy comes from the joint time-space analysis of video. Here the idea is to partition a video into patches, and then replace certain patches with others, e.g. to reconstruct missing frames or add or remove objects [45]. This technique can also be used for anomaly detection: if a piece of a new video cannot be reconstructed from snippets that exist in the system's database, then it is anomalous [4].

To the best of our knowledge resampling-based workload manipulations as we propose here have been used only in few isolated cases, and that in a very limited manner. The closest related work we know of is the Tmix tool, used for the generation of networking traffic. This tool extracts communication vectors describing different connections from a traffic log (sequences of $\langle \text{requestBytes}, \text{responseBytes}, \text{thinkTime} \rangle$) and then replays them subject to feedback from the simulated system's performance [44]. A subsequent paper also mentions the possibility of resampling traces to create diverse load conditions [22], but their approach is simpler than ours as they do not use the concept of sessions nor retain phenomena like the daily cycle. A similar construction was proposed by Krishnamurthy et al. in the context of evaluating e-commerce systems [26]. In this case they reuse sequences of user operations in order to ensure that illegal sequences are not generated by mistake by the workload modeling procedure. In the domain of parallel job scheduling, Ernemann et al. resize and replicate jobs in order to make a trace suitable for simulations with a larger machine [13]. Our goal, in contrast, is to use the resampled traces to perform better and more comprehensive evaluations. Kamath et al. have suggested to merge several traces and simulate a queueing mechanism in order to increase load [24]. However, this is limited to load values that are the sums of loads from existing traces. Ebling and Satyanarayanan created micro-models of application file behaviors based on a trace, and then combined them stochastically to create test workloads [10]. Again this is similar in concept; the difference from our work is that we use snippets of the traced data directly as the elements of workload being resampled, whereas they create models that risk losing important details. Finally, Chen et al. use a sequence of short samples of MapReduce workloads to reduce the volume of a large workload [7, 6]. This sort of sub-sampling makes no attempt to mimic the processes that generate the workload, and may destroy internal structures, especially if the sample lengths are too short.

## 3. GRANULARITY OF RESAMPLING

Resampling can be done at different levels. In many cases, the coarsest level is the activity of a user, which may be partitioned into sessions. The constituents of a session depend on what sort of work we are looking at. It can be the submittal of parallel jobs, downloads from web servers that are composed of packets being sent over the Internet, or individual accesses to file data.

Resampling at the job level is similar to resampling in statistics, e.g. as applied in the bootstrap method [11], which is similar to using the empirical distribution as a model. Note, however, that by resampling complete jobs we retain the correlations between job attributes (e.g. job size and runtime), which would be lost if we resampled from each marginal distribution independently.

Resampling is all about creating new mixed versions of the workload. But at the same time, we wish to retain at least some of the local structure. Specifically, we typically want to retain the locality properties exhibited by normal work practices. Also, it may be important to retain the structure of batches of related jobs or sessions. For example, this is necessary for the evaluation of adaptive systems that learn about their workload and adapt to changing workload conditions [40, 17]; without locality and structure, such systems don't have what to exploit.

To motivate the use of resampling at the user level, we studied the similarity between each user's jobs. First we divided the work of each user into sessions [46]. Then we analyzed the similarity of jobs in one session with each other and with the jobs in subsequent sessions, using three attributes: the number of processors used, the jobs' runtimes, and their estimated runtimes. The metric for similarity was the ratio of the smaller value to the larger one: $r = \min\{j1.att, j2.att\} / \max\{j1.att, j2.att\}$. This is by definition in the range $[0, 1]$, with 0 indicating a large difference and 1 indicating identity. When comparing two sessions, the similarity is calculated between all pairs of jobs, where one job comes from one session and the other job from the other session. Then we characterize the similarity between the sessions using the average similarity

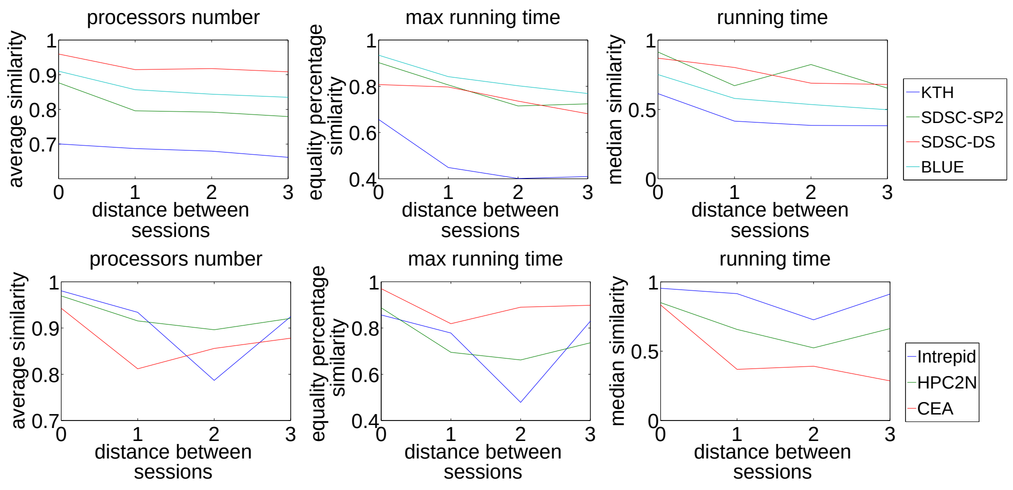


Figure 1. Similarity between jobs as a function of the distance between them in sessions.

Table II. The fraction of users for whom the similarity between jobs was larger at one distance than at another distance, using the CPU-number property.

| | distance 0 vs 1 | | | distance 1 vs 2 | | |
|---|---|---|---|---|---|---|
| Log name | 0 larger | equal | 1 larger | 1 larger | equal | 2 larger |
| LNAL-CM5 | 0.727 | 0.0160 | 0.257 | 0.663 | 0.0053 | 0.332 |
| CTC-SP2 | 0.722 | 0.0152 | 0.262 | 0.670 | 0.0325 | 0.300 |
| KTH-SP2 | 0.782 | 0.0075 | 0.211 | 0.692 | 0.0150 | 0.293 |
| SDSC-SP2 | 0.771 | 0.0120 | 0.217 | 0.747 | 0.0080 | 0.245 |
| SDSC-BLUE | 0.856 | 0.0000 | 0.144 | 0.762 | 0.0046 | 0.233 |
| HPC2N | 0.856 | 0.0046 | 0.140 | 0.684 | 0.0000 | 0.316 |
| SDSC-DS | 0.767 | 0.0031 | 0.230 | 0.702 | 0.0092 | 0.288 |
| ANL-Intrepid | 0.788 | 0.0052 | 0.207 | 0.746 | 0.0000 | 0.254 |
| CEA-Curie | 0.866 | 0.0022 | 0.131 | 0.746 | 0.0043 | 0.250 |

between job pairs, the median similarity similarity between job pairs, or the fraction of job pairs that had identical values.

Fig. 1 shows a sample of the results, using logs available from the Parallel Workloads Archive [34]. The horizontal axis is the distance in sessions between the compared jobs, and the vertical axis shows the average or median of the level of similarity. In all the graphs, jobs in the same session are the most similar to each other. The top row shows cases where the degree of similarity is reduced with distance in a monotonic manner. This is interpreted as reflecting locality, where users perform similar work for some time and then move to do something else. The second row shows cases where the change in similarity is not monotonic. This reflects some other work patterns, where similar jobs are executed again after some time. But in both cases we see a pattern, indicating that the sessions are not independent and that performing the resampling at the session level would lose potentially important information.

Another way to characterize the locality is to find the fraction of users for whom the similarity at a short distance is higher than the similarity at a longer distance. Such data is shown in Tables II to IV, for distances of 0 vs. 1 sessions and 1 vs. 2 sessions. This again demonstrate that the similarity drops when the distance grows, as in most cases about 70–85% of the users exhibit higher similarity at the shorter distance. In effect, this testifies to the existence of locality in these workloads, which we want to retain.

To validate this result, we used bootstrapping to compare the results shown above with results that would be obtained if we sample jobs independently. To do so we retain the structure of sessions

Table III. The fraction of users for whom the similarity between jobs was larger at one distance than at another distance, using the maximum running time property.

| | distance 0 vs 1 | | | distance 1 vs 2 | | |
|---|---|---|---|---|---|---|
| Log name | 0 larger | equal | 1 larger | 1 larger | equal | 2 larger |
| LANL-CM5 | 0.765 | 0.1016 | 0.134 | 0.668 | 0.1016 | 0.230 |
| CTC-SP2 | 0.757 | 0.1041 | 0.139 | 0.705 | 0.1085 | 0.187 |
| KTH-SP2 | 0.895 | 0.0075 | 0.098 | 0.789 | 0.0150 | 0.195 |
| SDSC-SP2 | 0.791 | 0.0924 | 0.116 | 0.795 | 0.0924 | 0.112 |
| SDSC-BLUE | 0.947 | 0.0069 | 0.046 | 0.794 | 0.0069 | 0.199 |
| HPC2N | 0.907 | 0.0279 | 0.065 | 0.781 | 0.0233 | 0.195 |
| SDSC-DS | 0.837 | 0.0613 | 0.101 | 0.739 | 0.0583 | 0.202 |
| ANL-Intrepid | 0.834 | 0.0777 | 0.088 | 0.715 | 0.0777 | 0.207 |
| CEA-Curie | 0.843 | 0.0625 | 0.095 | 0.718 | 0.0625 | 0.220 |

Table IV. The fraction of users for whom the similarity between jobs was larger at one distance than at another distance, using the running time property.

| | distance 0 vs 1 | | | distance 1 vs 2 | | |
|---|---|---|---|---|---|---|
| Log name | 0 larger | equal | 1 larger | 1 larger | equal | 2 larger |
| LANL-CM5 | 0.733 | 0.0000 | 0.267 | 0.711 | 0.0053 | 0.283 |
| CTC-SP2 | 0.829 | 0.0043 | 0.167 | 0.722 | 0.0065 | 0.271 |
| KTH-SP2 | 0.827 | 0.0075 | 0.165 | 0.684 | 0.0000 | 0.316 |
| SDSC-SP2 | 0.759 | 0.0000 | 0.241 | 0.759 | 0.0040 | 0.237 |
| SDSC-BLUE | 0.888 | 0.0114 | 0.101 | 0.757 | 0.0092 | 0.233 |
| HPC2N | 0.879 | 0.0326 | 0.088 | 0.721 | 0.0279 | 0.251 |
| SDSC-DS | 0.840 | 0.0215 | 0.138 | 0.706 | 0.0245 | 0.270 |
| ANL-Intrepid | 0.850 | 0.0000 | 0.150 | 0.710 | 0.0000 | 0.290 |
| CEA-Curie | 0.817 | 0.0151 | 0.168 | 0.685 | 0.0151 | 0.300 |

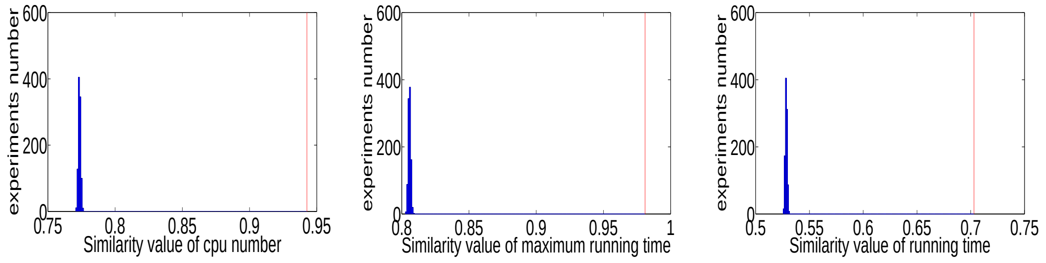


Figure 2. Comparison of the similarity between jobs in the same session as computed from the CEA-Curie log (vertical line), and the distribution of similarity levels that are seen when the jobs are randomized.

for each user, but mix the jobs randomly among the sessions. This is repeated 1000 times, and each time the degrees of similarity between jobs in the same session are computed as above. Fig. 2 shows a sample of the results for one log. Obviously, the similarity among jobs that appear together in the original log is much higher than the similarity observed when jobs are randomized, as would happen if we resample individual jobs.

An implicit assumption in our resampling procedure is that users are independent. This is not strictly valid because users affect each other: if one user overloads the system, others may feel this and reduce their own activity. However, a large part of such interactions is due to all users operating on the same daily cycle, and we take care to retain this correlation between the resampled users. Moreover, resampling at the user level rather than at the session level allows for more sophisticated

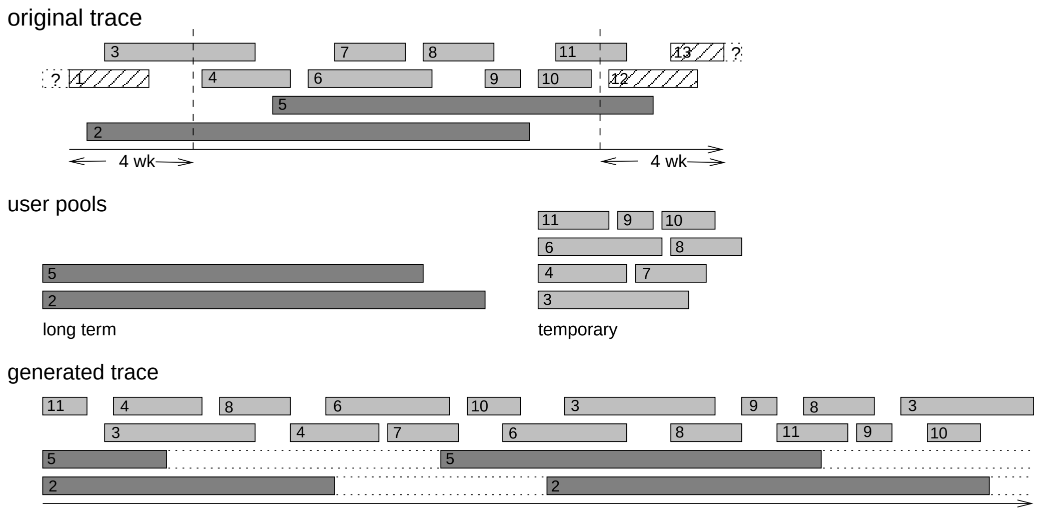


Figure 3. Conceptual framework of dividing users into long-term and temporary, and reusing them in a generated trace.

user behavior models. Specifically, we can introduce feedback effects whereby a user may decide to terminate a session because system performance is poor, and submit his subsequent jobs in a later session. Our work on incorporating such feedback effects will be reported separately; for now the main point is that if we resample at the session granularity such effects will be effectively excluded.

Based on the above considerations, we decided to perform our resampling at the user level, in order to retain the locality in the modified workloads that we produce and allow for the future inclusion of feedback effects.

## 4. MECHANICS OF RESAMPLING

Creating a new workload by resampling users means that we dissect the given trace into sub-traces representing different users, and then recombine these sub-traces in different ways. Note that we do not manipulate each user's sub-trace. Thus the sequence of jobs representing each user will be the same as in the original trace, and the intervals between them will also be the same. This guarantees the same locality properties as in the original trace, as noted above. We also take care to synchronize the resampled users using a common timeframe, so that jobs always start on the same day of the week and the same time of the day as in the original trace. This ensures that the daily cycle of activity is retained in the produced workload, which may be important [48, 18].

An important issue in dissecting a trace into separate users is how to handle end effects. After all, there is no reason to assume that the beginning or end of the tracing period is synchronized in any way with the beginning or end of the activity of any particular user. We approach this problem by making a distinction between temporary users and long-term users (see Fig. 3). This distinction relates to basic aspects of human user work patterns, and is expected to be relevant to other system types too.

Temporary users are all the users that interact with the system for a limited time, for example while conducting a project. These users arrive to the system at a certain point, interact with it for a short while, and are expected to leave shortly after that and never return. Long-term users, in contradistinction, are the users that routinely use the system all the time. These users may be expected to have been active before logging started, and to send more jobs also after the end of the recording period.

In analyzing the log, we distinguish between temporary users and long-term users according to the interval between their first job and their last job in the log. If the interval is long enough (above 12 weeks in our implementation), the user is classified as long-term. Otherwise the classification is

Table V. Results of classifying users in the different logs.

| | long-term | | temporary | |
|---|---|---|---|---|
| Log | users | jobs | users | jobs |
| LANL-CM5 | 159 | 119,998 | 37 | 1,727 |
| SDSC-Par | 57 | 45,087 | 32 | 7,354 |
| CTC-SP2 | 314 | 63,287 | 236 | 10,625 |
| KTH-SP2 | 102 | 25,202 | 66 | 2,349 |
| SDSC-SP2 | 173 | 44,251 | 206 | 8,790 |
| SDSC-BLUE | 426 | 221,745 | 31 | 1,435 |
| HPC2N | 178 | 194,429 | 66 | 7,949 |
| SDSC-DS | 230 | 74,764 | 192 | 9,012 |
| ANL-Intrepid | 124 | 58,875 | 81 | 7,725 |
| PIK-IPLEX | 175 | 724,045 | 46 | 4,764 |
| CEA-Curie | 269 | 244,733 | 223 | 38,814 |

temporary. The threshold of 12 weeks is chosen based on observation of the distribution of periods of activity by different users. We found that for many users their period of activity was up to about 12 weeks; these are the temporary users. For the rest there was a uniform distribution from 12 weeks to the full length of the log. This is interpreted as representing long-term users whose activity was arbitrarily intersected with the logging period. The numbers of temporary and long-term users found in different logs, and the jobs that they submitted, are shown in Table V. There tend to be somewhat more long-term users than temporary ones. As may be expected, the long-term users submit the vast majority of the jobs. Temporary users have in average less jobs and sessions due to their shorter activity. However, parameters that don't depend on the activity length, such as session lengths, are similar for long-term and temporary users.

Data about the different users is kept in separate user pools, one for temporary users and the other for long-term users (Fig. 3). However, temporary users whose full period of activity falls within a short time (4 weeks) from the beginning or the end of the logging period are discarded. The reason for doing so is that there is a high probability that the activity of these users was truncated, but we cannot know for sure. The threshold of 4 weeks is chosen because when plotting the cumulative number of users observed as a function of the number of weeks into the log, in the first few weeks the graph climbs at a higher rate. This is interpreted as being influenced by first observations of users that have already been active before. Then, when the increase settles on a lower and relatively constant average rate, this is interpreted as predominantly representing the arrivals of new users.

Given the pools of temporary and long-term users, the resampling and generation of a new trace is done as follows:

- **Initialization:** We initialize the trace with some temporary users and some long-term users. The numbers of users to use are parameters of the trace generation, and can be used to change the load or the ratio of temporary to long-term users (the defaults are the numbers of long-term users in the original log, and the average number of temporary users present in a single week of the original log). The probability to select each temporary user is proportional to the number of weeks during which the user was active in the log. Users are not started with their first job from the trace, because we are trying to emulate a workload that was recorded over an arbitrary timespan, and there is no reason to assume that the beginning of the logging period should coincide with the beginning of a user's activity. Therefore each user is started in some arbitrary week of his traced activity. However, care is taken that jobs start on the same day of the week and time of the day in the simulation as in the original trace.
- **Temporary users:** In each new week of the simulation, a certain number of new temporary users are added. The exact number is randomized around the target number, which is a parameter of the trace generation (the default is the average rate at which temporary users arrived in the original trace). The randomization uses a binomial distribution, with

a probability $p$ equal to the fraction of temporary users expected to start every week. The selected users are started from their first traced jobs. A user can be selected from the pool multiple times, but care is taken not to select the same user twice in the same week.

- **Long-term users:** The population of long-term users is constant and consists of those chosen in the initialization. When the traced activity of a long-term user is finished, it is simply regenerated after a certain interval. While such repetitions are not very realistic, they allow us to extend the work of the long-term users as needed. We also note that repetitions only occur after rather long intervals, because logs are typically at least a year long. The interval between the regenerations corresponds to the sum of the intervals between the user's period of activity and the full logging period. Naturally the regenerations are also synchronized correctly with the time and day.

Note that this process can go on indefinitely, and indeed one of the applications of workload resampling that we describe in Section 6 is to extend traces and allow for longer simulations.

The exact number of users in the initialization, the week of activity from which they start, the number of temporary users added each week, and the identity of the selected users are all randomized. Therefore our simulation creates a different workload in each run. But all these workloads are based on the same sub-sequences of jobs, and are therefore all statistically similar to each other and to the original trace.

## 5. RESAMPLING VALIDATION

In order to perform resampling and implement the applications described in the next section it is enough to just create a new workload trace that is composed of the jobs of the different users as described above. However, we actually perform a full simulation of also scheduling these jobs. This enables us to directly use the generated workloads to evaluate various parallel job schedulers. In subsequent work we also consider adding feedback, whereby the system performance influences user behavior and may affect when subsequent jobs are submitted [38]. In any case, the simulation also creates a log file which contains the new workload. Comparing this generated workload with the original one allows us to validate the resampling process.

### *5.1. Marginal Distributions*

The validation is based on comparing the generated workloads to the original one. We start with a comparison of various marginal distributions that describe the workload's structure, with a focus on the user level because this is what we modify. An example is shown in Fig. 4 based on the ANL-Intrepid log; similar results are obtained for other logs too. The different panels show the following distributions:

- Number of jobs submitted by different users.
- Number of sessions performed by users.
- Average session length for different users.
- Total amount of CPU time (work) used by users in all their jobs.
- The users' first arrival times.
- The users' final departure times.
- The users' periods of activity.
- The distribution of job arrivals across days of the week, for all users together.

In all but the last of these, the users are first sorted according to the metric, and then the distribution is plotted. The horizontal axis specifies the users' serial numbers after this sorting. Note that the number of users participating in each workload may be slightly different, due to the random selection of how many new users arrive each week. As we can see, all the distributions are very similar to the original one. This is attributed to the fact that despite the random mixing due to the resampling, the sequence of jobs for each user is retained.

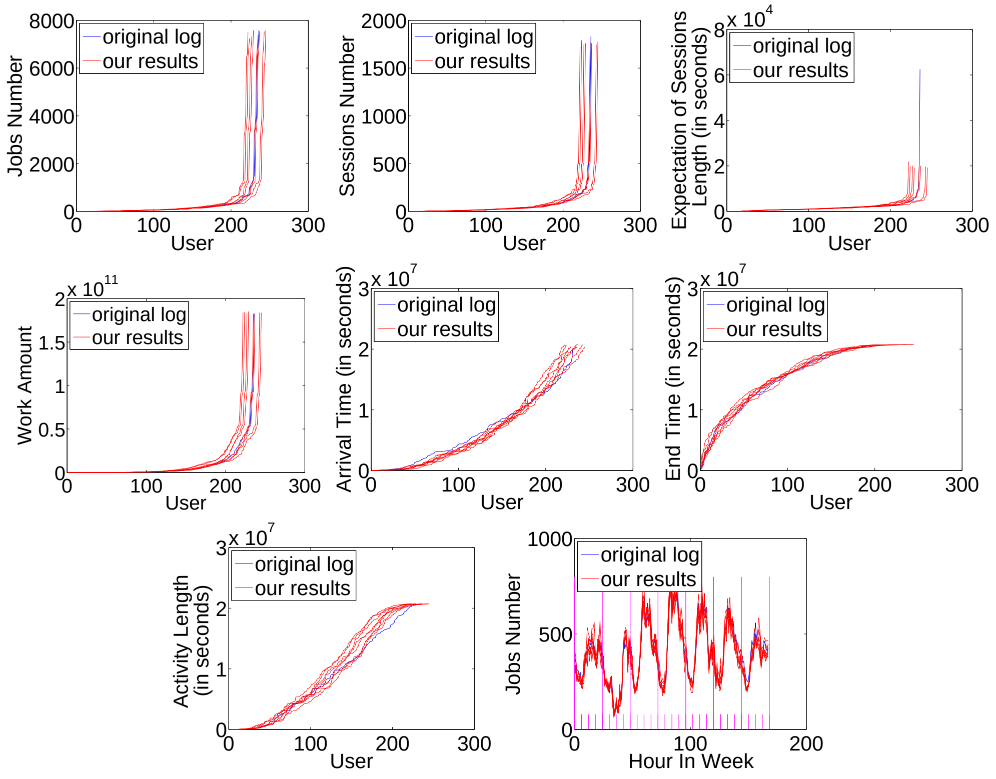


Figure 4. Comparison between various distributions of the original ANL-Intrepid log and 8 generated logs based on it. The last plot includes marks every 6 hours and a longer one at midnight.

Table VI. Locality calculated with the stack depth algorithm of the original workload and eight generated workloads (shown as average ± standard deviation).

| | Runtime | | Max runtime | | CPU number | |
|---|---|---|---|---|---|---|
| Log name | Orig. | Resamp. | Orig. | Resamp. | Orig. | Resamp. |
| LANL-CM5 | 36.01 | 36.97±0.37 | 2.53 | 2.51±0.03 | 1.15 | 1.08±0.01 |
| CTC-SP2 | 28.01 | 27.72±0.93 | 4.51 | 4.33±0.13 | 3.44 | 3.35±0.11 |
| KTH-SP2 | 35.25 | 35.03±0.37 | 6.46 | 6.42±0.10 | 3.83 | 3.87±0.10 |
| SDSC-SP2 | 25.13 | 24.48±0.82 | 4.09 | 3.96±0.11 | 2.89 | 2.79±0.08 |
| SDSC-BLUE | 25.11 | 25.97±0.28 | 3.12 | 3.46±0.02 | 1.58 | 1.57±0.01 |
| HPC2N | 18.67 | 19.44±0.19 | 2.36 | 2.37±0.02 | 0.98 | 0.95±0.01 |
| SDSC-DS | 25.01 | 25.68±0.59 | 4.09 | 3.95±0.15 | 2.07 | 2.02±0.06 |
| ANL-Intrepid | 13.21 | 14.02±0.24 | 2.58 | 2.56±0.07 | 1.28 | 1.27±0.02 |
| CEA-Curie | 28.0 | 28.06±0.59 | 2.42 | 2.39±0.08 | 4.93 | 5.02±0.26 |

## 5.2. *Locality*

Of course, marginal distributions don't tell the whole story. It is also important to retain the correlations in the workload. To verify that correlations are retained, we look into the locality of the workloads and their self-similarity.

A simple way to measure locality is using the stack-depth algorithm. To do this, we traverse the whole workload trace and extract a certain attribute of the jobs, e.g. their runtime. We keep these runtimes in a stack. For each new job from the trace, we check whether its runtime is already in the stack or not. If it is we note the depth in the stack where it was found, and move it up to the top of the stack. If it was not, we just put it on the top. Thus if the workload has locality and the same

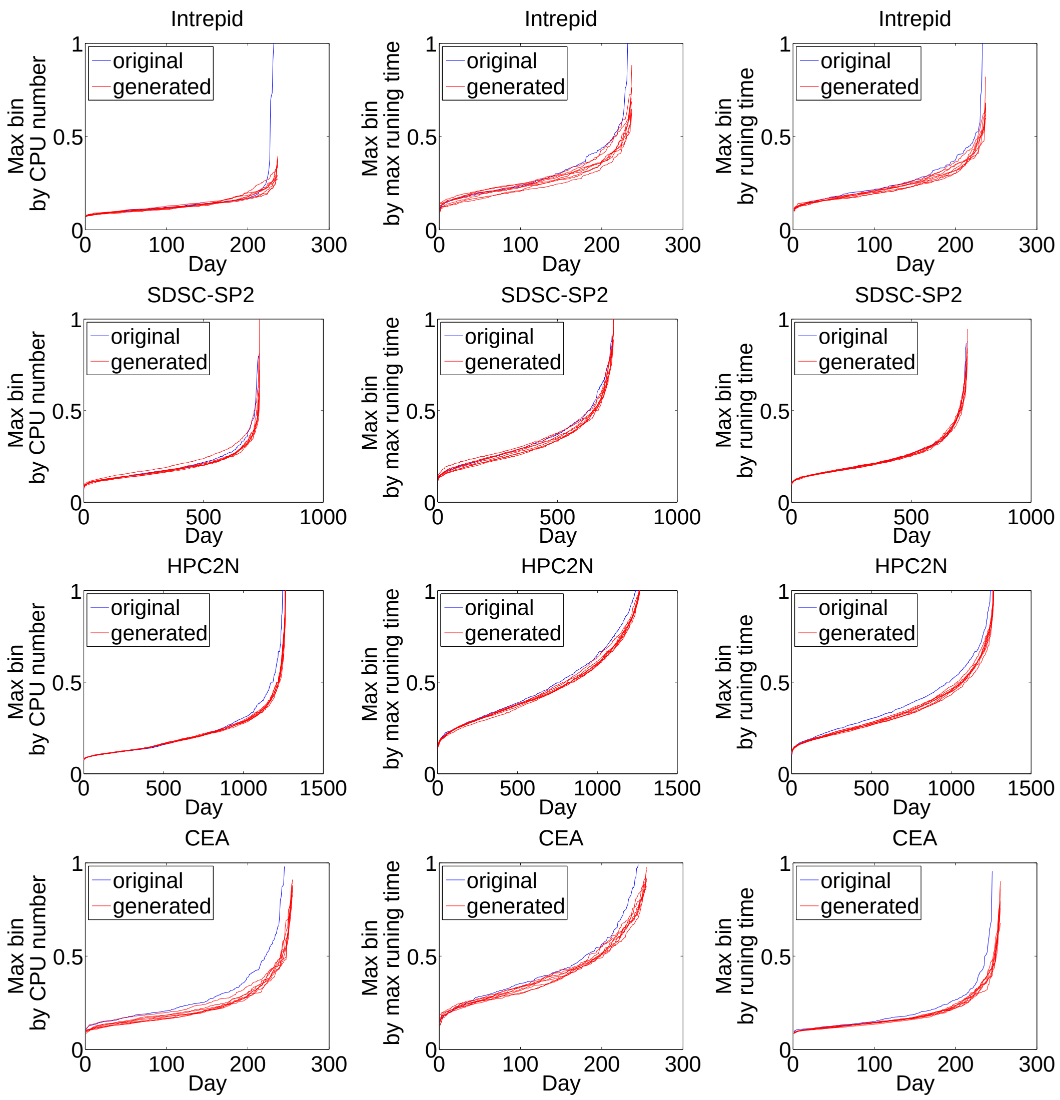


Figure 5. Using the bins-based algorithm to check locality. The graphs show the fraction of jobs in each day that are concentrated into one bin; if there was no locality, all values should be $\frac{1}{16} = 0.062$.

values tend to appear next to each other, we will often find them near the top of the stack and the average depth will be small. If there is no locality, the average depth will be about half the stack depth.

The results of performing this analysis using three workload properties are shown in Table VI. Note that for the runtime and maximal runtime properties we do not require that exactly the same value be found in the stack, as it is unreasonable to expect that long jobs will have the same runtime up to the second. We therefore allow differences of up to 5%. Still, for runtimes there are many more different values, and the average stack depth found is between 13 and 36. For the other attributes it is much lower. But the important thing is that the stack depths found for the resampled logs are very similar to those of the original logs.

The stack depth algorithm checks the similarity of successive jobs. But it can't measure the similarity of jobs submitted during a certain period of time, e.g. a day. To characterize this we use a metric designed specifically to capture local concentrations when sampling from a distribution [17]. The idea is as follows. First we characterize the underlying distribution by defining 16 equal-weight bins. In other words, we identify the $\frac{1}{16} = 6.25$ percentile, the $\frac{2}{16} = 12.5$ percentile, and so on. Then we divide the log into individual days. For each day, we find what fraction of the jobs fall in each of the bins we created before. If the distribution of jobs in this day is the same as the overall distribution, then the number of jobs in each bin will be the same. But if on this day there is a concentration of jobs with certain characteristics (as we expect when there is locality) then one of the bins will have many more jobs than the others. The distribution of the maximal bin across all the days in the log then characterizes the locality.

The results of performing this calculation are shown in Fig. 5 for four logs. We compare the distribution found in the original logs with those found in eight resampled logs. Note that the range of possible values is from $\frac{1}{16}$ to 1. As we can see, the resampled distributions are generally similar to the original ones, albeit in some cases the resampled distributions are a bit below the original. This means that there is a bit less locality, implying that some of the original locality is due to correlation between different users. Interestingly, the distributions for different logs, or different job attributes in the same log, can be different.

## *5.3. Self Similarity*

Another potentially important property of the workload is its self similarity, which reflects on its burstiness and long-range dependence. To validate the resampling methodology we need to compare the self similarity of the produced workloads to that of the original log. We will start with a brief description of self similarity and its meaning in this area. Then we will explain briefly how it is measured by the Hurst parameter, and describe how we calculate the Hurst parameter of the workloads. Finally, we will present data for the self-similarity of the produced workloads.

Self similarity refers to situations in which a phenomenon has the same general characteristics at different scales [32, 37]. If we zoom in, we see the same structure as we did before: parts of the whole are actually scaled-down copies of the whole. In nature and in workloads (as opposed to mathematics) we cannot expect perfect copies of the whole, but we can expect the same statistical properties.

For example, the job arrivals to a parallel supercomputer are seen to be bursty, and the same bursty behavior persists if we aggregate the arrivals over several orders of magnitude, by using longer and longer time units [16]. Self similarity like this has been shown in many computer workloads, including LAN traffic, web usage, and file systems [28, 36, 21, 8]. It is important because it means that the arrivals do not conform to a Poisson process, and that load fluctuations do not average out over longer time periods. The reason is that the arrival rates at different times are correlated with each other, and this correlation spans multiple time scales, leading to long-range dependence. The resulting load fluctuations have implications for capacity requirements and quality of service. Thus it is crucial to retain the self-similarity of workloads in order to achieve reliable performance evaluations.

The metric used to measure self-similarity is called the Hurst parameter ($H$). If this parameter is in the range of $0.5 < H < 1$, the process is self similar. Otherwise, it is not self similar. Assume we start with a time series $x_1, x_2 ... x_n$ (for example, $x_i$ may be the number of jobs that arrived in the $i$th time unit). First, we subtract the mean $\overline{X}$ from each sample, giving $z_i = x_i - \overline{X}$. Then, we calculate the deviation after $j$ time units for all $j$: $y_j = \sum_{i=1}^{j} z_i$. Then, we calculate the range that was covered, which is the difference between the maximum and the minimum deviations during these $n$ time units: $R(n) = \max_{1 \le j \le n} y_j - \min_{1 \le j \le n} y_j$. Finally, we calculate the standard deviation $S(n)$ of the observations $x_1, x_2 ... x_n$, and normalize the range. The model is that the rescaled range $\frac{R(n)}{S(n)}$ should grow like $c \cdot n^H$. To check this we take the log leading to $\log\left(\frac{R(n)}{S(n)}\right) = \log(c) + H \cdot \log(n)$. Thus if the process is indeed self similar, plotting the log of the rescaled range as a function of $\log n$ will lead to a straight line, and the slope of the line gives $H$.

In order to apply the above procedure, we need to generate data for different values of $n$. To choose the values of $n$ and the data elements for each $n$, we use common methods, as reviewed in [16]. Specifically, we use logarithmically spaced $n$s separated by a factor of 1.2, starting from where there are enough samples so that most intervals are not empty. For each $n$ we use multiple subsets of the data; for large $n$ these subsets may overlap.

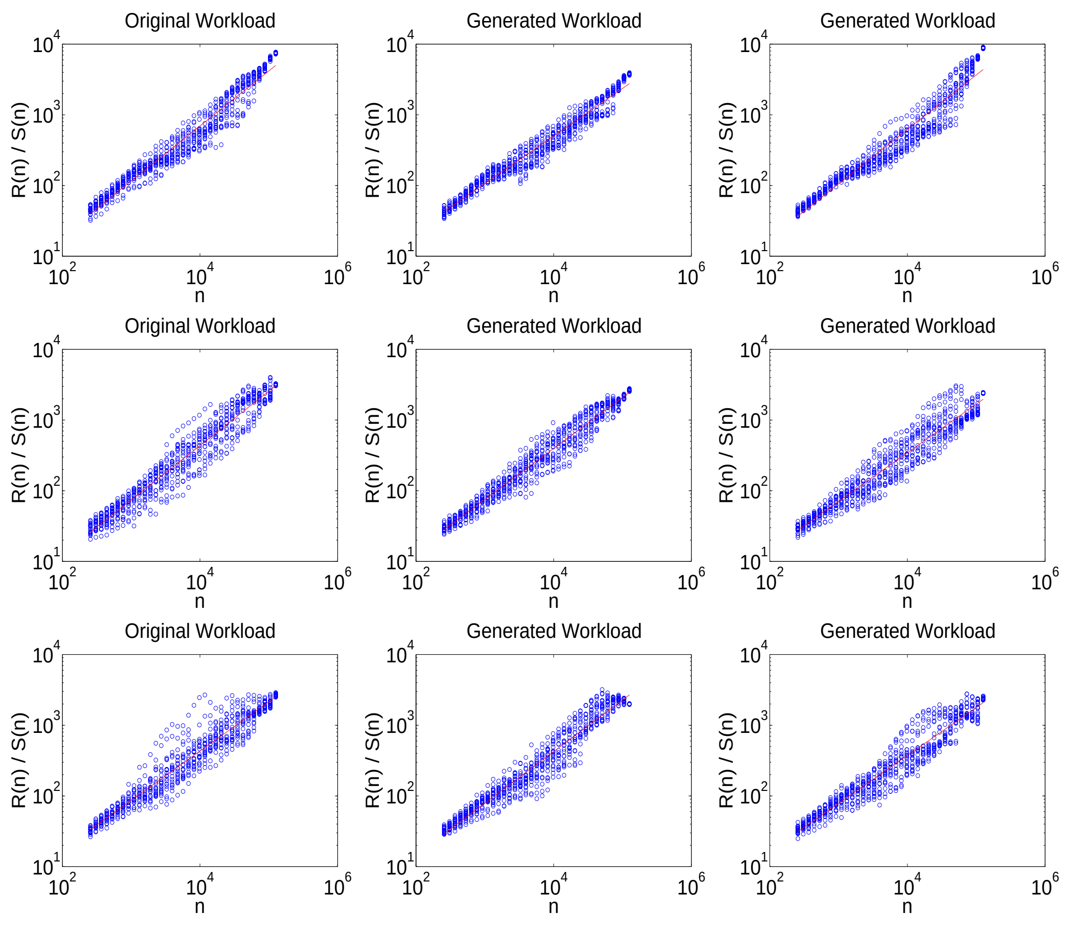


Figure 6. Comparison of Pox-plots for the original log and two resampled ones. From top: the LANL-CM5 log, the HPC2N log, and the ANL-Intrepid log.

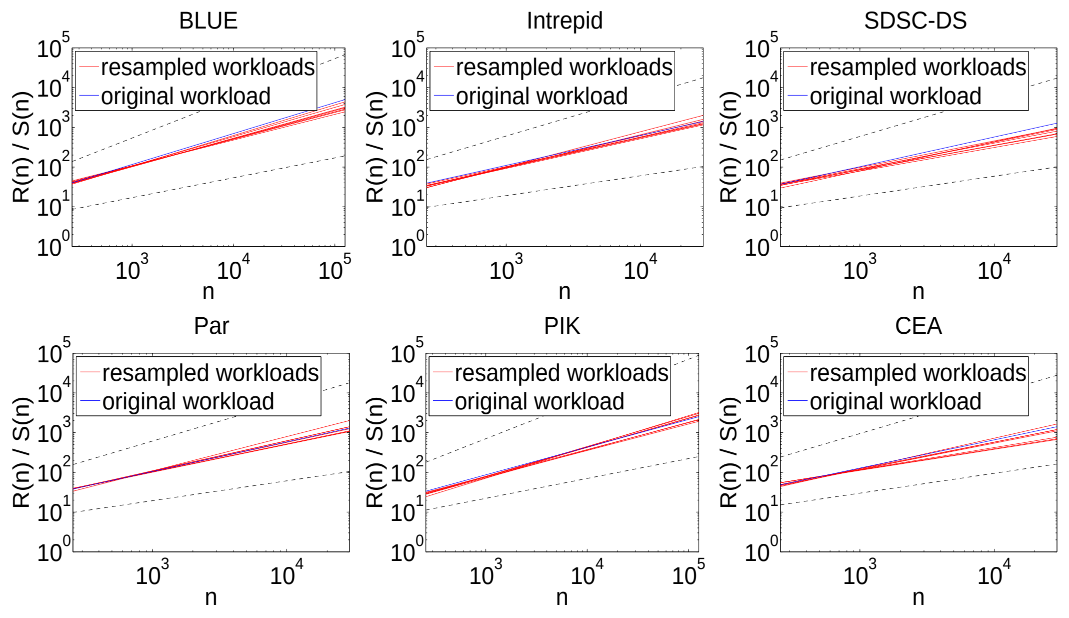


Figure 7. Comparison of the pox-plot regression lines of the original workload and the generated workloads. Dashed lines showing the slopes corresponding to $H = \frac{1}{2}$ and $H = 1$ are given for reference.

Table VII. Hurst parameter $H$ of the original workloads and the average and standard deviation of its value in eight generated workloads.

| Log name | Original $H$ | Resampled $H$ |
|---|---|---|
| LANL-CM5 | 0.690 | $0.679 \pm 0.043$ |
| SDSC-Par | 0.745 | $0.745 \pm 0.059$ |
| CTC-SP2 | 0.617 | $0.620 \pm 0.059$ |
| KTH-SP2 | 0.666 | $0.648 \pm 0.040$ |
| SDSC-SP2 | 0.638 | $0.715 \pm 0.028$ |
| SDSC-BLUE | 0.778 | $0.704 \pm 0.041$ |
| HPC2N | 0.769 | $0.722 \pm 0.033$ |
| SDSC-DS | 0.747 | $0.651 \pm 0.048$ |
| ANL-Intrepid | 0.754 | $0.789 \pm 0.054$ |
| PIK-IPLEX | 0.697 | $0.722 \pm 0.043$ |
| CEA-Curie | 0.710 | $0.630 \pm 0.090$ |

Given a large number of subsets of different sizes, we calculate the $\frac{R(n)}{S(n)}$ metric for each one and create a pox-plot, which is a scatter plot of these values on log-log axes. We use linear regression to find the trend line, and calculate its slope. We did this for all the logs except the low-load OSC-cluster, and for eight randomly resampled workloads that were produced as described in the previous section from each one. Fig. 6 shows examples of the pox plots and regression lines for three logs (LANL-CM5, HPC2N, and ANL-Intrepid). It is easy to see that in all cases the points create an oblique cloud close to a straight line, and that the plots for the resampled workloads are similar to those of the original workload.

Fig. 7 shows a direct comparison of the regression lines obtained from generated workloads and those that are obtained from the original ones. The slopes which give the $H$ values are compared in Table VII. From these results, it is clear that the slope of each resampled workload is far bigger than 0.5 and far smaller than 1 (actually, there is no slope lower than 0.6 or higher than 0.9). Therefore, we concluded that these resampled workloads behave similarly to the recorded workloads. From the table we can see that the average $H$ of the resampled workload is smaller than the original 6 times, and bigger 4 times, and that in most cases the difference is smaller than the standard deviation. This means that we don't have a large systematic deviation (which may indicate a problem), but only random fluctuations that affect each workload a bit differently.

Overall, these results provide significant support to the reliability of the generated workloads. In addition they indicate that the long-range dependence can be captured by the activity of the individual users, and does not depend on correlations between users. Therefore resampling at the user level retains the self similarity. This is in contrast to shuffling the workload, meaning dividing it into short segments and rearranging them, which is known to destroy self similarity [14]. The reason it works for user resampling is probably because of users who are active for long periods.

## 6. APPLICATIONS OF RESAMPLING

The use of resampling is expected to lead to more reliable performance evaluations, due to being based more closely on real workload traces, and incorporating all the complexities of real workloads — including those that are unknown to the analyst. In the following we discuss some examples.

### 6.1. *Verification of Performance Results*

As noted above, one of the problems with using a workload trace directly is that it provides a single data point. This has the obvious deficiency that it is impossible to calculate any kind of confidence intervals except perhaps by the method of batch means [35]. But with resampling we can create many resampled randomized versions of the workload, and evaluate the performance of

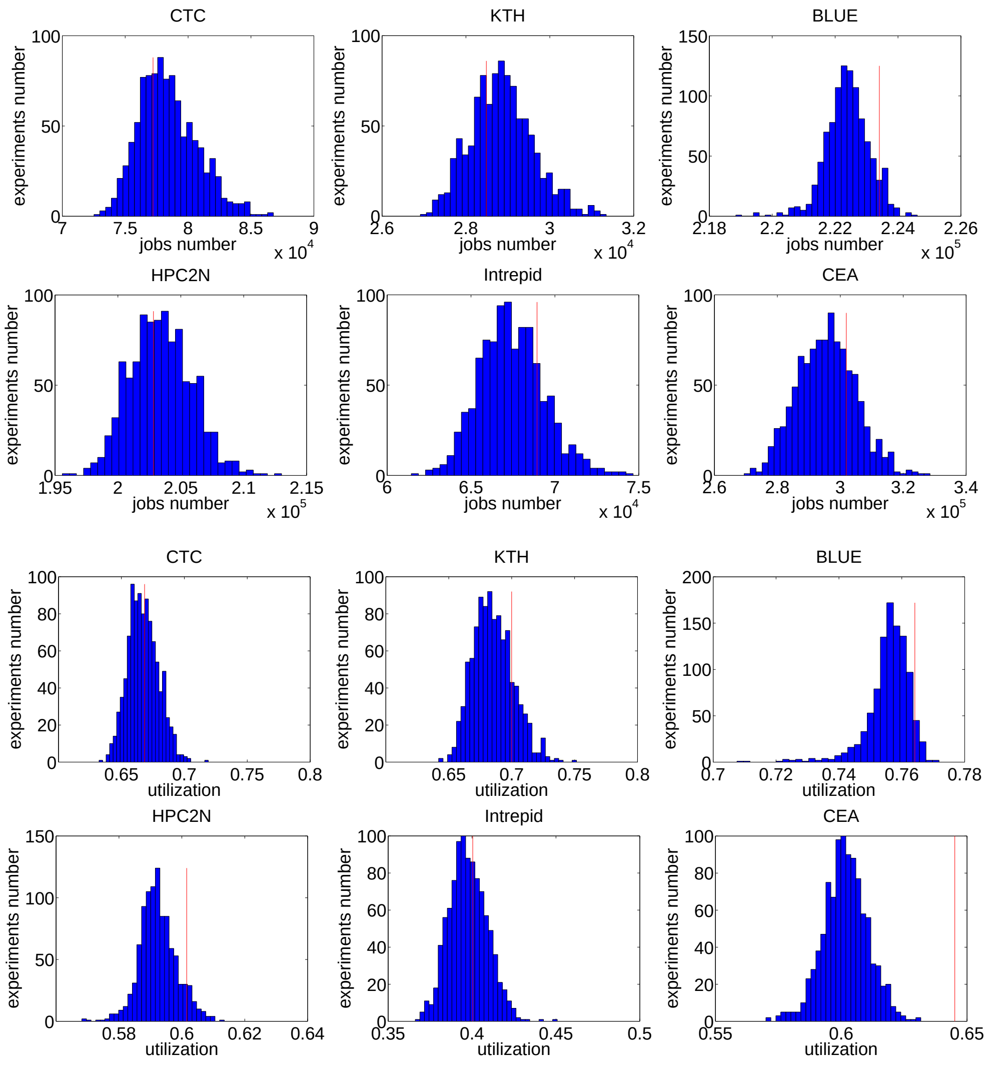


Figure 8. Histograms of the throughput and utilization in a thousand simulations with resampled workloads compared to using the original workload (vertical red line).

the system with all of them, thus obtaining multiple data points that all adhere to the same underlying statistics. The distribution of these data points can then be used to compute confidence intervals for performance metrics. This is essentially an application of the well-known technique of bootstrapping used in statistical analysis [12].

Given the resampling mechanism described above, implementing this idea is trivial: simply create a large number of workloads, say 1000, based on the original log, run the scheduler simulation on all of them, and tabulate the results. But to check this we need to also examine the basic characteristics of the produced workloads, and convince ourselves that they remain representative. To do so we indeed generated 1000 resampled variants of each log, calculated various metrics on each of these 1000 variants, and created a histogram of these metric values. We also included the original values for comparison.

We performed the checks on the nine logs from the archive that have user estimates (needed for the simulation), and results for six of them are shown in Fig. 8. The top two rows show that the throughput (represented by the total number of jobs during the simulation period) was typically distributed around the original value. The result for the BLUE log was the largest deviation observed; with this log 92% of the variants had a lower throughput than the original log, but the

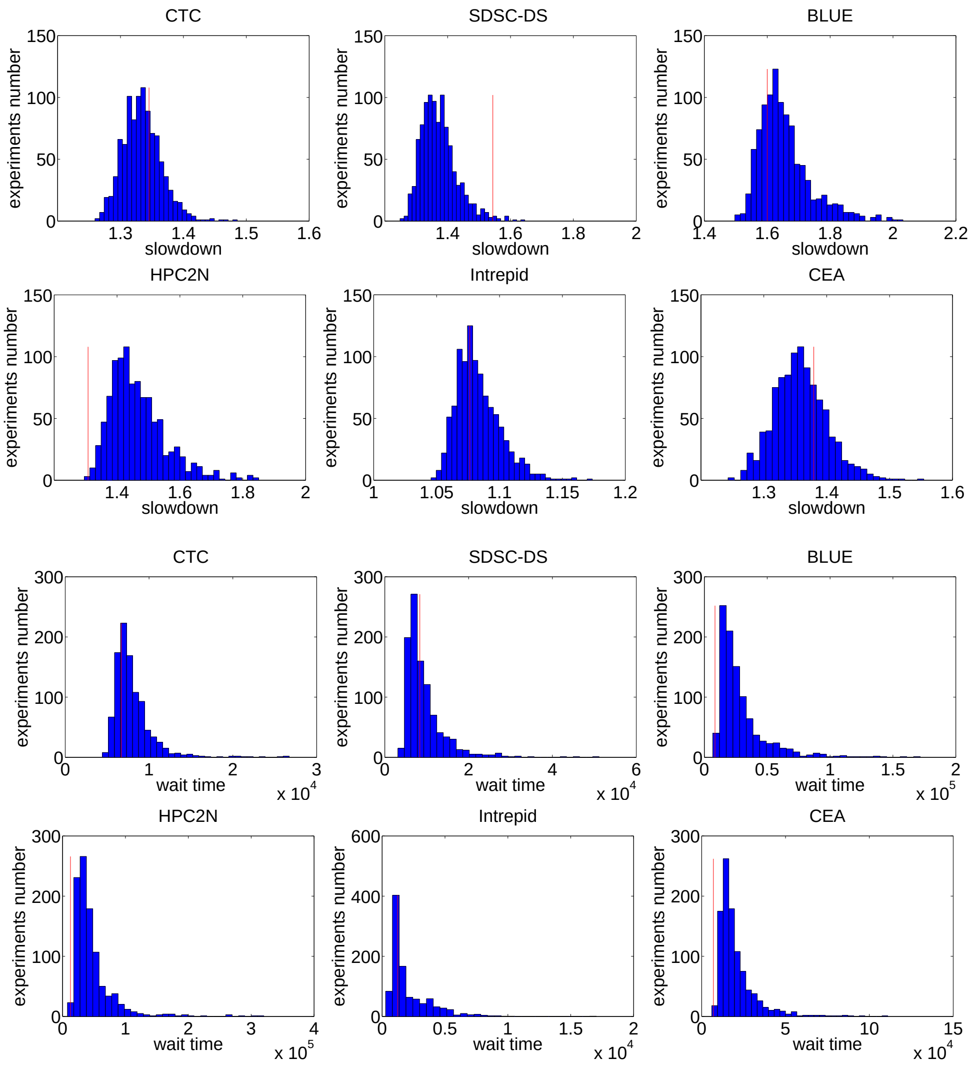


Figure 9. Histograms of the average slowdown and waiting time in a thousand simulations of EASY on resampled workloads, compared to a simulation using the original logs (vertical red line).

difference between the median throughput and the original was only 0.45%. For utilization (bottom two rows) the results were more diverse, and varied between distributions around the original value — as for CTC — and distributions that are generally below the original value — as for CEA, which was the most extreme. This may indicate some systematic bias which we do not understand yet. But note that even for CEA the difference was less than 8%.

Accepting the generated workload distributions as reasonable, we turn to check the results of evaluations of the EASY scheduler, which is probably the most commonly used backfilling scheduler [29, 15]. The results for waiting time and slowdown are shown in Fig. 9 (results for response time exhibit similar behavior to wait time). The slowdown results are the most varied. In six of the nine logs we checked, the distribution was more or less around the value obtained using the original log. This is exemplified by the BLUE and CTC logs in the figure. But in other cases the original result was at the very end of the distribution, either higher or lower than nearly all the others (as in DS or HPC2N, respectively, which were the two most extreme cases). The results for wait time were more one-sided, being distributed either around the original values (as for DS and CTC) or largely above them (as for BLUE and HPC2N). The explanation appears to be that in some

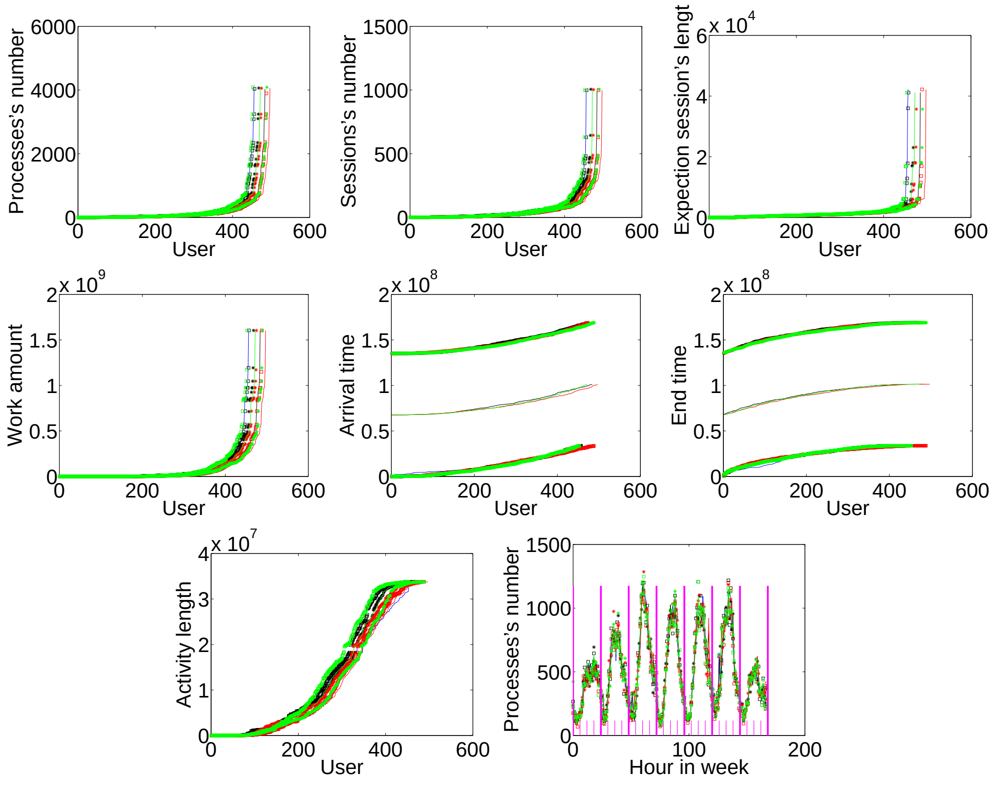


Figure 10. Comparison between the original SDSC-DS log to the first, third, and fifth parts of an extended resampled log that is 5 times longer.

logs there are more sparse periods, in which very few jobs arrive and therefore all wait times are short or nil. Our resampling tends to distribute users and jobs somewhat more evenly.

For both metrics, these results underscore the importance of using the resampling methodology to identify cases where the result using the original log may not be truly representative. Importantly, the spread of the results indicates that the resampling indeed produces workloads that are different from each other, even though they are derived from the same source and exhibit the same statistics. On the other hand, we never saw results that were completely separated from the original result, meaning that in all cases at least some of our 1000 repetitions produced results like the original log. Also, in most cases the most extreme differences were not more than 10–20%.

Note that this application of bootstrapping serves only to provide confidence intervals for evaluations based on a single log. We consider the possibility of extending this by mixing data from multiple logs in Section 6.5.1. Such mixing will provide confidence intervals for more general evaluations that are based on all the available data.

## *6.2. Extending a Trace*

Another simple use of workload resampling is in order to extend a trace. While some of our workload traces are pretty long, with hundreds of thousands of jobs submitted over 2 years or more, others are shorter. In addition, a significant part of the trace may be needed as a "warmup period" to ensure that the simulated system achieves its steady state [35]. Given only the raw traces, the length of the simulation may therefore be quite limited.

But with resampling we can extend the simulation to arbitrary lengths. As indicated above, this is achieved by regenerating long-term users, and randomly sampling new temporary users every week. In principle this can be continued indefinitely.

To check the resulting extended workloads, we studied three repetitions of extending given traces to five times their original length. For example, given a trace that represented one year's worth of activity, we used it to create three traces that are each five years long. We then compared the original trace with the first year, the third year, and the fifth year of each repetition. The results for the SDSC-DS log are shown in Fig. 10, using the same distributions as in Fig. 4.

As one can see, the distributions for all three repetitions and the three periods of the extended trace all agree with each other and with the original trace data to a high degree. Note that we treat each of the three periods as a separate log, and do not carry over users that were identified in one period to another period. This causes the distributions of arrival times and end times to be separated into three, corresponding to the different periods. Remarkably, in each of these we see the same end effects as in the original shorter trace.

### *6.3. Changing the Load*

An important aspect of systems performance evaluation is often to check the system's performance under different load conditions, and in particular, how performance degrades with increased load. Given a single trace, crude manipulations are typically used in order to change the load. These are

- Multiplying all arrival times by a constant, thus causing jobs to arrive at a faster rate and increasing the load, or causing them to arrive at a slower rate and decreasing the load. However, this also changes the daily cycle, for example causing jobs that were supposed to terminate during the night to extend into the next day. An alternative approach that has a similar effect is to multiply all runtimes by a constant. This has the deficiency of creating an artificial correlation between load and response time.
- Multiplying all job sizes (here meaning the number of processors they use) by a constant, and rounding to the nearest integer. This has two deficiencies. First, many jobs and machine sizes are powers of two. After multiplying by some constant in order to change the load, they will not be powers of two, which may have a strong effect on how they pack, and thus on the observed fragmentation. This effect can be much stronger than the performance effects we are trying to measure [30]. Second, small jobs cannot be changed with suitable fidelity as the sizes must always be integers. An alternative approach that has essentially the same effect is to modify the machine size. This at least avoids the problem presented by the small jobs.

With resampling, however, manipulating the load is relatively easy: One can simply increase or reduce the average number of active users. This changes the load while retaining all other attributes of the workload and avoiding the introduction of any artifacts. In particular, some logs have a very low utilization, in the range of 10–30%, which makes them uninteresting in terms of evaluating schedulers for parallel machines (because there are seldom enough concurrent jobs for the scheduler to have to make any decisions). Using resampling we can increase the load significantly and make these logs usable.

To implement this, three minor changes need to be made in the mechanism described above. The first is to change the number of long-term users in the initialization. Additional long-term users will be started as needed based on a random selection, taking care to use all existing long-term users before replicating one that was selected already, and also taking care that replicas of the same user will have a large difference in their start times. Likewise, we need to change the number of temporary users in the initialization. Finally, we need to change the rate at which additional temporary users arrive each week.

When users (and load) are added, the simulated system may saturate. We identify such conditions and ignore the saturated simulation results with a warning. Identifying saturation is based on noticing that the number of outstanding jobs (jobs that have arrived but not terminated yet) tends to grow. This is done as follows.

1. Tabulate the number of outstanding jobs at the beginning of each week of the simulation.
2. If the number of outstanding jobs grows due to a load fluctuation, but then decreases again, this does not indicate saturation. Therefore we replace each weekly count by the minimum count from that week to the end of the simulation, leading to a non-decreasing sequence of counts.

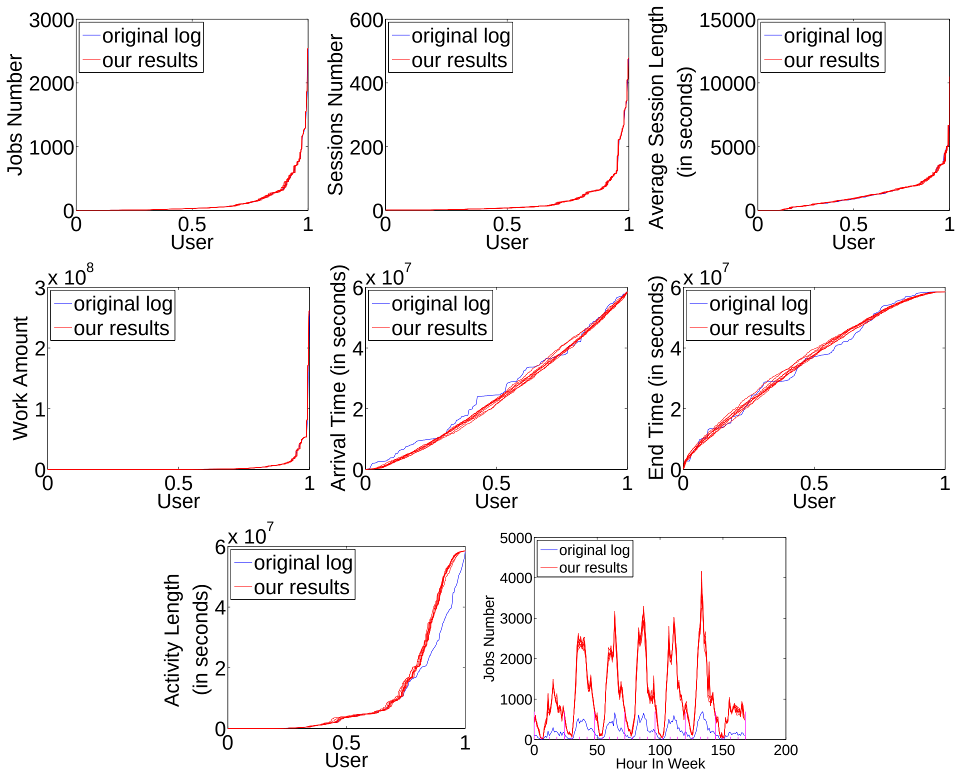


Figure 11. Comparison between the original OSC-cluster log to resampled workloads where the load is increased by a factor of 6.

3. Delete the last 20% of the values, to avoid false positives based on fluctuations that occur towards the end of the simulation.
4. Use linear regression to fit a straight line to the remaining counts. If the slope is lower than 1 (meaning that on average the number of outstanding jobs grows by no more than one job per week) the simulation is declared stable. If it is higher, the simulated system is saturated.

Verifying that resampling with a modified number of users leads to reasonable workloads shows that indeed all the distributions are similar to those of the original traces (but taking into account that the number of users is different). Fig. 11 shows the results for one extreme case, based on the OSC cluster log. The average utilization of this log is only 12.8%, making it unusable for evaluations of parallel job schedulers. We therefore increased the number of users by a factor of 6, targeting an average utilization of approximately 76.8%. In the graphs, the user numbers on the horizontal axis are normalized to the range $[0, 1]$ to enable comparison with the original log that has much fewer users. It is easy to see that the high-load simulations produce distributions that are very similar to the original log. The main difference is in the arrival time and end time distributions, which are smoother, because in our simulations users arrive at a constant average rate. Also, in the last graph portraying the weekly cycle of activity, one can see the big difference in the number of jobs that are being used.

The goal of all these workload manipulations is to enable the evaluation of parallel job schedulers, and in particular, their performance under different load conditions. To check this we again used simulations of the EASY backfilling scheduler [29]. For each log, we multiplied the number of users by various factors in the range 0.8 to 1.5, and performed 10 independent simulations (with different randomized resampling) for each load value. For the OSC cluster log, the range was from 1 to 9, because the original utilization of this log is very low as noted above. A sample of the results for slowdown and response time are shown in Fig. 12. The results for waiting time were very similar to those of response time.

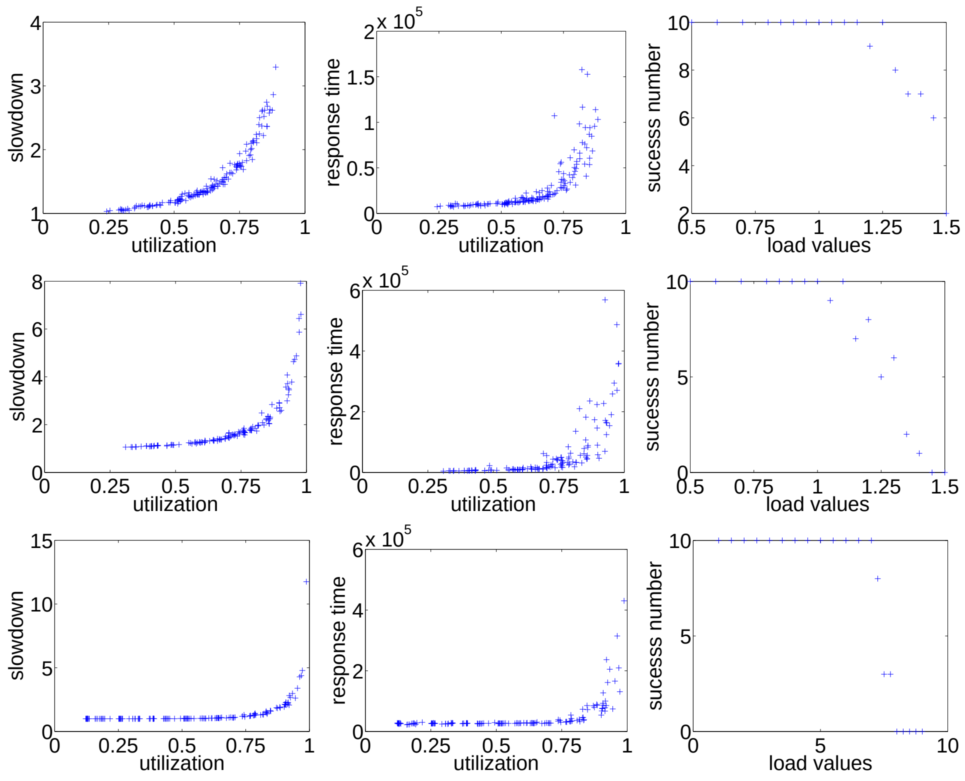


Figure 12. The performance of EASY under different load conditions for different logs: SDSC-DS, BLUE, and OSC-cluster respectively from top to bottom. Recall that the original OSC cluster log has very low load, so the load had to multiplied by higher factors to reach the range of interest.

The last panel for each log shows the fraction of simulations at each load level that were successful, meaning that our automated procedure did not conclude that the system is becoming saturated. Note that the loading factor, namely the factor by which we multiply the number of users, does not translate directly and deterministically into a commensurate change in the utilization. Due to the random selection of users there may be fluctuations in the load. Therefore we find that when the loading factor grows beyond 1, which represents the original load, the number of successful simulations begins to drop. Consequently there are fewer results for the higher loads, but all the valid results indicate a utilization of no more than 100%.

As the results in Fig. 12 show, the performance profiles are as one might expect from queueing analysis. At low loads performance is good, and increasing the load has little effect. But as the system approaches saturation, the performance deteriorates precipitously. Interestingly, different systems (as represented by the logs of their workloads) have different saturation points. SDSC-DS seems to saturate at less than 90% utilization, whereas BLUE and OSC come close to 100%. This reflects the ability of the scheduler to pack jobs together and reduce fragmentation, and depends both on the scheduler and on the workload statistics.

### 6.4. *Over-Sampling Rare Behaviors*

Workload logs sometimes contain unique users that behave anomalously in a specific period compared to the rest of the users. For example, a user may submit an inordinate number of jobs during a single week thus creating a flash crowd. Our goal here is to assess the effect of such behaviors on the performance of the rest of the jobs. We do this by creating special pseudo-users that encompass the special behavior, and then amplifying their weight in the generated workload by selecting these users more than others. For example, this allows us to create workloads with different

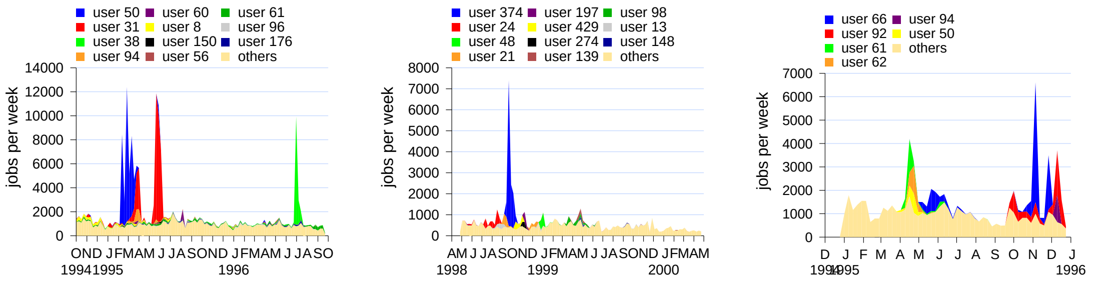


Figure 13. Flurries (flash crowds) in the original version of three logs: LANL-CM5, SDSC-SP2, and SDSC-Par95.

levels of flash crowds, and thus find the effect of the flash crowds on the performance of the rest of the jobs. Such simulations are an alternative to modeling the load spikes as suggested in [3].

The procedure to over-sample special behaviors of interest proceeds as follows:

1. We first create another pool of users, called the rare behaviors pool.
2. The rare behaviors of interest are defined by three parameters: the user's serial number, the period's beginning time, and the period's ending time. For each job in the workload, if the job is part of a rare behavior, instead of assigning it to the appropriate user in the temporary users pool or long-term users pool, we insert it to this user's place in the rare behaviors pool.
3. We define a parameter called RB_NPW to represent the expected number of times that rare behaviors should occur each week. This is a value between 0 and the arrival rate of new temporary users. In each week we divide the new temporary users into two: RB_NPW of them on average will be rare-behavior users, and the rest will be regular temporary users. Note that the total number of temporary users stays as before.
4. When evaluating the performance results of the simulation (throughput, wait time, response time, and slowdown) we skip the jobs that belong to rare behavior users, in order to focus on their influence on the rest of the jobs.

While rare behaviors are treated as a sub-class of temporary users, their definition is independent of how temporary users are normally defined. Thus the duration of a rare behavior can be longer than the limit on the activity of a temporary user; in other words, it can actually be a long-term user. Also, a single user's activity can be partitioned into several independent rare behaviors, or all the activity can be considered a single rare behavior.

The concept of over-sampling rare behaviors is completely general. But in the context of parallel workloads, a specific type of rare behavior that has aroused some interest is the so-called flurries of activity [41]. These are bursts of activity in which a single user submits a huge number of jobs during a relatively short period. Examples of the three workloads with the largest flurries, the LANL-CM5, SDSC-SP2, and SDSC-Par95, are shown in Fig. 13. The flurry jobs typically require very low resources (for example a single processor for less than a minute) and therefore their influence is unclear.

Flurries are similar to the flash crowds that have been observed in other types of workloads, e.g. the web, except that flash crowds are the result of the convergence of many users rather than the abnormal behavior of a single user. The work on workload flurries showed that they may taint performance evaluation results, and therefore the suggestion was to remove them [19]. Indeed, in other sections of this paper we consistently use the "cleaned" versions of the workloads, where flurries have been removed. But here we use the flurries from the original logs to demonstrate how we can amplify them and thus investigate the effect of having more or less flurries. The chosen set of rare behaviors in the LANL-CM5 log is users 50 and 38 which each have a single flurry, and user 31 who has two. In SDSC-SP2 there is one large flurry by user 374. Finally, in SDSC-Par95 we consider all the activity of users 66 and 92, approximately from October and until the end of the log.

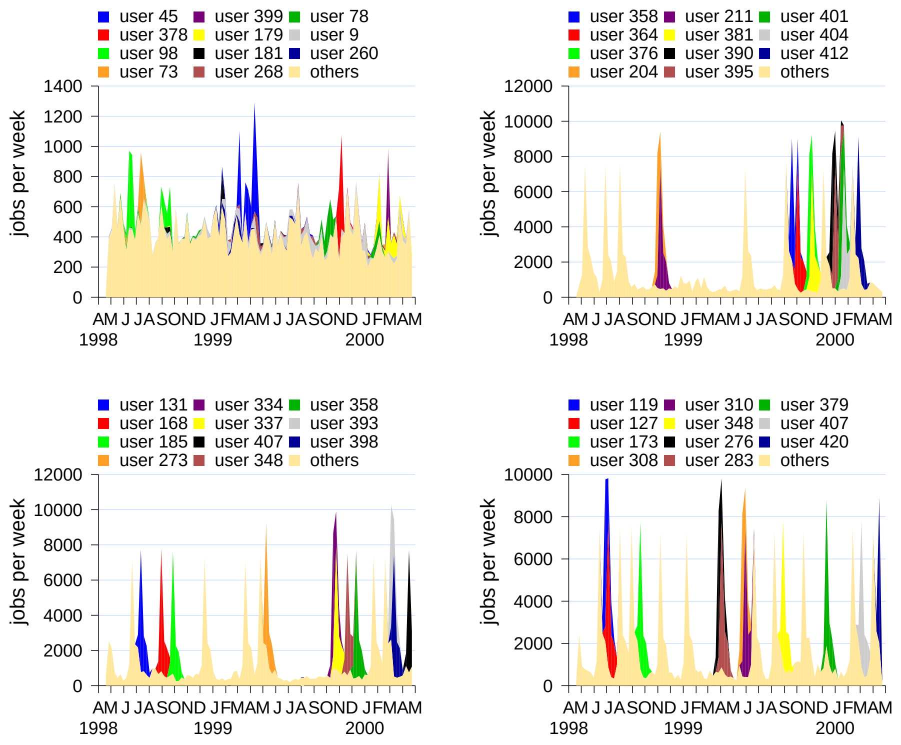


Figure 14. Adding flurries to the base workload of the SDSC-SP2 log. Note the much smaller scale in the first panel, where there are no flurries. In the others the values of RB_NPW are $\frac{1}{8}$, $\frac{1}{6}$, and $\frac{1}{4}$. In each case the 11 most active users are marked.

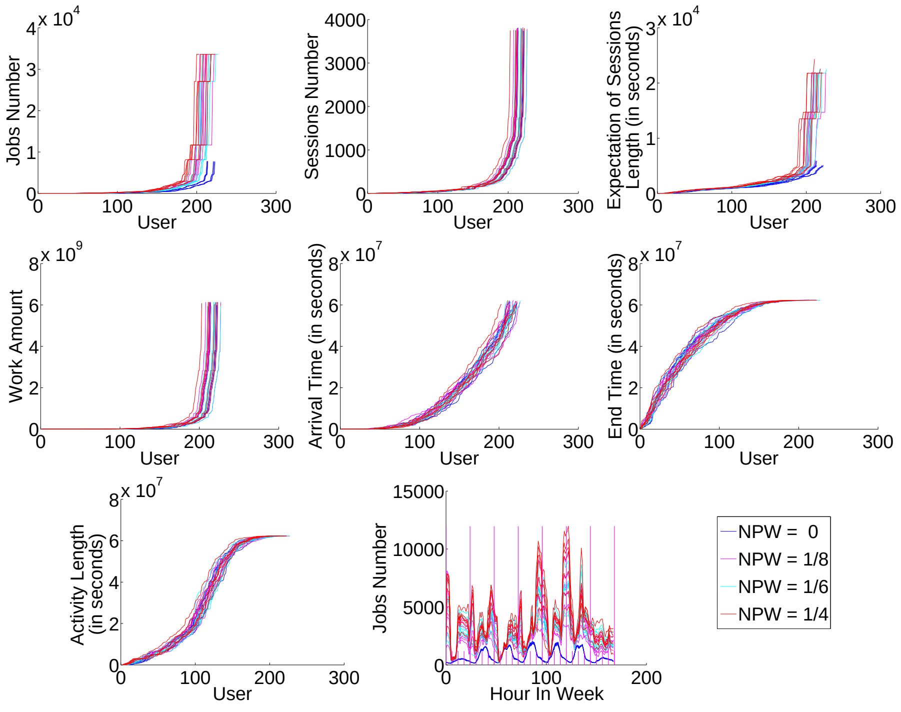


Figure 15. Comparison between the workloads created by running the simulation with different values of RB_NPW on the LANL-CM5 log.

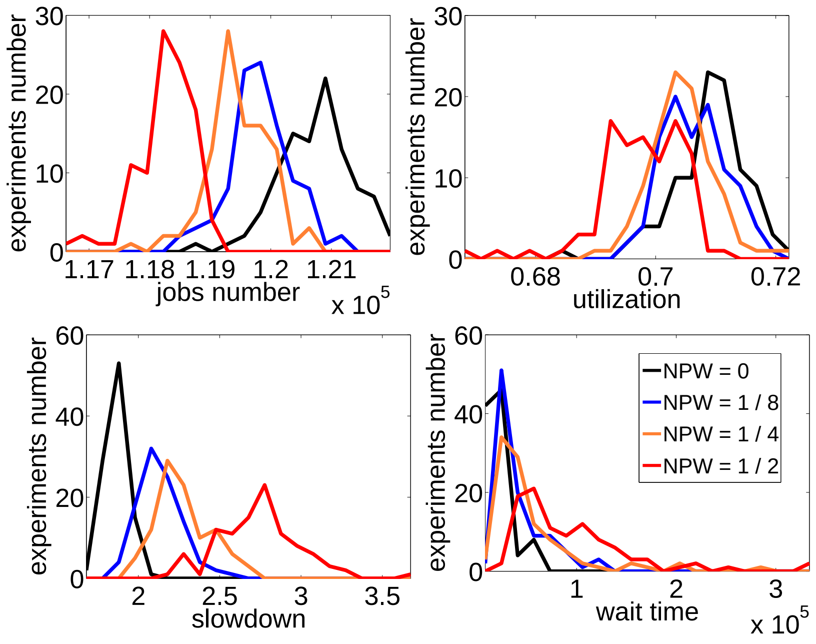


Figure 16. Distributions of number of non-flurry jobs, utilization, slowdown, and wait-time for 100 repetitions of simulating the performance of EASY on the LANL-CM5 workload with different frequencies of flurries.

First we will demonstrate that this application works and we can indeed control the prevalence of flurries. For each log we choose RB_NPW to be $0, \frac{1}{8}, \frac{1}{6}, \frac{1}{4}$. This means that we expect to have no flurries, or a flurry starting on average once in each 8 weeks, once in 6 weeks, and once in 4 weeks. The results for the SDSC-SP2 log are presented in Fig. 14. When there are no flurries at all (first graph), the scale of the number of jobs is dramatically lower. In addition, when the RB_NPW gets bigger, the scale also gets bigger, because it increases the probability that two flurries will be chosen in the same week or in close weeks. Finally, it is easy to see that there are more flurries when RB_NPW gets bigger.

The next step of the validation is to see how these changes affect the workloads' characteristics. To do this we repeated the experiment seven times for each value of RB_NPW and compared the created workloads. The results are presented in Fig. 15 for the CM5 log. The main differences can be seen in the graphs of the number of jobs and the average session lengths by each user. When there are no flurries, the maximal values are much lower. When flurries are included we see discrete components in the distributions that reflect repetitions of the same large flurries.

To show the effect of flurries on the normal, non-flurry jobs, we conducted 100 repetitions of each workload and simulated their scheduling. We used four different values of RB_NPW: $0, \frac{1}{8}, \frac{1}{4}$, and $\frac{1}{2}$. Then we created a histogram of the performance for each value. The results for the LANL-CM5 log are presented in Fig. 16. The results for the other two logs are less distinct, but qualitatively similar.

The graphs in the top row show that the number of jobs and the utilization are lightly reduced when the value of RB_NPW gets bigger. The reason is that we get approximately the same amount of users for each value of RB_NPW, but when its value gets bigger, the percentage of flurries is increased. Due to the fact that the flurry jobs are not included in these statistics, less jobs are left.

The other two metrics, wait time and slowdown, aren't biased and really indicate the performance reliably. For all three logs, more flurries lead to worse performance. For example, it is easy to see that with a lower RB_NPW value, there are much more experiments with low slowdown and wait time. For all the logs higher RB_NPW values cause a heavier tailed distribution of both metrics, and therefore worse performances. Therefore we conclude that flurries disrupt the performance of all the jobs, even if each flurry job uses only little resources.

## *6.5. Additional Applications*

In addition to the above, we note the following ideas for using workload resampling. These have not been tested yet and are presently left for future work.

*6.5.1. Mixing Traces.* In many cases we have more than one trace at our disposal, typically coming from different locations or different times. To obtain generally applicable results, data from all these traces should be used. This can be done either by performing evaluations based on each trace individually, or by mixing the traces, that is by resampling from a number of traces rather than from only one trace. This mechanism has been used in the past in order to reduce the dependence of analysis results on a single trace [48], or to increase the load [24]. It was also suggested for Tmix [44].

Resampling from several traces is based on the assumption that this is the better way to achieve general results that are independent of the peculiarities of any individual trace. An interesting research question is whether this is indeed the case. And could it be that mixing and evaluations are actually transitive, and equivalent results are obtainable by averaging of results from multiple traces that are resampled independently? We intend to study this question by using both approaches and comparing the results.

*6.5.2. Improving Stationarity.* A special case is using resampling to create a stationary workload trace. Many of the original traces seem to be non-stationary, with different parts having different statistical properties. As a consequence performance results are then some sort of weighted average of the results under somewhat different conditions, but we don't know the details of these conditions or the weights. Resampling can be used to mix the different conditions and create a more uniform trace.

Alternatively, when examined more closely the workloads are sometimes found to be *piecewise* stationary, meaning that they are relatively stationary for some time and then change. It is therefore better to perform several stationary evaluations and combine the results, rather than using a single non-stationary model that might lose important locality properties. Resampling can then be used to create stationary segments that are long enough to be simulated reliably.

We note in passing that some workloads are inherently non-stationary because the system configuration was changed during the time that the workload trace was recorded [20]. In such situations the trace should actually not be used as-is, because the results would be an unknown mix of results for the two different constituent workloads. But the problem would be solved by using resampling, as then all the data will be used to create a single consistent workload.

*6.5.3. Improved Shaking to Reduce Sensitivity.* Simulations of system performance are sometimes very sensitive to the exact value of some input parameter. For example, we have found a specific case where changing the runtime of one job from 18 hours and 30 seconds to exactly 18 hours caused the average bounded slowdown of *all* the jobs in the trace to change by about 8% [41]. We developed "shaking" as a general methodology to overcome such mishaps [42].

The idea of shaking is to cause small random perturbations to the workload and re-run the evaluation. This is repeated many times, leading to a distribution of results. This distribution is then used as the outcome of the evaluation, rather than the single point derived from the original trace. The claim is that the distribution (or its mean) more faithfully characterizes the results that would be obtained by this workload and similar ones. Our results indicate that shaking does indeed reduce instability considerably in several different cases.

The original formulation of shaking operated at the job level. Each job was moved slightly independently of the others. This could potentially cause problems if say one job was delayed and a subsequent job was moved forward and ended up before the first job. We therefore intend to now try to perform shaking at a higher level, e.g. by slightly shifting whole sessions, or even the sub-traces belonging to different users. The effect will be evaluated by comparing the original results with our current shaking results and the new shaking results. Shaking will also be compared with resampling to allow for statistical analysis as described above.

*6.5.4. Selective Manipulation of the Workload.* Another reason to manipulate workloads is to change their properties, so as to check the effect of these properties on system performance. In the current work we treat all users as equivalent, but this is not really so. Some users may run long jobs. Others may prefer numerous small jobs. Some users run jobs that require a lot of memory while others run more compute-intensive jobs.

The implication is that we can influence the characteristics of the workload by classifying users according to their behavior (or the behavior of their applications), and then resample with a selective bias in favor of a certain class of users. This is an extension of the idea of emphasizing rare behaviors as described above. It will enable the creation of workloads that stress different parts of the system.

*6.5.5. Reducing or Enhancing Locality.* Finally, a special case of manipulating the workload is changing its locality properties, as was done e.g. in [33, 5] (where they identify locality with burstiness). Locality can be very meaningful for adaptive systems that learn about their workload and adjust accordingly [17].

The mechanism for changing the locality is to introduce locality into the sampling process. Locality is typically present in user sessions (as we showed in Section 3). Therefore, to reduce locality the resampling must be done at the job level, not the session or user level. Regrettably, simple randomization does not work, as it violates the workload's stability properties. We will therefore need to develop a mechanism for resampling jobs subject to stability constraints. The question is how to do so and still get good randomization.

Enhancing locality can be done by introducing repetitions, i.e. specifically selecting the same jobs over and over again [17]. However, this also needs to be done subject to stability constraints, and subject to the overall statistical properties of the workload.

## 7. CONCLUSIONS

Workload resampling is proposed as a mechanism which allows the performance analyst to marry the realism of workload traces with the flexibility of workload models. Moreover, it combines the following attributes:

- Retaining the complex internal structure of the original trace, including features that we do not know about, and
- Allowing for manipulations that affect specific properties that we know about and want to change as part of the evaluation.

The idea is to partition a given workload trace into independent sub-traces, e.g. representing the activity of individual users. These subtraces can then be re-combined in different ways in order to create new workload traces with desired attributes: they can be longer than the original, have a higher or lower load than the original, or just be different from the original so as to provide an additional data point.

A major concern in this work was how to do the resampling correctly, meaning that the created workloads will be as similar as possible to the original workload. One aspect of this was the decision to perform the resampling at the level of users, and not, say, at the level of individual sessions. This maintains the correlations between successive sessions of the same user. Another aspect was the decision to retain the time of day and day of week when each user starts (and hence also when each job starts). This leads to workloads that retain the correlations among users who all operate according to a common daily and weekly cycle.

One concern that was not handled here is the issue of workload stability constraints. Real workloads exhibit a self-throttling effect whereby less additional work is submitted when the system is highly loaded. Given that we use each user's sequence of jobs as-is, our generated workload traces will not display such effects. To be more realistic we therefore need to model the feedback from system performance to user behavior. Such models turn out to be rather complicated, and this work will be reported separately when it is completed.

The above description focused on parallel system workloads, which are useful for evaluating the performance of parallel job schedulers. However, we believe that workload resampling has far wider applicability. Specifically, workload resampling is clearly applicable to any context in which the composition of the workload can be described as a hierarchical structure. Examples include networking, web servers, file systems, and architectural workloads. For example, it would be interesting to try to replace the large benchmark suites used in computer architecture evaluations with workload mixes based on resampling from the different applications in the suite. If successful, this may lead to an innovative fast approach for evaluating new designs. But using workload resampling in other domains first requires additional research, e.g. to determine the most appropriate granularity of resampling.

*Acknowledgments*

This research was supported by the Ministry of Science and Technology, Israel

## REFERENCES

1. J. Aikat, S. Hasan, K. Jeffay, and F. D. Smith, "*Towards traffic benchmarks for empirical networking research: The role of connection structure in traffic workload modeling*". In 20th *Modeling, Anal. & Simulation of Comput. & Telecomm. Syst.*, pp. 78–86, Aug 2012, DOI:10.1109/MASCOTS.2012.19.
2. P. Barford and M. Crovella, "*Generating representative web workloads for network and server performance evaluation*". In *SIGMETRICS Conf. Measurement & Modeling of Comput. Syst.*, pp. 151–160, Jun 1998, DOI: 10.1145/277851.277897.
3. P. Bodík, A. Fox, M. J. Franklin, M. I. Jordan, and D. A. Patterson, "*Characterizing, modeling, and generating workload spikes for stateful services*". In 1st *Symp. Cloud Comput.*, pp. 241–252, Jun 2010, DOI: 10.1145/1807128.1807166.
4. O. Boiman and M. Irani, "*Detecting irregularities in images and in video*". In 10th *IEEE Intl. Conf. Comput. Vision*, vol. 1, pp. 462–469, Oct 2005, DOI:10.1109/ICCV.2005.70.
5. G. Casale, N. Mi, L. Cherkasova, and E. Smirni, "*Dealing with burstiness in multi-tier applications: Models and their parameterization*". *IEEE Trans. Softw. Eng.* **38**(**5**), pp. 1040–1053, Sep/Oct 2012, DOI: 10.1109/TSE.2011.87.
6. Y. Chen, S. Alspaugh, and R. Katz, "*Interactive analytical processing in big data systems: A cross-industry study of MapReduce workloads*". *Proc. VLDB Endowment* **5**(**12**), pp. 1802–1813, Aug 2012.
7. Y. Chen, A. Ganapathi, R. Griffith, and R. Katz, "*The case for evaluating MapReduce performance using workload suites*". In 19th *Modeling, Anal. & Simulation of Comput. & Telecomm. Syst.*, pp. 390–399, Jul 2011, DOI: 10.1109/MASCOTS.2011.12.
8. M. E. Crovella and A. Bestavros, "*Self-similarity in world wide web traffic: Evidence and possible causes*". In *SIGMETRICS Conf. Measurement & Modeling of Comput. Syst.*, pp. 160–169, May 1996, DOI: 10.1145/233008.233038.
9. S. Dubnov, Z. Bar-Joseph, R. El-Yaniv, D. Lischinski, and M. Werman, "*Synthesizing sound textures through wavelet tree learning*". *IEEE Comput. Graphics & Applications* **22**(**4**), pp. 38–48, Jul 2002, DOI: 10.1109/MCG.2002.1016697.
10. M. R. Ebling and M. Satyanarayanan, "*SynRGen: An extensible file reference generator*". In *SIGMETRICS Conf. Measurement & Modeling of Comput. Syst.*, pp. 108–117, May 1994, DOI:10.1145/183018.183030.
11. B. Efron, "*Bootstrap methods: Another look at the jackknife*". *Ann. Statist.* **7**(**1**), pp. 1–26, Jan 1979, DOI: 10.1214/aos/1176344552.
12. B. Efron and G. Gong, "*A leisurely look at the bootstrap, the jackknife, and cross-validation*". *The American Statistician* **37**(**1**), pp. 36–48, Feb 1983, DOI:10.2307/2685844.
13. C. Ernemann, B. Song, and R. Yahyapour, "*Scaling of workload traces*". In *Job Scheduling Strategies for Parallel Processing*, pp. 166–182, Springer-Verlag, 2003, DOI:10.1007/10968987_9. Lect. Notes Comput. Sci. vol. 2862.
14. A. Erramilli, O. Narayan, and W. Willinger, "*Experimental queueing analysis with long-range dependent packet traffic*". *IEEE/ACM Trans. Networking* **4**(**2**), pp. 209–223, Apr 1996, DOI:10.1109/90.491008.
15. Y. Etsion and D. Tsafrir, *A Short Survey of Commercial Cluster Batch Schedulers*. Tech. Rep. 2005-13, Hebrew University, May 2005.
16. D. G. Feitelson, "*Workload modeling for performance evaluation*". In *Performance Evaluation of Complex Systems: Techniques and Tools*, M. C. Calzarossa and S. Tucci (eds.), pp. 114–141, Springer-Verlag, Sep 2002, DOI: 10.1007/3-540-45798-4_6. Lect. Notes Comput. Sci. vol. 2459.
17. D. G. Feitelson, "*Locality of sampling and diversity in parallel system workloads*". In 21st *Intl. Conf. Supercomputing*, pp. 53–63, Jun 2007, DOI:10.1145/1274971.1274982.
18. D. G. Feitelson and E. Shmueli, "*A case for conservative workload modeling: Parallel job scheduling with daily cycles of activity*". In 17th *Modeling, Anal. & Simulation of Comput. & Telecomm. Syst.*, Sep 2009, DOI: 10.1109/MASCOT.2009.5366139.
19. D. G. Feitelson and D. Tsafrir, "*Workload sanitation for performance evaluation*". In *IEEE Intl. Symp. Performance Analysis Syst. & Software*, pp. 221–230, Mar 2006, DOI:10.1109/ISPASS.2006.1620806.

20. D. G. Feitelson, D. Tsafrir, and D. Krakov, *Experience with the Parallel Workloads Archive*. Tech. Rep. 2012-6, Hebrew University, Apr 2012.
21. S. D. Gribble, G. S. Manku, D. Roselli, E. A. Brewer, T. J. Gibson, and E. L. Miller, "*Self-similarity in file systems*". In *SIGMETRICS Conf. Measurement & Modeling of Comput. Syst.*, pp. 141–150, Jun 1998, DOI: 10.1145/277851.277894.
22. F. Hernández-Campos, K. Jeffay, and F. D. Smith, "*Modeling and generating TCP application workloads*". In 4th *Broadband Comm., Netw. & Syst.*, pp. 280–289, Sep 2007, DOI:10.1109/BROADNETS.2007.4550436.
23. J. Jann, P. Pattnaik, H. Franke, F. Wang, J. Skovira, and J. Riodan, "*Modeling of workload in MPPs*". In *Job Scheduling Strategies for Parallel Processing*, pp. 95–116, Springer-Verlag, 1997, DOI:10.1007/3-540-63574-2_18. Lect. Notes Comput. Sci. vol. 1291.
24. P. Kamath, K.-c. Lan, J. Heidemann, J. Bannister, and J. Touch, "*Generation of high bandwidth network traffic traces*". In 10th *Modeling, Anal. & Simulation of Comput. & Telecomm. Syst.*, pp. 401–412, Oct 2002, DOI: 10.1109/MASCOT.2002.1167101.
25. L. Kovar, M. Gleicher, and F. Pighin, "*Motion graphs*". *ACM Trans. Graph.* **21(3)**, pp. 473–482, Jul 2002, DOI: 10.1145/566570.566605.
26. 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.
27. V. Kwatra, A. Schödl, I. Essa, G. Turk, and A. Bobick, "*Graphcut textures: Image and video synthesis using graph cuts*". *ACM Trans. Graph.* **22(3)**, pp. 277–286, Jul 2003, DOI:10.1145/882262.882264.
28. W. E. Leland, M. S. Taqqu, W. Willinger, and D. V. Wilson, "*On the self-similar nature of Ethernet traffic*". *IEEE/ACM Trans. Networking* **2(1)**, pp. 1–15, Feb 1994, DOI:10.1109/90.282603.
29. D. Lifka, "*The ANL/IBM SP scheduling system*". In *Job Scheduling Strategies for Parallel Processing*, pp. 295–303, Springer-Verlag, 1995, DOI:10.1007/3-540-60153-8_35. Lect. Notes Comput. Sci. vol. 949.
30. V. Lo, J. Mache, and K. Windisch, "*A comparative study of real workload traces and synthetic workload models for parallel job scheduling*". In *Job Scheduling Strategies for Parallel Processing*, pp. 25–46, Springer-Verlag, 1998, DOI:10.1007/BFb0053979. Lect. Notes Comput. Sci. vol. 1459.
31. 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.
32. B. B. Mandelbrot, *The Fractal Geometry of Nature*. W. H. Freeman and Co., 1982.
33. N. Mi, G. Casale, L. Cherkasova, and E. Smirni, "*Injecting realistic burstiness to a traditional client-server benchmark*". In 6th *Intl. Conf. Autonomic Comput.*, pp. 149–158, Jun 2009, DOI: 10.1145/1555228.1555267.
34. "*Parallel workloads archive*". URL http://www.cs.huji.ac.il/labs/parallel/workload/.
35. K. Pawlikowski, "*Steady-state simulation of queueing processes: A survey of problems and solutions*". *ACM Comput. Surv.* **22(2)**, pp. 123–170, Jun 1990, DOI:10.1145/78919.78921.
36. V. Paxson and S. Floyd, "*Wide-area traffic: The failure of Poisson modeling*". *IEEE/ACM Trans. Networking* **3(3)**, pp. 226–244, Jun 1995, DOI:10.1109/90.392383.
37. M. Schroeder, *Fractals, chaos, Power Laws*. W. H. Freeman and Co., 1991.
38. 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.
39. J. Sommers and P. Barford, "*Self-configuring network traffic generation*". In 4th *Internet Measurement Conf.*, pp. 68–81, Oct 2004, DOI:10.1145/1028788.1028798.
40. D. Talby and D. G. Feitelson, "*Improving and stabilizing parallel computer performance using adaptive backfilling*". In 19th *Intl. Parallel & Distributed Processing Symp.*, Apr 2005, DOI:10.1109/IPDPS.2005.252.
41. D. Tsafrir and D. G. Feitelson, "*Instability in parallel job scheduling simulation: The role of workload flurries*". In 20th *Intl. Parallel & Distributed Processing Symp.*, Apr 2006, DOI:10.1109/IPDPS.2006.1639311.
42. D. Tsafrir, K. Ouaknine, and D. G. Feitelson, "*Reducing performance evaluation sensitivity and variability by input shaking*". In 15th *Modeling, Anal. & Simulation of Comput. & Telecomm. Syst.*, pp. 231–237, Oct 2007, DOI: 10.1109/MASCOTS.2007.58.
43. K. V. Vishwanath and A. Vahdat, "*Swing: Realistic and responsive network traffic generation*". *IEEE/ACM Trans. Networking* **17(3)**, pp. 712–725, Jun 2009, DOI:10.1109/TNET.2009.2020830.
44. M. C. Weigle, P. Adurthi, F. Hernández-Campos, K. Jeffay, and F. D. Smith, "*Tmix: A tool for generating realistic TCP application workloads in ns-2*". *Comput. Commun. Rev.* **36(3)**, pp. 67–76, Jul 2006, DOI: 10.1145/1140086.1140094.
45. Y. Wexler, E. Schechtman, and M. Irani, "*Space-time video completion*". In *Conf. Comput. Vision & Pattern Recog.*, vol. 1, pp. 120–127, Jun 2004, DOI:10.1109/CVPR.2004.1315022.
46. 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, 2012, DOI:10.1007/978-3-642-35867-8_12. Lect. Notes Comput. Sci. vol. 7698.
47. N. Zakay and D. G. Feitelson, "*Workload resampling for performance evaluation of parallel job schedulers*". In 4th *Intl. Conf. Performance Engineering*, pp. 149–159, Apr 2013, DOI:10.1145/2479871.2479893.
48. J. Zilber, O. Amit, and D. Talby, "*What is worth learning from parallel workloads? a user and session based analysis*". In 19th *Intl. Conf. Supercomputing*, pp. 377–386, Jun 2005, DOI:10.1145/1088149.1088200.

# 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.

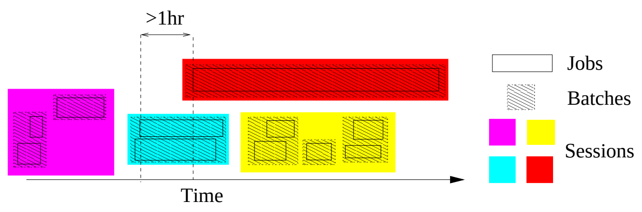


Fig. 1. Illustration of batches and sessions.

## 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

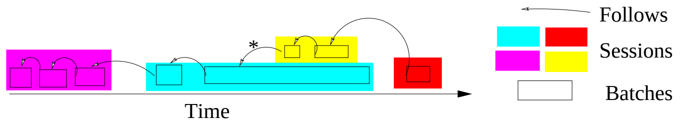


Fig. 2. Illustration of the follows relation.

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.

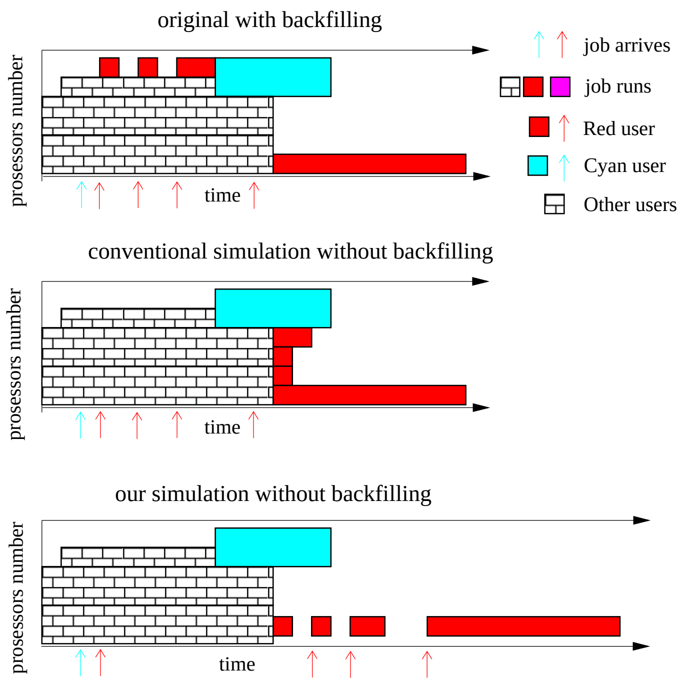


Fig. 3. Top: example schedule on a system that uses backfilling. This workload is then used to simulate a FCFS scheduler in a conventional simulation (middle) and a feedback based simulation (bottom).

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.

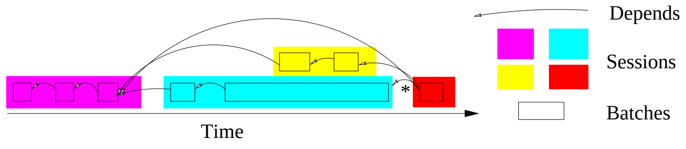


Fig. 4. Illustration of the depends on relation.

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

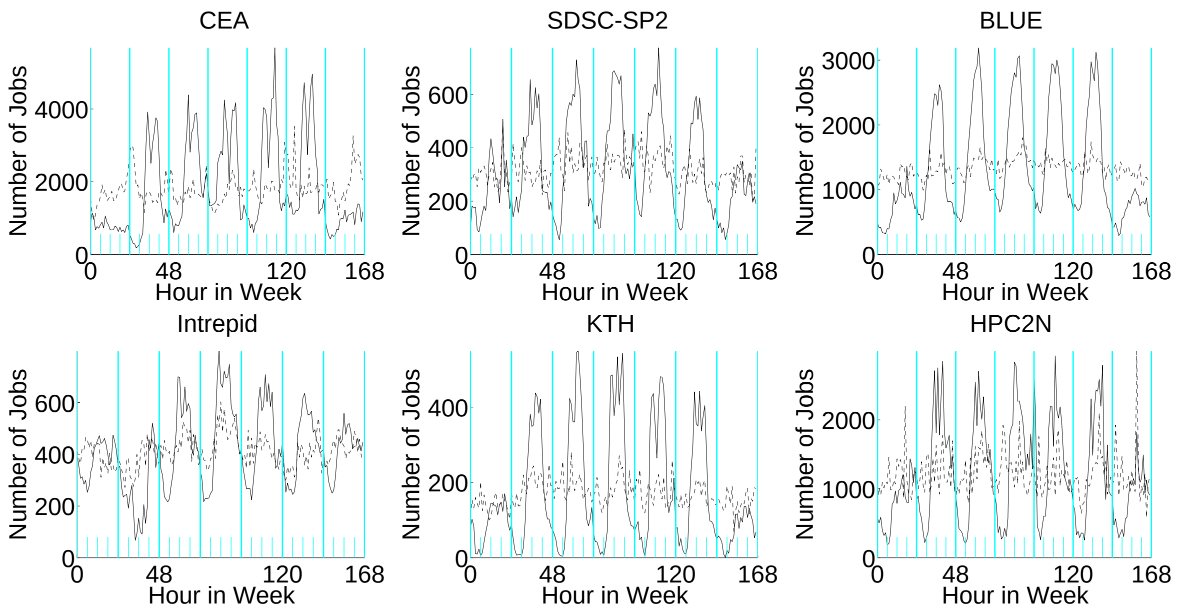


Fig. 5. Distribution of arrival times per week using the adjusted user model (dashed line) compared to the original trace (solid line).

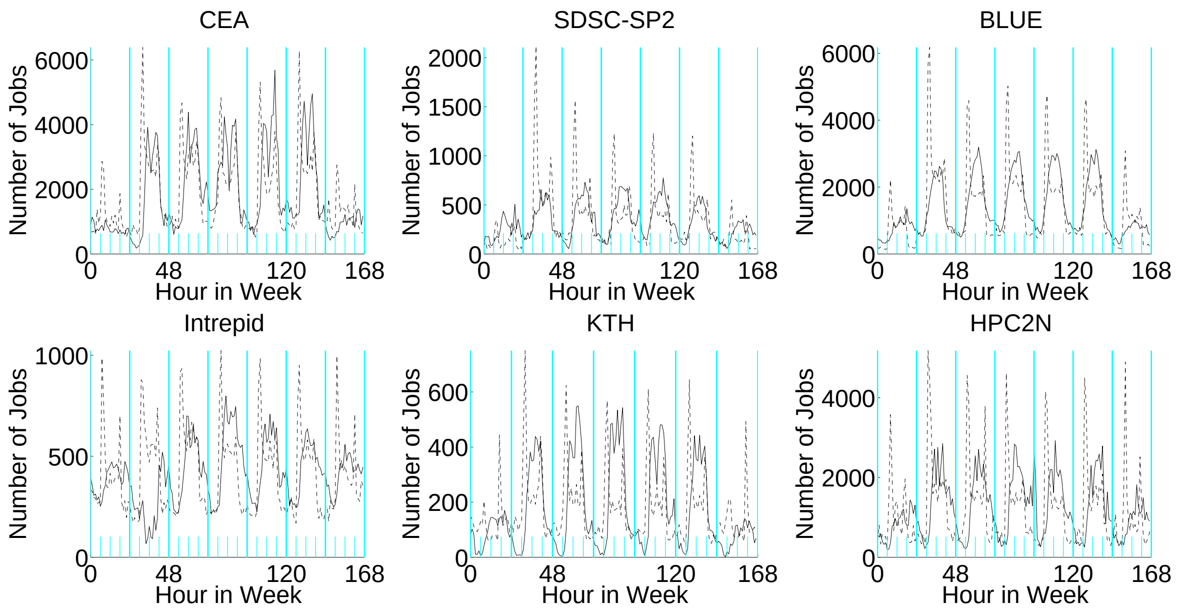


Fig. 6. Distribution of arrival times per week using the distribution based user model (dashed line) compared to the original trace (solid line).

| 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

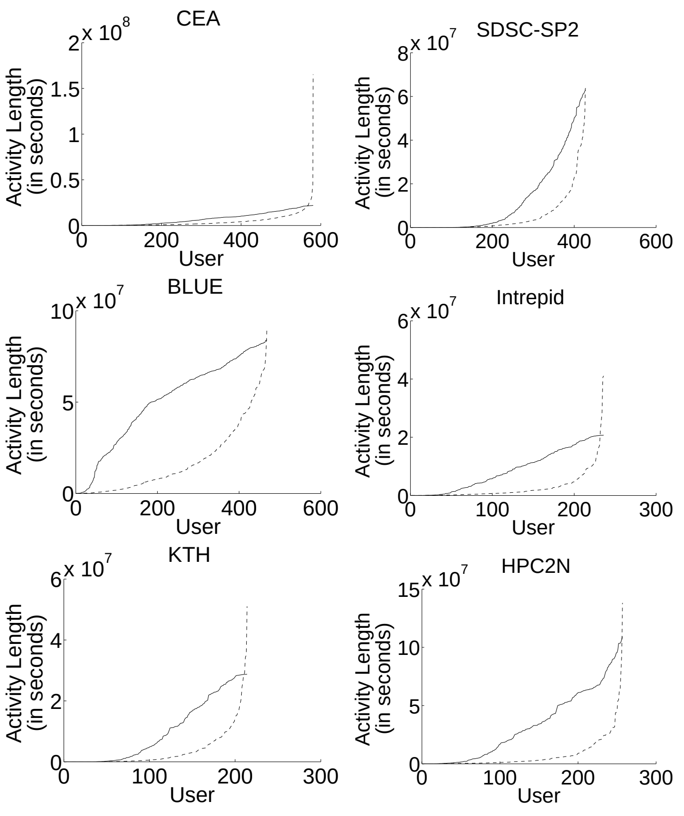


Fig. 7. Distribution of the activity length per user using the distribution based user model (dashed line) compared to the original trace (solid line).

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.

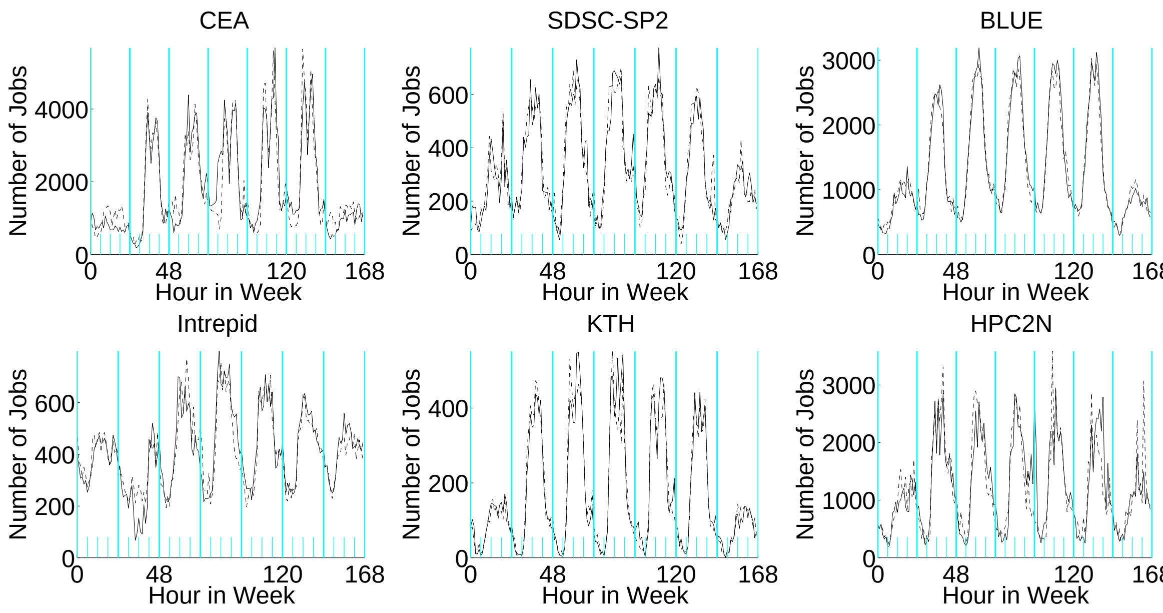


Fig. 8. Distribution of arrival times per week using the fluid user model (dashed line) compared to the original trace (solid line).

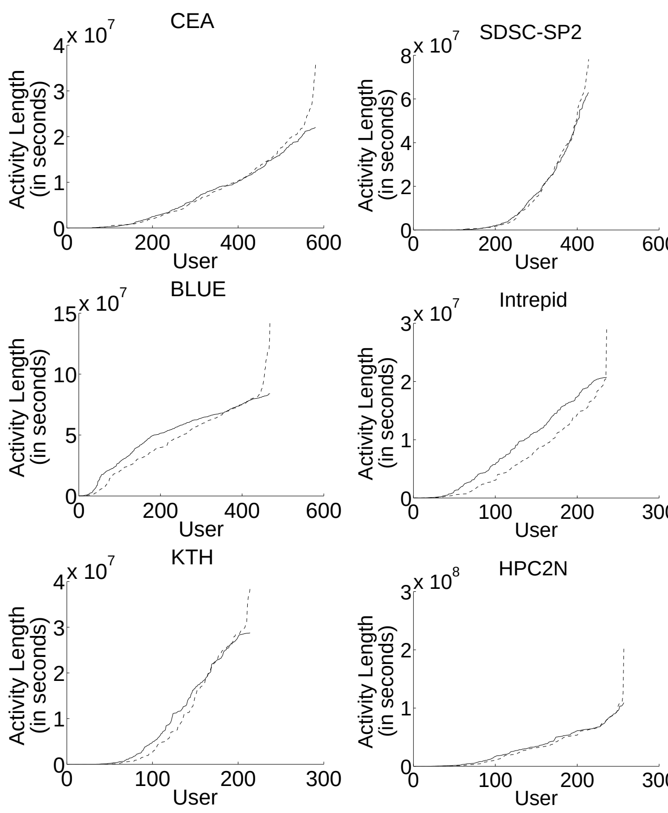


Fig. 9. Distribution of the activity length per user using the fluid user model (dashed line) compared to the original trace (solid line).

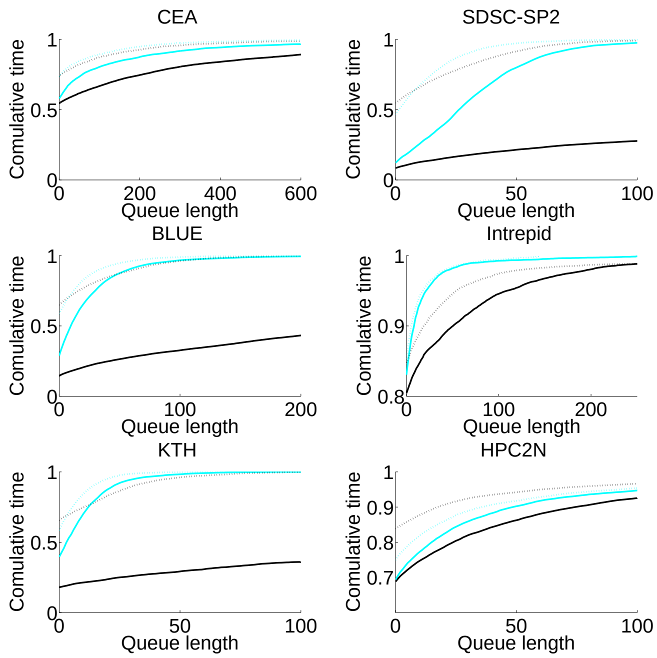


Fig. 10. Distributions of queue lengths during simulations: conventional simulation (solid line) is compared with feedback-based simulation with fluid user model (dashed line), using the EASY (Cyan) and FIFO (Black) schedulers.

## 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.

# Chapter 4

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

**Published**. In 22nd Conference on Modeling Analysis and Simulation of Computer and Telecommunication Systems (MASCOTS), pp. 51-60, Sep 2014.

Sections:

# Chapter 5

# Semi-Open Trace Based Simulation for Reliable Evaluation of Job Throughput and User Productivity

**Published**. In 7th IEEE International Conference on Cloud Computing Technology & Science (CloudCom), pp. 413-421, Nov 2015.

Sections:

# Semi-Open Trace Based Simulation for Reliable Evaluation of Job Throughput and User Productivity

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*—**New scheduling algorithms are first evaluated using simulation. In these simulations, the workload has a huge influence on the measured performance of the simulated system. Therefore, it is customary to use workload traces recorded previously from real systems. Such open-system simulations preserve all the jobs' properties. However, preserving the jobs' arrival times actually destroys the logic of the user's workflow, especially dependencies and think times between successive jobs. Furthermore, performance in such simulations is measured by the average wait time and slowdown, under the fixed load and throughput conditions dictated by the trace. Therefore, it is impossible to evaluate the system's effect on throughput and productivity.**

**As an alternative we propose semi-open trace based simulations that include dynamic user activity and internal feedback from the system to the users. In these simulations, like in a real system, users adjust their job-submittal behavior in response to system performance. As a result, the simulations produce different loads and throughputs for different scheduling algorithms or parametrizations. We implemented such a simulation for evaluating the schedulers of parallel job systems. We also developed a novel user-aware scheduler designed specifically to increase users' productivity. While conventional simulations cannot measure this scheduler's influence reliably, and would suggest it is useless, our simulation evaluates it realistically and shows its beneficial effect on the users' productivity and the system's throughput.**

## I. Introduction

Scheduling and resource management supporting the execution of jobs on large scale systems — such as supercomputers, clusters, and clouds — is a challenging endeavor. In such systems one cannot just put a new design into production without first extensively evaluating it. This leads to the challenge of faithful evaluations, which reflect all the complexities of the real system. Our focus is on one aspect of this complexity, which is analysing and simulating correctly the users' behavior and and their interaction with the simulated system.

Simulations are widely used in order to evaluate the performance of new system designs. These simulations are usually driven by traces recorded from a real system. Using a real trace is an attempt to create the most realistic simulations possible, which will lead to reliable evaluation results. In particular, using trace data directly retains all the structure that exists in the workload, including locality and correlations between different workload attributes, which might be lost if a statistical model was used instead. For example, traces of real parallel jobs available from the Parallel Workloads Archive [10] include data on job arrival times, parallelism, and runtime.

Note, however, that using the trace data as is implies an open-system model in the simulation. In other words, the arrival process is preordained, and is not affected by the state and performance of the simulated system. As a result the system throughput is dictated by the trace being used, so the simulation cannot be used to measure user productivity (where productivity is assumed to be related to the rate of running additional jobs). Consequently it also cannot be used to evaluate designs intended to promote productivity or satisfy certain service-level agreements, for example user-aware schedulers [21]. Moreover, the preordained arrivals exclude a faithful replication of user feedback effects, which are especially important in cloud system workloads. Finally, each trace can only provide a single load data point to the evaluation.

To overcome these limitations Shmueli suggested a new simulation methodology for evaluating parallel jobs schedulers [19], [21]. This methodology includes a feedback loop, in which the arrival of additional jobs depends on the termination of certain previous jobs submitted by the same user. He also suggested an algorithm to model the think times between jobs (the intervals from the termination of one job to the submittal of the next one) and the possibility of aborting sessions if performance is bad. This methodology solves the problem of predefined load, and facilitates evaluations where the achieved throughput (and hence productivity) in the simulation can change. But it had the drawback of using synthetic users, and sampling the attributes of the jobs from distributions. This is not consistent with our goal to have a workload which is as close to reality as possible.

In previous work we have shown how to extend this and integrate it with trace-based simulations, by extracting dependencies from the trace [27]. Given the dependencies, we can now change the arrival times in the simulations to reflect the users' reactions to system performance, while preserving all other properties of the recorded trace. We can also combine this with resampling, which allows us to generate multiple similar workloads from the same trace, and also to manipulate the load by modifying the number of users [26]. Combining resampling and feedback together leads to Trace-Based Users-Oriented Simulations (TBUOS), which achieves the dual goal of preserving as much detail as possible from the trace and exhibiting realistic user behavior adjusted to the conditions during the simulation.

To demonstrate the importance and effect of TBUOS we suggest the User Priority Scheduler (UPS), which attempts to prioritize jobs belonging to users who are expected to be active on the system. This departs from the more common approach of prioritizing jobs according to their individual circumstances, e.g. how long they have been waiting in the queue. Evaluation with TBUOS shows that this scheduler can indeed improve system throughput and thereby also user productivity.

The contributions of this paper beyond previous work are:

- A new methodology for performance evaluation. TBUOS is a novel simulation which is the first to achieve a comprehensive realistic workload with controlled load (via resampling) and throughput adjustment (via dependencies and feedback). Previous work had suggested each of these components in isolation.
- Advanced understanding of the interplay between load, performance, and throughput, e.g. how throughput depends on the user population and not only on feedback.
- A new user-aware scheduler called UPS. This is conceptually similar to the CREASY scheduler proposed by Shmueli, but introduces the innovation of prioritization by users instead of treating jobs individually. This increases the level of coordination and reduces the risk of spreading resources too thinly across competing users.

Being the first framework to combine resampling with a simulation of the workflow behavior of users, our suggested approach cannot be expected to be the last word. Our focus is on showing an example of a semi-open simulation, and demonstrating the capabilities of such a simulation. This is especially important for cloud systems, which serve both batch workflows and interactive server workflows.

In the next two sections we review related work and our previous work on resampling and feedback. Section IV then explains how to combine feedback and resampling and shows simulation results. This is followed by Section V, which introduces UPS in detail and evaluates its performance, and by the conclusions.

## II. Related Work

This paper touches upon three distinct concerns: the workloads used to evaluate parallel job schedulers, the simulation methodology, and the policy considerations of the schedulers.

In terms of workloads, the two most common approaches have been to replay job traces directly, or else to create statistical models based on job traces. Models facilitate the creation of multiple similar workloads, potentially with controlled variations such as different loads, but they suffer from not necessarily including all the important features of the real workload [9], [2]. Resampling (explained later) improves the representativeness of evaluation workloads by modeling only the parts that need to be manipulated, and using real data to fill in the remaining details [26].

The simulations typically follow an open systems model, where jobs are submitted by some external population of users. Therefore job arrivals are independent of system performance and state. But a closed system model with feedback may be more realistic [11], [19], [17], [16], [14], as poor performance may delay the submittal of additional jobs until previous ones have terminated, and perhaps even discourage users and cause them to submit fewer additional jobs. Such effects have been discussed in several different areas. In a database context, Hsu et al. claim that replaying timestamps from a trace loses feedback [13]. Ganger and Patt recognize the influence of the lack of feedback in simulations of storage subsystems as part of a larger system, and suggested giving higher priority to critical requests even if this degrades the performance of the storage system by itself [12]. In evaluations of networks, the feedback is important in shaping the traffic. For example, TCP congestion control is highly dependent on current conditions, so using traced timestamps for packets in simulations is wrong [23], [11]. Also, several papers dealt with a human user's feedback effects on the performance of applications [15], [22], [16], [24]. For example, Yang and Veciana modeled users that aborted their downloads due to poor performance. Synchronous protocols were shown to naturally throttle load due to feedback [18], [1].

In order to include feedback in evaluations one needs a model of how users react to load. While direct experimental evidence is rare [8], some works have considered user tolerance of delays and bandwidth limitations (e.g. [5], [15], [22], [24]). Shmueli used synthetic workloads with a feedback model and a user-aware scheduler designed to exploit this feedback effect [19], [20]. Our work extends those results in two significant ways. First, we show how to conduct more realistic simulations (TBUOS) using both feedback and resampling. This retains all the structure that exists in workload traces, and avoids any potential over-simplifications that may exist in synthetic workloads. Second, we propose a user-aware scheduler (UPS) that is not based directly on the feedback model being used. This generalizes the results and eliminates the risk that they depend on prior knowledge.

A parallel job contains multiple processes, which need to run in parallel on distinct (possibly virtual) processors. The scheduler allocates processors to such jobs and then they run with no preemptions. A common scheduling approach is EASY, which serves jobs in arrival order and uses backfilling (taking jobs from the back of the queue to fill in holes in the schedule) to pack jobs more tightly and optimize response time and slowdown. The UPS scheduler, based on EASY, has a similar basic structure to Shmueli's CREASY scheduler [21], but introduces user-based prioritization.

Scheduling on cloud platforms is a relatively young field of research. Most papers focus on economic aspects and the matching of requirements to resources and only a few tried to simulate and improve their performance. For both purposes, it is important to take into consideration the feedback and interaction with the users' workflows. The jobs order in a cloud workflow is predefined and preserving it is essential in order to produce the required results. Therefore Di et al. have conducted extensive trace-based analyses of workloads,

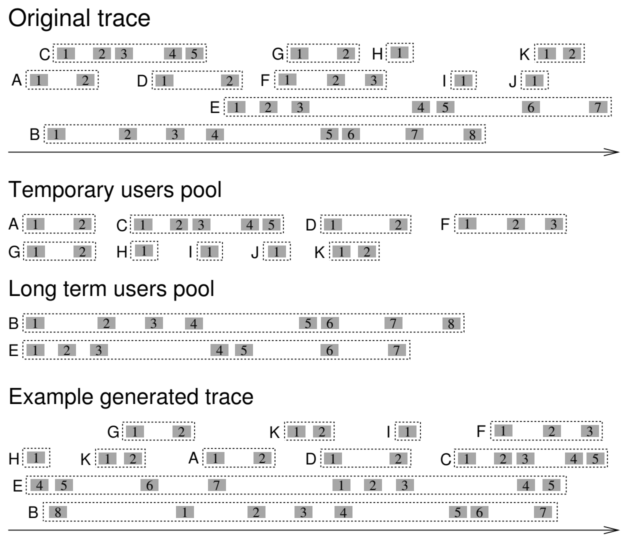


Fig. 1. Illustration of the resampling process.

and created the GloudSim trace-driven simulation environment [7], [6]. While common simulations such as CloudSim and GloudSim ignore dependices, some suggest a new simulation methodology that enforces depedency preservation [4], [3]. This methodology contains a similar mechanism to ours to enforce the preservation of dependencies, but it assumes the dependencies are known; we need to deduce dependencies from a recorded trace.

## III. Background

TBUOS is based on a combination of resampling and feedback. For completeness, we review these two mechanisms.

### A. Workload Resampling

Replaying a trace provides only a single data point of performance for one workload, while in evaluations several related workloads may be needed, e.g. to compute confidence intervals. Resampling is a way to achieve this [26]. It is done by partitioning workload traces into their basic independent components, and re-grouping them in different ways to create new workloads. Thus resampling combines the realism of real traces with the flexibility of models.

In the context of parallel job scheduling resampling is done at the level of users. We first partition the workload into individual subtraces per user. We then sample from this pool of users to create a new workload trace.

Analysis of real traces suggests the identification of two types of users. Temporary users are those that interact with the system for a limited time, for example while conducting a project. After a short duration they leave the system never to return. Long-term users, in contrast, appear to be active during much of the trace, and may be expected to have been active before logging started, and to send more jobs also after the end of the recording period.

During simulation we handle the dynamics of these user types differently. The simulation is built in units of one week, so as to preserve not only daily but also weekly cycles of activity. In the beginning of each week, a random number of temporary users is chosen such that the accumulated number will be distributed around the original average number of temporary users. The arrival times of the jobs of these users are shifted so that their first recorded job will arrive in this week. A user may be chosen multiple times or zero times.

Long term users are assumed to be active continuously. To create a variation between simulations, we start each such user from an arbitrary week in the trace and shift all subsequent jobs accordingly. If the simulation is supposed to continue after their last job has terminated, we just duplicate their whole sequence of jobs again. Note that the population of long term users is constant throughout the simulation.

An illustration of the resampling process appears in Figure 1. In this example, user $K$ was chosen twice, and the long term user $E$ was chosen to start from his 4th job, and after his 7th one we start again from the first one.

Due to the regeneration of the long term users' jobs and the addition of new temporary users each week, resampling simulations can continue indefinitely. We usually stop the simulation at a time that corresponds to the length of the original trace. Hence the number of jobs is approximately the same as in the original trace. Resampling produces workloads that are similar to the original workload [26]. However, it is also easy to induce modifications, such as extending a trace or changing its average load. Importantly, while the resampled workloads differ from the original in length, statistical variation, or load, they nevertheless retain important elements of the internal structure such as sessions and the relationship between the sessions and the daily work cycle.

### B. Simulations with Internal Feedback

Commonly used simulations are trace driven, and use an open-system model to play back the trace and generate the workload. This means that new requests get issued during simulation exactly according to the timestamps from the trace, irrespective of the logic behind the behavior of the users and of the system state. As a result 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 problem with this approach is that each trace contains a signature of the scheduler that was used on the traced system [19]. In other words, the users actions reflect their reactions to the scheduler's decisions. And real users would react differently to the decisions of another scheduler. Therefore, when we want to evaluate a new scheduling policy using 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.*

The way to integrate such considerations into trace-driven simulations is by manipulating the timing of job arrivals. In other words, the *sequence* of jobs submitted by each user stays the same, but the *submittal times* are changed [27].

Specifically, each job's submit time is adjusted to reflect feedback from the system performance to the user's behavior.

The first step in creating such a feedback based simulation is to regenerate the dependency relations between the users' batches of jobs. Session are defined based on inter-arrival times between jobs [25]. Batches are jobs within the same session that run in parallel without waiting for each other, as when a user submits a new job before the previous one had terminated. But a batch can depend on the arrival or termination of previous ones, and these constraints can be tracked and enforced. A batch then arrives to the simulated system only when it has no pending constraints.

However, a batch cannot arrive immediately when all its constraints are removed. Rather, its arrival should reflect reasonable user behavior. One possible model of user behavior is the "fluid" user model. The idea of this model is to retain the original session times of the users, but allow batches of jobs to flow from one session to another according to the feedback. To do that, we keep each session's start and end timestamps from the original workloads. Batches are given think times from the distribution of intra-session think times, but if this leads to an arrival beyond the end of the session, the batch will arrive at the beginning of the next session. If a session is skipped, it is reinserted the following week. This model creates workloads that have very similar distributions to the original [27].

## IV. Trace-Based Users-Oriented Simulation

Our goal is to develop a simulation methodology which supports features such as increasing the load and obtaining multiple data points for each setting, uses realistic workloads, and simulates the effect of a system's design reliably, *including its influence on the users' behavior* (which may result in a different throughput). In the previous section we described the two main components, namely resampling and feedback. In this section we will show how to combine them into a trace-based user-oriented simulation (TBUOS). We start with a description of the technicalities of TBUOS. Then, we explain how this leads to more realistic simulations. Finally, we analyze the dynamics of TBUOS, and show how TBUOS supports simulations where system performance affects throughput.

### *A. Integrating Resampling and Feedback*

The description of TBUOS is quite short (Figure 2). First, we preserve all the elements of resampling as they were. This includes:

1) Building the long term users and the temporary users pools.
2) Sampling a few temporary users each week (according to the original rate).
3) Regenerating the long term users when their traced activity is finished.

By not changing anything we retain all the benefits of resampling, such as the ability to extend a trace or increase the load (as was shown in [26]).

The main difference from conventional simulations using resampling is that when a user arrives to the system (whether a temporary user or the regeneration of a long term user), all his jobs are not immediately inserted into the simulation's queue of pending jobs. Rather, we use the feedback mechanism in an integrated manner within the resampling so that job arrivals will reflect the user behavior model. But the Feedback itself remains without any change. This includes:

1) Building the batches structures from the users' sub-traces.
2) Deducing the relations between the batches of each user.
3) Removing constraints as batches arrive and terminate.
4) Releasing to the simulation only the independent batches.
5) Using the user model to choose each batch's arrival time.

The combination of resampling and feedback simply means that jobs that are created by the resampling are then passed through the feedback mechanism. This divides them into batches, tracks each batch's constraints, and sends to the queue of pending jobs only the independent batches at each time step.

Temporary users in TBUOS arrive at the same average rate as in the trace, and each one has the same jobs as in the trace. When the last job of the user terminates, the user leaves the system. Therefore, each temporary user contribute the same load in TBUOS as in conventional simulations. However, due to the feedback effects, the jobs of each individual user may be spread out differently in simulated time.

The number of long term users in TBUOS is constant and equal to their number in the trace. However, unlike temporary users, they are all generated during the initialization, and they never leave the system. Instead, they are regenerated after the termination of their last job. Due to the feedback effect, the end time of the last job is not predefined but rather depends on the terminations of previous jobs. This means that the number of jobs submitted by simulated long term users will be different from the number in the original trace, depending on the performance of the system in the simulation. For example,

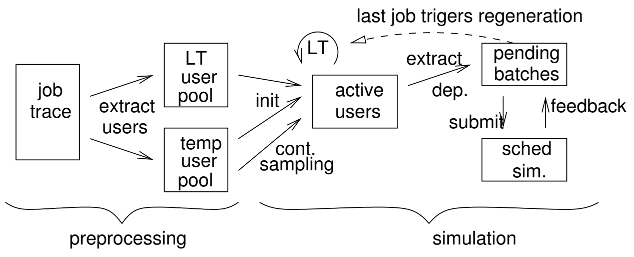


Fig. 2. TBUOS flow.

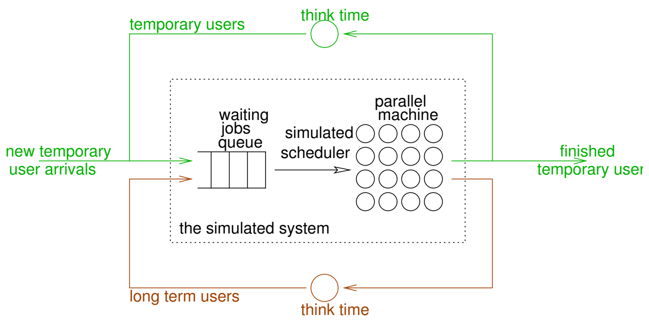


Fig. 3. TBUOS model of long term and temporary users.

if the the new design leads to better service, the user will send his next batches earlier. Therefore, the users' traced activity will be finished earlier, and the user will be regenerated and send more jobs, contributing to increased throughput.

Figure 3 illustrates the behavior of the two types of users, and shows why TBUOS is a semi-open system. The temporary users are the open part. They arrive independently of system state, send their jobs (using internal feedback between the jobs), and eventually leave the system. The long term users are the closed part. They remain in the system for the duration of the simulation. It is worth mentioning that only up to about 20% of the jobs in our workloads belong to temporary users, and in average much less than that [26]. Therefore, a very big part of the workload comes from the closed part.

## B. *Improved Realism of Simulations*

As stated our goal is to develop a simulation methodology which supports features such as increasing the load and obtaining multiple data points for each setting, but at the same time maintains realistic workloads with all the features that exist in real traces. TBUOS achieves this by inheriting the characteristics of resampling and feedback. For example, each user submits the same jobs in the same sequence with the same dependencies between them, as in the original workload [26], [27].

However, resampling by its very nature does mix up the users from the original trace. Thus user activity that was originally performed when the system was highly loaded may be matched up with activity of another user that was originally performed when the system was lightly loaded [19]. This is an unnatural combination. When the system is highly loaded jobs take longer to execute and therefore come at longer intervals. When the system is lightly loaded jobs run immediately, leading to a higher rate of submitting new jobs. These different behaviors are not expected to coexist at the same time.

TBUOS solves this problem by using feedback in addition to resampling. The feedback preserves all the properties of the workload (including internal dependencies and think-times between the jobs) except the arrival times of the jobs, which are adapted to the system's performance. In particular, it adjusts the job arrival rate to match the momentary conditions in the simulated system. As a result, the simulated workload in TBUOS is a more representative workload than the original traced workload *in the context of the simulated system*.

To demonstrate this effect we compare an evaluation of the EASY scheduler using conventional simulations with resampling and using TBUOS, based on traces from the Parallel Workloads Archive[1] (Figure 4). The first observation is that

[1]The traces used in this paper are from the following systems: the San Diego Supercomputer Center Blue Horizon (BLUE), the Seth cluster from the High-Performance Computing Center North in Sweden (HPC2N), the Los Alamos National Lab Connection Machine CM-5 (CM5), the Cornell Theory Center IBM SP2 (CTC), the Argonne National Laboratory BlueGene/P system Intrepid (Intrepid), the Swedish Royal Institute of Technology IBM SP2 (KTH), the San Diego Supercomputer Center DataStar (SDSC-DS), and the San Diego Supercomputer Center IBM SP2 (SDSC-SP2). Full details about these logs are available at http://www.cs.huji.ac.il/labs/parallel/workload/.

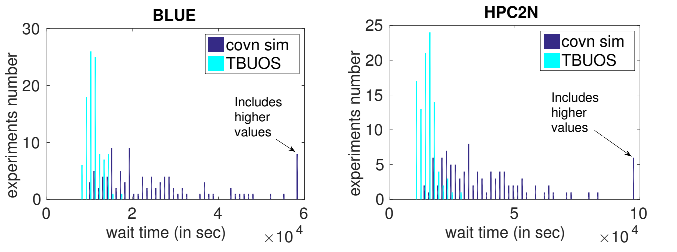


Fig. 4. Histograms of the average wait time (in seconds) in a hundred simulations of the EASY scheduler. Using TBUOS leads to lower wait times and a less skewed distribution.

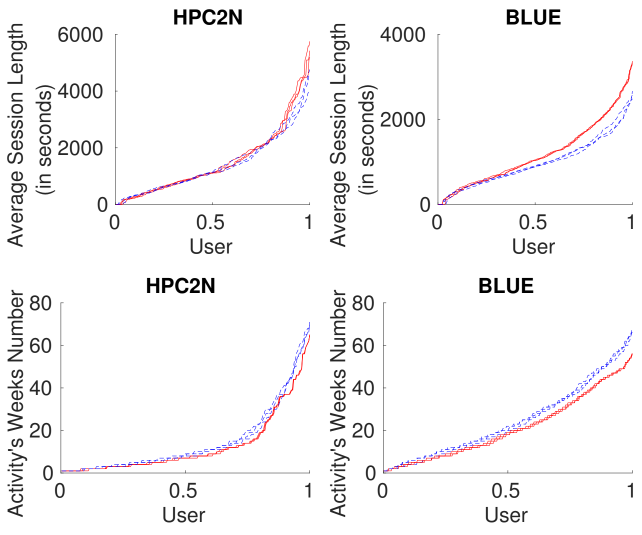


Fig. 5. Comparing user behavior in resampling (red solid lines) and TBUOS (blue dashed lines) workloads. The $X$ axis represents users sorted by the metric being shown; user IDs are normalized to [0..1] to enable easy comparison between logs with different numbers of users.

the average and maximum wait times for each log are usually much lower in TBUOS. Indeed, in some simulations the average wait time was more than 100,000 seconds, which is 27.8 hours, meaning that the average wait time came out to be more than a day. Under TBUOS there was no simulation with an average wait time of more than 8 hours, and the common values are smaller than 20,000 seconds (5.6 hours), which is much more realistic.

We conjecture that the reason for this effect is that the resampling loses the synchronization between the users as exists in reality. An interesting question is how TBUOS avoids the unrealistic long wait times. To analyze this, we compared the relevant user behavior properties, including the number of weeks in which the users were active and the average session lengths (Figure 5). In TBUOS users are active during more weeks and their session lengths tend to be shorter. This may be interpreted as evidence that in TBUOS users react to the system state, and if the system is overloaded, they send less jobs during the session and delay the next session to another time.

Another interesting difference between the distributions in Figure 4 is their shapes. In most cases the conventional

TABLE I
AVERAGE WAIT TIME MAX/MIN RATIO WITH AND WITHOUT FEEDBACK.

| Sim type | BLUE | CM5 | CTC | HPC2N | Intrepid | KTH | SDSC-DS | SDSC-SP2 |
|---|---|---|---|---|---|---|---|---|
| conv + res | 17.31 | 7.08 | 5.48 | 12.52 | 2.68 | 6.59 | 9.29 | 13.47 |
| TBUOS | 2.25 | 1.42 | 1.58 | 2.99 | 2.37 | 2.04 | 2.27 | 2.05 |

simulations with resampling create skewed distributions with a tail to high values, whereas TBUOS creates a much narrower and less skewed distribution. This is quantified using the ratio of maximal to minimal average wait time results in Table I. This reflects the same effect as above: resampling can create unrealistic load conditions, but TBUOS avoids them. To appreciate the significance of this finding, recall that these figures show the distribution of results from 100 independent simulations. If you run only one simulation, you can get any of these results. In particular, without TBUOS your simulation may happen to be from the tail of the distribution, and therefore non-representative in general. with TBUOS the danger is much reduced, because all simulations are reasonably similar.

### C. Enabling an Effect on Throughput

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. In conventional simulations, the throughput of the system being evaluated is dictated by the job arrival timestamps, instead of being affected by the actual performance of the scheduler. Even when feedback is used, this alone does not change the throughput.

It is worth mentioning that feedback-based simulations do change the throughput *per user*. In other words, the throughput of each user does depend on the simulated system's performance, which is an important step forward. But this is an uncommon performance metric, and more research about its impact is needed. The more common metric of *global* throughput is not changed by feedback alone, because the number of users and the number of jobs that each one of them submits are dictated by the trace.

TBUOS, which combines resampling and feedback, is unique in facilitating dynamically changing throughputs. The large number of long term users operate as a closed-system model, and lead to throughput that depends on the system's performance. If the scheduler allows a user to send jobs earlier, this user will actually send *more* jobs during the simulation. Therefore, in TBUOS the scheduler is able to influence the users' productivity, and the simulation will actually produce different throughputs for different system designs.

To demonstrate the ability of the scheduler to change the throughput in TBUOS we consider the simulation of a poor first-come, first-serve scheduler (FCFS) and a more effective EASY scheduler. For each simulation type (conventional+resampling or TBUOS) and scheduler (FCFS or EASY) we run a hundred simulations and tabulate the throughputs achieved. Figure 6 shows the results. Using conventional simulation with resampling, the end time and the intervals between the jobs are preserved. As a result, the distributions of throughput under FCFS and EASY are similar to each other and to the original value.

On the other hand, it is easy to see that in TBUOS the EASY scheduler led to increased throughput relative to the FCFS scheduler for all the traces. The difference testifies to the more realistic simulation of the users in TBUOS, including a logical response to a poor system performance. This results in reduced throughput with the FCFS scheduler, somtimes less than in the original trace.

An important observation regarding Figure 6's results is that the throughput with TBUOS can be quite different from the throughput of the original trace. This indicates that each trace includes a signature of the original scheduler from the traced system, and may not be suitable for the direct evaluation of other schedulers. But TBUOS compensates for this mismatch, and can even compensate for gaps in the evaluation, as actually happened to us in the following example. The CTC trace was initially simulated using a system size of 430 nodes. But later it was discovered that the correct size was 336 nodes [10], implying that the simulation had an artificially low load. With such low load good performance is easy to achieve, and when using TBUOS the throughput was increased considerably to "fill the system". Once we identified the source of the problem and used the correct size, the change in load was reduced significantly (Figure 7).

Note that TBUOS is not specifically designed to handle workloads with unrealistic low load and create a realistic simulation. For example, if the recorded trace has only one session per week, TBUOS will also have one session per week (and therefore extremely low load). However, when we use a recorded workload with realistic users' properties, if the simulated system has good performance, the users may send more jobs and use the system's resources better. This enables the evaluation to overcome the potential deficiencies of the recorded trace in the context of this specific simulation.

## V. Simulating A User-Aware Scheduler

The performance metrics used in conventional trace-based simulations are the average response time and slowdown. This led researchers to focus on the packing of jobs in the schedule as a means to improve the average values in simulation. The suggested justification is that decreasing the average wait time will improve user satisfaction and productivity. However, this reasoning is debatable, and it may even be that increased wait times correspond to higher user productivity. This can be demonstrated by a simple hypothetical example. Assume that the system is highly loaded and all the processors are being used. In this situation, using a FCFS scheduler ensures fairness, but also guarantees that all users have to wait long times for their jobs to run. This risks users becoming frustrated and aborting their sessions, thereby reducing their productivity. An alternative LIFO scheduler can perhaps reduce this effect by prioritizing the most recently active users at the expense of the users who had submitted jobs longer ago.

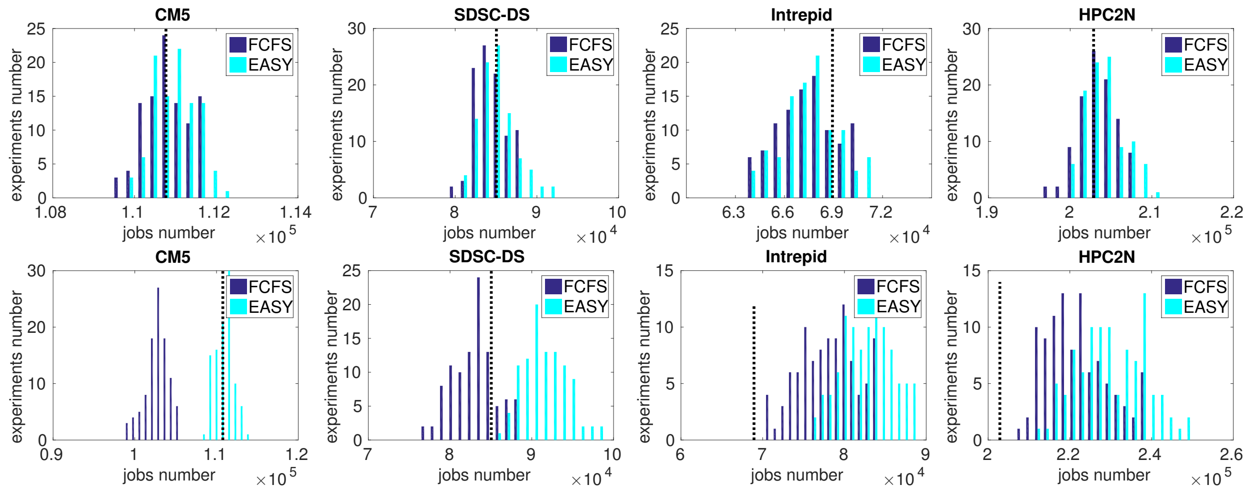


Fig. 6. Histograms of the throughput in one hundred simulations of EASY and FCFS schedulers. Top: conventional simulation (with resampling but no feedback). Bottom: TBUOS. The vertical dashed black line represents the throughput in the original workload. Clearly in conventional simulations the throughput does not depend on the system, in contrast to TBUOS in which EASY leads to higher throughputs.

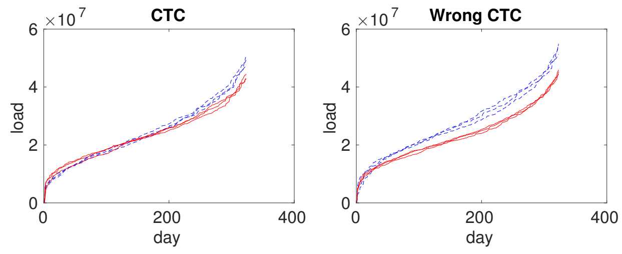


Fig. 7. Comparing the load per day (number of processor-seconds) in two versions of the CTC trace which differ only in the specified system size. TBUOS (blue dashed lines) compensates for the artificially lower load when too many nodes were specified.

More generally, user-aware schedulers try to anticipate user behavior. The scheduler's goal is then to improve productivity by facilitating more effective use of the system, allowing users to submit more jobs. This is done by trying to assess whether users are still active, and prioritizing those who have the highest probability of submitting more jobs. This may increase the average wait time, because jobs belonging to users who are thought to be inactive are delayed, but will hopefully also increase the system throughput and the productivity of active users. Evaluating such schedulers requires methodologies like TBUOS, because under conventional simulations the throughput is fixed in advance.

## *A. The User Priority Scheduler*

In this section we describe a simple new scheduler named the User Priority Scheduler (UPS). This is a user aware scheduler intended to motivate users to submit more jobs. It's general structure is similar to the CREASY scheduler suggested by Shmueli [21]. Jobs submitted to UPS are kept in a priority queue. When a scheduling decision has to be made (e.g. when some job terminates and processors become available) the highest priority job is selected for execution. The priorities are based on a weighted sum of two terms, one of which reflects the probability that a user will submit more jobs, and the other reflecting the waiting time of the job.

In more detail, the operation of UPS is based on EASY. Selecting jobs for execution is done as follows. First the queue is scanned in job priority order, and all jobs that can run (because enough processors are available) are dispatched. Then a reservation is made for the first queued job in order to ensure that it will not be starved. Finally the remainder of the queue is scanned in user priority order and jobs are backfilled provided they do not violate the reservation for the first job.

The differences from EASY are as follows. In EASY jobs are prioritized by their arrival time, and both the original scheduling and the backfilling are done in this order. In UPS the original scheduling is done according to job priorities, which reflect a weighted sum of wait time and user priority. The backfilling is done in a slightly different order: first according to user priority, and then according to waiting time for all the jobs of each user.

Given a job $J$ submitted by a user $u$ its job priority is calculated by the following expression:

$$Jpri(J) = \alpha_{user} \cdot Upri(u) + \frac{\alpha_{arr} \cdot J.wait}{4 \cdot (\text{SECONDS\_IN\_HOUR})}$$

$Jpri$ and $Upri$ are the job and user priorities, respectively. $Upri$ represents the scheduler's "user awareness" as explained below. $\alpha_{user}$ and $\alpha_{arr}$ are the weights of the two terms, where $\alpha_{user} + \alpha_{arr} = 1$. They determine the balance between the user awareness and the waiting time. If $\alpha_{arr} = 1$ then we only consider the wait time and ignore the user, so this is similar to EASY (but with backfilling order based on the user priority). If $\alpha_{user} = 1$ all the weight is placed on user awareness, at the risk of starving jobs submitted by low-priority users.

The user priority $Upri(u)$ reflects the scheduler's goal to give higher priority to users who are assumed to be active and therefore have the potential to submit additional jobs. As an initial suggestion, we consider two metrics for user activity:

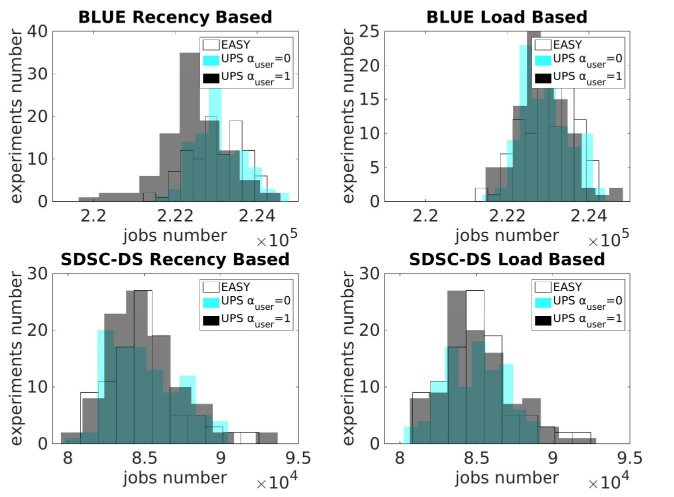


Fig. 8. Distribution of throughput results in simulations without feedback of EASY and UPS (recency based on left and load based on right) with $\alpha_{user} = 0$ or 1. The scheduler doesn't have a significant effect on the throughput.

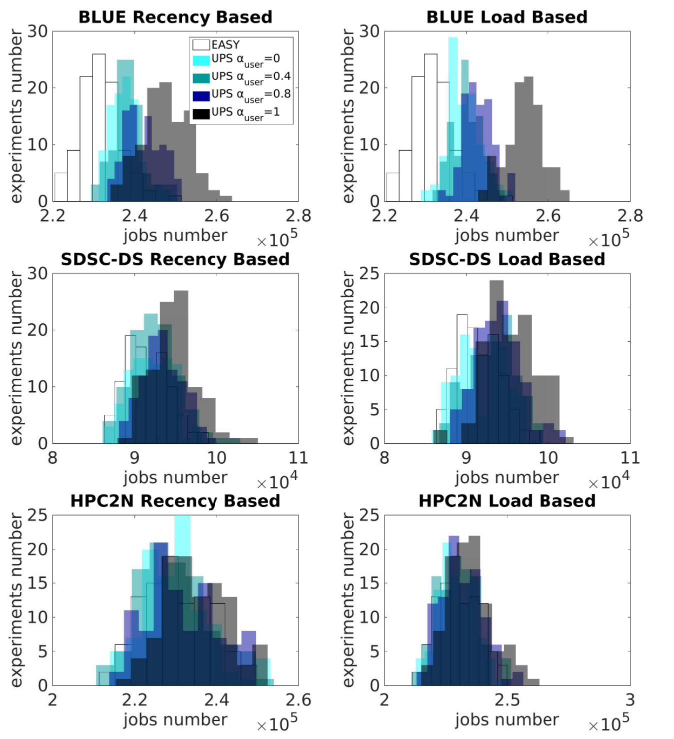


Fig. 9. Distribution of throughput results in TBOUS Simulations comparing EASY with UPS with different values of $\alpha_{user}$ (left: recency based; right: load based). Different logs lead to different effects of the user awareness, from significant differences at top, through noticeable differences in the middle, to minor differences at bottom.

1) Load based: priority is determined by the estimated work (requested runtime×processors) in waiting jobs, where more work implies lower priority. The idea is that if users have dependencies between their jobs, we will be able to run quickly the jobs of the user that needs less resources, and then he will be able to send the following jobs sooner, maybe even in the same session.
2) Recency based: priority is determined by the last job arrival from this user, where more recent activity leads to higher priority. The hypothesis of this approach is that maybe the user is still within an active session, and if he will see the results rapidly, he may decide to continue the session.

Either of these metrics can be used to define an order on users, and then we assign user $i$ in this order the priority $Upri(u_i) = 1/i$. Consequently the effective range of $Upri(u)$ is $[0, 1]$. Users with no waiting jobs are assigned $Upri(u) = 0$.

$J.wait$ is the waiting time of job $J$ in seconds, so a job's priority grows linearly with the time it waits in the queue. The dividing factor of four hours normalizes this with respect to the user priorities, such that a value of 1 is reached after 4 hours. Thus if $\alpha_{user} = \alpha_{arr} = 0.5$ a newly arrived job belonging to the highest priority user will have $Jpri = 0.5$, and any job by any other user that is waiting for 4 hours or more will have a higher priority than it. But if $\alpha_{user} = 0.2$ and $\alpha_{arr} = 0.8$ then any other job that is waiting more than 1 hour will already have a higher priority.

The UPS avoids starvation when $\alpha_{arr} > 0$ because $Upri \leq 1$, so eventually every job can become the highest priority job. It then gets a reservation due to the EASY algorithm, and subsequently gets an allocation of processors.

### B. Simulation Results

First we analyze the performance of UPS using a simulation without feedback. We compare the UPS scheduler with $\alpha_{user} = 0$ and $\alpha_{user} = 1$ for both approaches for prioritizing the users (Figure 8). The throughput does not change noticeably between the different schedulers. In particular, $\alpha_{user} = 1$ does not lead to higher throughput, which means that user awareness does not seem to provide any benefits.

However, we claim that the user aware scheduler is actually better. To demonstrate this, we show in Figure 9 the results obtained using TBUOS for different values of $\alpha_{user}$, and both approaches for prioritizing the users. We can see that with feedback UPS facilitates higher throughputs in average than EASY, and the gap grows with higher emphasis on user prioritization (larger $\alpha_{user}$). Moreover, load-based prioritization appears to be more effective than recency based prioritization. However, the improvement of using user aware schedulers actually depends on the workload. In some logs, such as BLUE, there is a very significant effect. In others, such as HPC2N, there is only a minor effect. The logs shown in the figure were selected to demonstrate the range of effects we saw with the 8 logs used.

## VI. Conclusions and Future Work

Conventional open-model trace-based simulations, which are commonly used to evaluate the performance of a new system designs, do no adjust the simulated workload to the system state. As a result they are unable to measure the true throughput with the new system design.

We suggest an alternative methodology named TBUOS. TBUOS also uses a recorded trace, and retains all the attributes of the recorded workload except one: instead of keeping job arrival times it keeps job dependencies and think-times, and adjusts the arrival times to reflect dependencies and performance. The simulation doesn't simulate only the scheduler, but also the users' behavior, namely how users would respond to the new system. As a consequence, simulations may produce different throughputs depending on the simulated system.

Moreover, scheduling algorithms which are based on affecting the input workload can't be evaluated with conventional simulations. The UPS is such a scheduler, and requires an evaluation with TBUOS to demonstrate its ability to increase the throughput.

Creating a user-oriented simulation is a new world, and there is no single correct approach for doing so. While TBUOS provides a proof-of-concept for user-based semi-open simulation, other approaches are possible. Specifically, several assumptions may be changed depending on the modeler's point of view. For example, maybe the temporary users should also have a dynamic number of jobs? Maybe some of the long term users may leave if the performance is too poor? Maybe even the jobs properties (such as the requested number of processors) should be adapted to the system?

In future work we intend to continue the development of TBUOS and UPS. More work is needed to better characterize user behavior and create better user feedback models. We also intend to consider more advanced models of the user population dynamics, including session aborts and system abandonment. Furthermore, we need to evaluate UPS more deeply, and compare it to other approaches, such as Shmueli's CREASY scheduler. Finally, A long-term goal is to implement this work also in additional domains, demonstrating this methodology to be effective in general in performance evaluation, and not only in the context of parallel jobs scheduling.

**Acknowledgment** This research was supported by the Ministry of Science and Technology, Israel.

## References

[1] V. S. Adve and M. K. Vernon, "*Performance analysis of mesh interconnection networks with deterministic routing*". *IEEE Trans. Parallel & Distributed Syst.* **5(3)**, pp. 225–246, Mar 1994, doi: 10.1109/71.277793.

[2] J. Aikat, S. Hasan, K. Jeffay, and F. D. Smith, "*Towards traffic benchmarks for empirical networking research: The role of connection structure in traffic workload modeling*". In 20th *Modeling, Anal. & Simulation of Comput. & Telecomm. Syst.*, pp. 78–86, Aug 2012, doi: 10.1109/MASCOTS.2012.19.

[3] M. A. Amer, A. Chervenak, and W. Chen, "*Improving scientific workflow performance using policy based data placement*". In 22nd *Policies for Distrib. Syst. & Netw.*, pp. 86–93, Jul 2012.

[4] W. Chen and E. Deelman, "*WorkflowSim: A toolkit for simulating scientific workflows in distributed environments*". In 8th *IEEE Intl. Conf. E-Science*, pp. 1–8, Oct 2012.

[5] 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, 2002, doi:10.1007/3-540-45798-4_8. Lect. Notes Comput. Sci. vol. 2459.

[6] S. Di and F. Cappello, "*GloudSim: Google trace based cloud simulator with virtual machines*". *Software — Pract. & Exp.* 2015, doi: 10.1002/spe.2303.

[7] S. Di, D. Kondo, and F. Cappello, "*Characterizing and modeling cloud applications/jobs on a Google data center*". *J. Supercomput.* 2014, doi: 10.1007/s11227-014-1131-z.

[8] 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.*, art. no. 10, Jun 2007, doi: 10.1145/1281700.1281710.

[9] D. G. Feitelson and E. Shmueli, "*A case for conservative workload modeling: Parallel job scheduling with daily cycles of activity*". In 17th *Modeling, Anal. & Simulation of Comput. & Telecomm. Syst.*, Sep 2009, doi:10.1109/MASCOT.2009.5366139.

[10] D. G. Feitelson, D. Tsafrir, and D. Krakov, "*Experience with using the Parallel Workloads Archive*". *J. Parallel & Distributed Comput.* **74(10)**, pp. 2967–2982, Oct 2014, doi:10.1016/j.jpdc.2014.06.013.

[11] 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.

[12] 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.

[13] 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.

[14] 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.

[15] 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.

[16] 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.

[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] 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)**, pp. 385–398, Oct 1990, doi:10.1109/71.80178.

[19] 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.

[20] 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.

[21] 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.

[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] W. Willinger, V. Paxson, and M. S. Taqqu, "*Self-similarity and heavy tails: Structural modeling of network traffic*". In *A Practical Guide to Heavy Tails*, R. J. Adler, R. E. Feldman, and M. S. Taqqu (eds.), pp. 27–53, Birkhäuser, 1998.

[24] 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.

[25] 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, 2012, doi:10.1007/978-3-642-35867-8_12. Lect. Notes Comput. Sci. vol. 7698.

[26] N. Zakay and D. G. Feitelson, "*Workload resampling for performance evaluation of parallel job schedulers*". *Concurrency & Computation — Pract. & Exp.* **26(12)**, pp. 2079–2105, Aug 2014, doi: 10.1002/cpe.3240.

[27] N. Zakay and D. G. Feitelson, "*Preserving user behavior characteristics in trace-based simulation of parallel job scheduling*". In 22nd *Modeling, Anal. & Simulation of Comput. & Telecomm. Syst.*, pp. 51–60, Sep 2014, doi:10.1109/MASCOTS.2014.15.

# Chapter 6

# Models for Evaluating Throughput

**Unpublished**.

Sections:

# Models for Evaluating Throughput

Netanel Zakay Dror G. Feitelson
Department of Computer Science
The Hebrew University, 91904 Jerusalem, Israel

***Abstract*—Analysts are interested in two categories of performance metrics: those concerned with time (response or wait time), and those concerned with rates (throughput or utilization, which reflect productivity and how well resources are used). In principle, these two categories are independent of each other, and both should be evaluated. But a common mistake is to "measure" the throughput in open-system evaluations, where the throughput is actually dictated directly by the workload. In order to evaluate throughput, the system model *must* include a feedback loop which modulates the workload being processed. The common solution is to create a pure closed system with a fixed number of users, who submit jobs in a loop. However, such behavior is often unrealistic. We review and analyze two alternative models that provide the required feedback by combining open and closed components: the *mixed model* which includes two such job classes, and the *re-open model* in which open user arrivals are combined with performance-dependent closed repetition of jobs by these users. These models allow evaluations of the trade-off between response time and throughput, including the throughput as it is observed by each user.**

***Index Terms*—throughput; utilization; mixed open-closed model; re-open model.**

## I. Introduction

Evaluating the performance of a system is a major part of system design. Reliable evaluations of a proposed system are expected to lead to better designs and reduced expenses, by considering multiple options, evaluating them, and choosing the best. Therefore, when a new system design is proposed or when we want to improve a current system, it is common to evaluate it before implementing it.

There are two main categories of system performance metrics. The first category includes the response time and the wait time. These capture the delays that a job suffers in the system. The assumption is that users are more satisfied with shorter delays. The second category includes the utilization and the throughput. These capture how much of the system's resources are used and the rate at which the system serves requests. The assumption is that higher utilization and serving more requests are better. In many systems a tradeoff is involved: higher throughputs lead to higher response times.

Importantly, the two categories are not just different manifestations of the same effects. They can change independently of each other, so both should be evaluated. But commonly used performance evaluation approaches such as trace-based simulation and open-system queuing analysis evaluate only the wait time and the response time. Measuring the throughput or utilization using these methodologies can only serve as a sanity check, because the throughput and utilization (which are linearly related in expectation in this case) are completely determined by the workload. In other words, these common evaluation methodologies effectively treat throughput and utilization as an input and not as part of the evaluation.

The classic approach to evaluate throughput is by using a closed model. But a pure closed model with a fixed number of users is unrealistic for many types of system. In many cases a more realistic workload scenario combines the closed behavior with an open behavior. For example, users may arrive randomly as in an open system, but then they may execute a workflow of multiple jobs that depend on each other as in a closed model. Trace-based simulations that preserve all the jobs' properties, including their arrival times, actually destroys the logic of the user's workflow, specifically the dependencies and think times between successive jobs. An better alternative is therefore to preserve the dependencies, and adjust the arrival times [14]. But still, if all the jobs in the workflow are eventually performed, the total amount of work and hence the system throughput are the same as in the original trace. The only thing that may change is the *per-user throughput*, because the workflows of individual users may be spread out differently. This motivates adding per-user throughput to the set of metrics that should be evaluated.

But in real life there do exist situations where the total throughput is indeed affected. To capture this one needs a system model that includes an explicit effect on the number of jobs processed by the system. This can take one of two forms. The first model is systems that use admission controls to throttle their users or just drop superfluous jobs (e.g. [2], [12], [1]). A good metric of performance in this case is the number of jobs that are rejected. The second model is users who change their behavior in response to system performance. For example, it is easy to envision users who become frustrated with poor performance and reduce their activity. This is the type of models we address here.

Importantly, such models allow analysts to assess the impact of system designs on throughput even when the system does not address this explicitly (e.g. it does not employ admission controls). In particular, they facilitate an analysis of the tradeoff between throughput and response time, and the identification of situations where higher response times are actually beneficial because they correlate with higher throughput and utilization. Moreover, such models may capture negative feedback effects as when users back off from an overloaded system and thereby prevent its saturation. Using an oblivious model — as is commonly done in trace-based simulations and queueing analyses — would miss such effects and lead to overly pessimistic results [9].

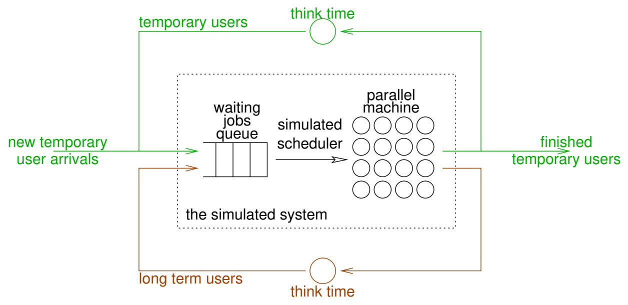


Fig. 1: *Flow of TBOUS.*

## II. TBUOS and Semi-Open Simulation

In a previous work we proposed the Trace-Based User-Oriented Simulation (TBUOS), which is a semi-open simulation that includes dynamic user activity and internal feedback from the system to the users [15]. In this simulation, we divide the users into two groups (Figure 1). One is temporary users that arrive to the system at a fixed rate, and are active for only a limited time. The other is long term users who are always active. This is modeled in simulations by sending their first traced job again after the termination of their last traced job.

We found that the overall throughput of a TBOUS simulation can differ from that of a conventional simulation based on the same trace, as a result of subtle interactions between the users. The temporary users each have a fixed number of jobs, so in the long run their contribution is fixed. But the long term users are coupled to them because they contend for the same processing resources. So if the load caused by temporary users affects the rate at which long-term users circulate, the overall throughput is affected. The goal of the present paper is to further investigate this effect, and compare it with another model where all users are temporary but their number of jobs is performance-dependent.

More formally, the two models we will be investigating are as follows:

- **The mixed model**. This model includes an open workload class and a closed workload class. It is an abstraction of the workload used in the TBUOS simulations.
  1) Open users submit a single job to the system and leave. This is like in a conventional open system. Arrivals of these users are not affected by the system state, so the load they impose on the system is also not affected by the system state. In other words, they have a constant contribution to the system throughput.
  2) Closed users who submit a job, wait for it to terminate, think, submit another job, and so on indefinitely. This is just like a conventional closed system, and arrivals are naturally affected by the system state.

  We assume that the properties of the jobs, such as the distribution of run-times, are the same for both classes. The interesting issue is the interaction between the two classes. As in TBOUS, the open class affects the performance of the closed class, and this effect can lead to changes in the system throughput.
  This model represents a system that has both permanent users and temporary users. For example, a cloud service that has users that paid to host a persistent service and other users that try the system for a limited time only and then leave. This is highly relevant today due to the growing popularity of cloud systems.
- **The re-open model**. This is an open model with repeated submittal of jobs: users arrive at a given arrival rate as in a conventional open model, but once they arrive they may submit several jobs one after the other with think times in between. As we show below, if the total number of jobs (or the probability to submit additional jobs) is fixed, the system cannot affect the throughput. But if the probability to submit additional jobs depends on system performance (and specifically on the response time) then throughput is indeed affected.
  This model may be suitable for a web server or similar systems. When users surf to a site they usually send some number of requests. However, if the performance is poor, a user may get discouraged and leave the site. On the other hand, if the server is highly responsive, the users may extend their activity and send more requests.

These models are shown graphically in Figure 2. Note that in the mixed model the two user classes are distinct, and the only interaction between them is that both use the same resources on the server. In the re-open model, on the other hand, there is only one user class, and each user either submits additional jobs (closed behavior) or departs (open behavior) with some probability.

An important issue in both models with whether the workload is assumed to be interactive or batch. In batch workloads closed jobs are submitted one after the other with no intervening think time. As a result the utilization is always 100% and the throughput equals the system's capacity (namely the maximal number of jobs that the system can serve in a unit of time). In this case the only metric that can change is the throughput observed by each user, because delays cause the users to execute the same jobs over longer periods of time.

In interactive workloads the think time throttles the rate of submitting jobs, and therefore changes to the scheduler can lead to changes in throughput. This can happen, for example, by delaying interfering jobs to non-prime time [3]. In our models we focus on interactive workloads.

## III. The Mixed Model

The mixed model in and of itself is not new, and has been used to model the combination of interactive and batch work for example (see [5, sect 7.4.3] and [6, sect 13.7]). The idea there is that the batch jobs constitute a closed system, with a new one starting immediately upon the termination of a previous one. Previous works described the model briefly and how to calculate the performance metrics. To this we add the motivation of affecting throughput, and present graphs that

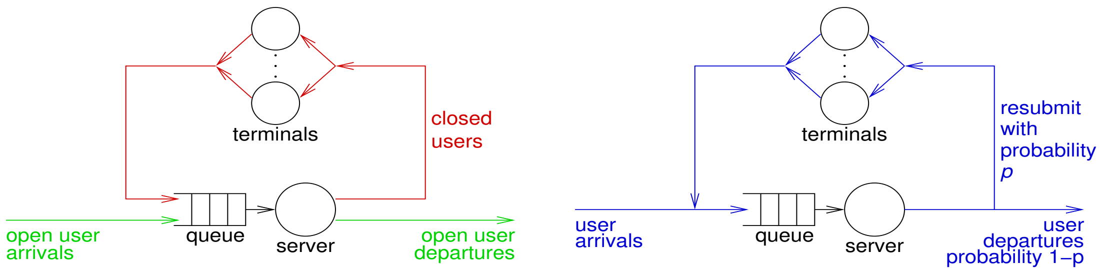


Fig. 2: *The mixed model (left) and the re-open model (right).*

show the relation between the inputs and the performance metrics. Also, we analyze this model using operational analysis, to show both intuitively and mathematically these relations between the performance metrics.

The Mixed Model is an abstraction of the model we used in simulations in the context of suggesting the use of workload resampling from recorded logs and TBOUS [13], [15]. As explained above, the two workload classes were long-term users who are active throughout and behave like a closed model, and temporary users who are active only for a limited duration, thus behaving like an open model (or rather, like a re-open model) in that their arrivals — and consequently also all the load they impose — is uncorrelated with system state.

## A. *Model and Dynamics*

Assume a single server system, where arriving jobs queue, receive service, and depart. The parameters of the model are as follows:

| | | |
|---|---|---|
| open part | $\lambda$ | arrival rate |
| closed part | $N$ | number of users |
| | $Z$ | think time |
| system params | $S$ | service demand |
| | $\mu$ | service rate ($\frac{1}{E[S]}$) |

This model assumes that all the jobs are similar, meaning that they have the same service demand $S$. Also, the scheduler does not differentiate between jobs submitted by the open users and the closed users.

Given the mixed workload, the system dynamics will evolve as follows. The open component of the workload is oblivious to the system state. It therefore imposes a fixed load of $\lambda E[S]$ per job. This has to be less than the system capacity, implying the common requirement $\lambda < \mu$.

Once the capacity taken up by the open component is acknowledged, the remaining capacity is $(\mu - \lambda)E[S]$. The closed part adjusts to fit in this left over capacity. The mechanism that affects this adjustment is the response time. The lower the system-wide response time, the sooner the closed users submit additional jobs, thereby increasing the load. But if the load is too high the response time will grow, thereby delaying the closed users, and subsequently delaying the submittal of additional jobs and reducing the load. This is a stabilizing negative feedback effect on the throughput $X$.

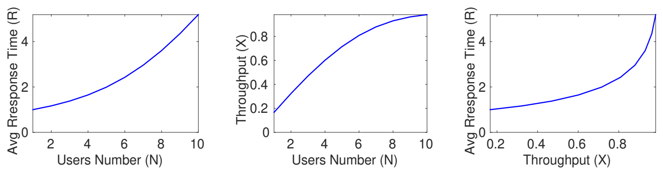


Fig. 3: *The average response time and throughput of the closed submodel (MVA results) with $\mu = 1.0$ and $Z = 5.0$. The first two graphs show how the number of users $N$ affects the average response time and the throughput. The third graph combines these results to show the average response time as a function of the throughput (or equivalently, the utilization).*

## B. *Common Evaluation*

As noted above the mixed model is well-known. For completeness, we provide a short review of the common approach to calculating the response times $R_c$ and $R_o$ (of closed and open jobs, respectively; they are not identical because open jobs arrive randomly, whereas closed job arrivals are correlated with system state, but the difference becomes negligible for large $N$).

1) The open part is oblivious to the system. Its utilization is $\lambda S$ and the throughput is $\lambda$.
2) The effect of the open part on the closed part is modeled by extending the service time to $S_c = \frac{S}{1-S\lambda}$. The closed part is then solved using the iterative MVA algorithm.
3) The average queue length of closed customers is added to the resident users of the open model to calculate $R_o$.

While this method calculates the performance metrics for the system, it doesn't provide the relations between the different metrics nor an intuitive explanation of the system's behavior. To provide this, we present some results in Figure 3. When the number of closed users increases both the response time and the throughput increase, but with different profiles: asymptotically the response time grows linearly, and throughput saturates at system capacity. The relation between them is highly non-linear, and similar to how response time depends on utilization in open systems. To understand this better, we use operational laws.

## C. *Operational Analysis*

First, we observe that the system can operate in either of two phases.

- Full utilization phase — in this phase the closed component uses all the remaining capacity.
- Partial utilization phase — in this case the closed component does not use up all the remaining capacity, because the combination of the number of users and the think time does not allow the submittal of sufficient jobs.

Let's start by analyzing the full utilization phase. The fact that the system is fully utilized means that the throughput equals to the system capacity and therefore

$$X = \mu$$

The throughput is the sum of the open throughput $\lambda$ and the closed throughput $\frac{N}{R_c+Z}$ (from the interactive response time law). Using the first equation we conclude that

$$\mu = \lambda + \frac{N}{R_c + Z}$$

From this we can extract $R_c$

$$R_c = \frac{N}{\mu - \lambda} - Z$$

Thereby characterizing the system performance using operational laws that do not require assumptions about distributions. This shows the intuitive result that (asymptotically) adding closed users leads to a linear increase in response time because they just pile up in the queue.

The partial utilization phase implies that

$$\mu > X = \lambda + \frac{N}{R_c + Z}$$

and therefore

$$R_c > \frac{N}{\mu - \lambda} - Z$$

In other words, the response time $R_c$ is large enough to throttle the closed users and prevent them from creating additional load. We then have two unknowns: $R_c$ and $X$, with the relationship

$$R_c = \frac{N}{X - \lambda} - Z$$

While this doesn't allow us to solve for $R_c$ and $X$, it provides the inverse relationship between them for given $N$ and $Z$. But if $N$ grows $X$ grows too, giving the result in Figure 3.

## IV. The Re-Open Model

The basic elements of the re-open model were introduced by Schroeder et al. under the name "partly-open" [8] and used by others [7], [4]. Specifically, this combined open and closed elements by mandating that users submit additional jobs with a probability $p$. However, *this alone does not affect throughput*. To affect throughput we add the new condition that $p$ be dependent on the system state.

The parameters of the model are as follows:

| | | |
|---|---|---|
| user params | $\lambda$ | arrival rate |
| | $p$ | probability to submit another job |
| | $Z$ | think time between submitted jobs |
| system params | $S$ | service demand |
| | $\mu$ | service rate ($\frac{1}{E[S]}$) |

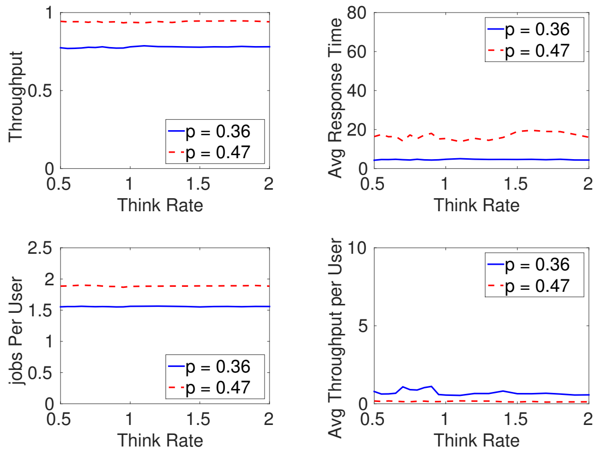


Fig. 4: *Simulating the re-open model for 10,000 users to check the impact of the "think rate" $\frac{1}{Z}$ on performance. Simulations use $\mu = 1.0$ and $\lambda = 0.5$, with $p = 0.36$ (solid) and $p = 0.47$ (dashed). These $p$ values produce high enough load to be interesting, while the system is still stable.*

In the following subsections we analyze this model under different assumptions. To examine this model's characteristics, we created a simulation that simulates the re-open model with a basic FCFS scheduling policy.

### A. Constant $p$ Value

When $p$ is constant, this model is a generalization of both the open and closed conventional models. A conventional open model is obtained when $p = 0$, and a conventional closed model is obtained when $p = 1$ and $\lambda = 0$.

If $0 < p < 1$ then each user submits a number of jobs and then leaves. As defined above with constant $p$ the throughput $X$ is fixed by the model, and it is essentially like an open system: each user submits $1 + p + p^2 + p^3 + \ldots = \frac{1}{1-p}$ jobs in expectation, so the throughput is $X = \frac{\lambda}{1-p}$. In such a model the stability constraint is

$$\lambda \leq (1 - p)\mu$$

This immediately leads to a bound on $p$, namely $p \leq 1 - \frac{\lambda}{\mu}$.

To evaluate this behavior, we modeled fixed $p$ in simulations. All the distributions including the think times, interarrival times, and service times are exponential. Figure 4 shows that the think time doesn't have an impact on any of the results, at least when the system is not too close to saturation. Figure 5 shows the impact of $p$. While throughput, utilization, and jobs per user grow moderately with $p$, the queue length and wait time grow much more precipitously, but only upon approaching saturation. And note that as the overall throughput increases, the throughput as observed by each user drops.

Finally, Figure 6 shows the relation between the response time, the throughput, and the throughput per user. Note that the throughput per user decreases dramatically for higher response times, because the users are active for much longer, but send the same number of jobs.

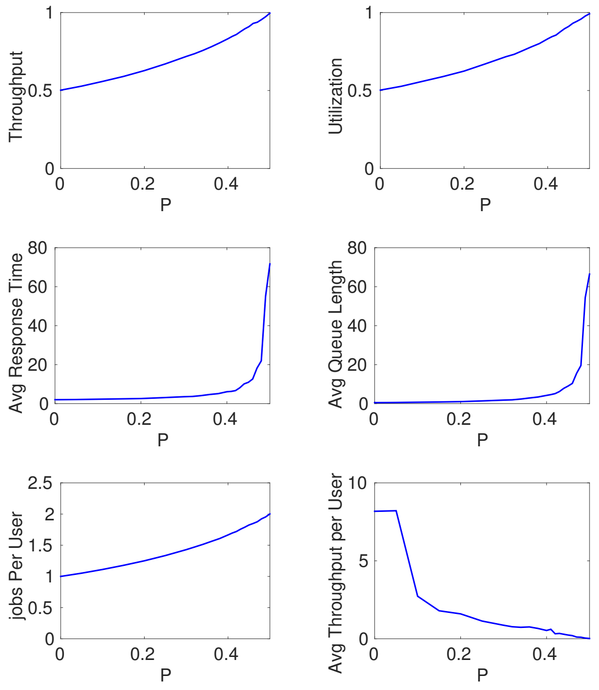


Fig. 5: *Simulating the re-open model for 10,000 users using $\mu = 1.0$, $\lambda = 0.5$, and $Z = 1$ in order to check the impact of $p$ on performance.*

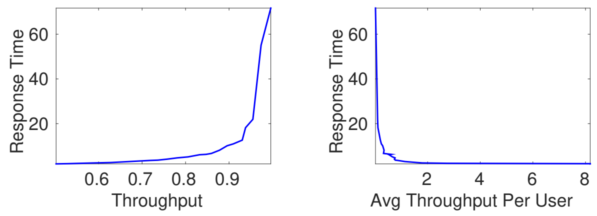


Fig. 6: *tradeoff between $R$ and $X$ in simulations of the re-open model.*

### B. $p$ as a Function of the Response Time

A more realistic model is to assume that the probability to send an additional job is not fixed, but rather depends on the performance of the system. High speed response may cause a user to extend his session with the system, while slow responses may cause him to leave the system. Therefore, a few works suggested to take the response time into account when calculating the probability of a user to send an additional job [10], [11].

This is an interesting model, because it means that the throughput is dynamic and depends on the performance of the system. Short response times will lead to higher throughput. But higher throughput will lead to more contention and thus to higher response times, and hence to reduced throughput. Therefore, the throughput and the response times balance each other, and again we have a stabilizing negative feedback effect.

To formalize the model we need to decide how $p$ depends on the response time of previous jobs. A reasonable assumption is that $p$ is a monotonically decreasing function of $r$, where $r$ is the response time of the last job of this user. We modeled two different functions for $p$:

- Exponentially decreasing function. This means that the probability to submit another job drops off exponentially with the response time of the previous job. In other words, $p = e^{-r}$. Therefore $p$ starts from one for zero response time and decreases exponentially to 0 with longer response times.
- Shmueli's model. Shmueli and Feitelson analyzed the probability of a user to continue a session for several logs in the Parallel Workloads Archive [10], [11]. They discovered that the relationship is a hyperbola

$$p = \frac{0.8}{0.05 \cdot r + 1}$$

where $r$ is in minutes. The average running time of jobs was from a couple of minutes to perhaps 10 minutes depending on the log. In our model, we use $S = 1$ as the unit of time. Therefore we used the same formula, but used $10r$ instead of $r$.

We simulated these two approaches. Figure 7 shows the resulting metrics for different $\lambda$. Note that the throughput and utilization are larger than $\lambda$ due to the job repetitions. As $\lambda$ increases, the throughput and utilization converge to 1. Thus when more users arrive, there are less resources available for

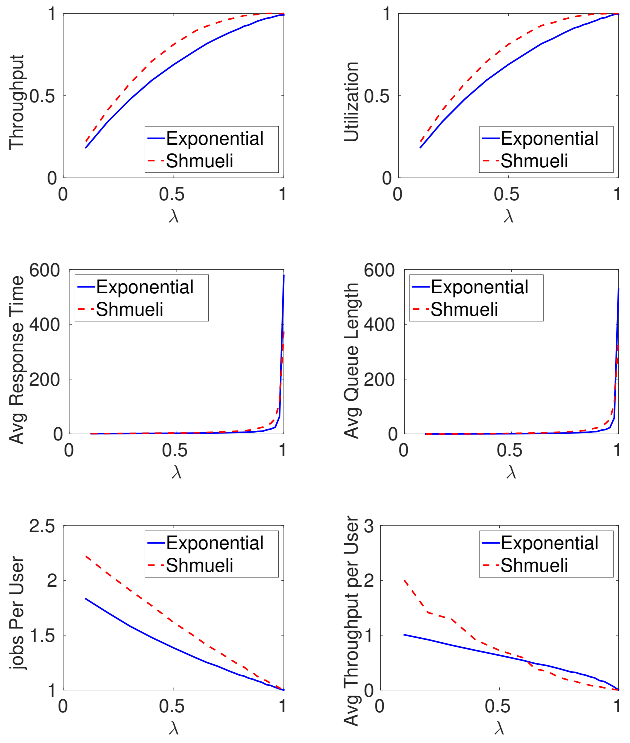


Fig. 7: *Simulating the re-open model for 10,000 users with $\mu = 1.0$ and $Z = 1$ where $p$ is exponentially decreasing with $r$ (solid) and where $p$ is set according to Shmueli's formula (dashed).*

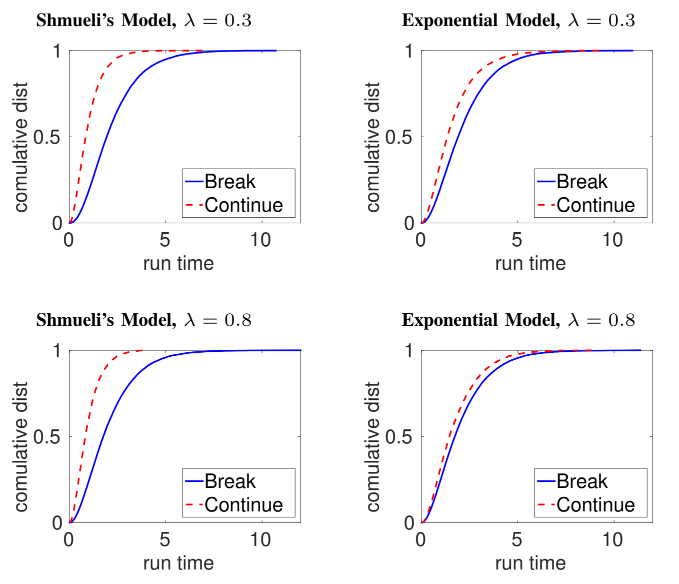


Fig. 8: *Runtime distributions of jobs after which the decision was to submit another job (continue), and jobs where the decision was to quit (break). Results for simulation of 10,000 users with* $\mu = 1.0$ *and* $Z = 1$.

running repeated jobs. Therefore response times grow and $p$ drops. Consequently users submit fewer jobs, and the system does not saturate. Also, the throughput per user drops, but much more moderately than in the constant $p$ model. And again, queue length and wait time shoot up only when the system is close to saturation.

Both approaches have different characteristics than the constant $p$ model due to the fact that a higher number of users (higher $\lambda$) leads each user to send fewer jobs. Comparing between the two approaches, Shmueli's approach starts with a higher number of jobs per user, and drops nearly linearly to 1, while in the exponential approach the slope become smaller for higher $\lambda$. As a result, the throughput when using Shmueli's approach converges to 1 for lower $\lambda$ and the average wait time is longer.

An interesting new metric we introduce is the throughput per user. This is the quotient of the number of jobs that a user submits divided by the total residence time in the system. In Shmueli's approach it starts higher, which means that users take advantage of the empty system to send more jobs, but they keep sending a high number of jobs when the system is loaded, and this leads to delays and subsequently to low throughput per user when $\lambda$ is high.

### C. Runtime Distribution Bias

An interesting artifact of the re-open model is the possible creation of bias in the distribution of job runtimes for each user. When $p$ depends on the response time of the previous job, this reflects the confluence of two factors: the length of the job itself, and how long it had to wait in the queue. So if the job was short there is an increased probability to continue with additional jobs, and if the job was long this probability is reduced — regardless of the performance of the system. As a result the sequence of jobs executed by a certain user may tend to include several short jobs and only one long job, the last one. This may affect the throughput per user metric.

Evidence for this effect is shown in Figure 8. The distribution of runtimes of jobs that were the last one for a user (meaning that the probabilistic decision was not to submit additional jobs) tends to have longer times than the distribution of runtimes of jobs that were not the last. The effect is stronger with the Shmueli model, and weaker when $\lambda$ is high, because then each user submits fewer jobs. More work is needed to decide if this is a problem or perhaps it actually reflects an effect that exists in reality.

## V. Conclusions and Future Work

In conventional trace-based simulations and open-system analyses the throughput is given, and only the response time can be evaluated. In addition there is no feedback and the system may saturate as users continue to submit jobs.

We considered two models that include realistic negative feedback effects and allow the tradeoff between response time and throughput to be explored. The mixed model includes closed users whose throughput is affected by contention from open users. In the re-open model users submit additional jobs depending on their response time. In either case, the total throughput converges to the system capacity as more users are added. Therefore a more interesting metric is the throughput per user.

Use of such models is mandatory if one wishes to assess the effect of a scheduling scheme on throughput. This is an important performance metric for schedulers that attempt to prioritize different users or affect their productivity and behavior [11], [15]. Preferring one model over the other depends on the type of evaluated system and the users' behavior in the environment.

Our next goal is to continue developing these models and make them more realistic, similarly to the TBOUS simulation [15]. One useful expansion is to combine them by adding repeated jobs based on performance to the mixed model. In effect, this creates a mixture of closed and re-open instead of closed and open.

Another interesting issue is the think time ($Z$). In both the mixed model and the re-open model results above, the think time was sampled from an exponential distribution with parameters given as an input of the model. However, in reality, the think time might depend on various factors. For example, if the response time was short, the user might still be engaged with his session and send the next job with a short delay. For longer response times, the user might break his session and send the next job later in a new session (e.g. after taking a coffee break) or even the next day (if the job finished late at night). This suggests that more elaborate user behavior models are needed [14].

### Acknowledgments

This work was supported by the Ministry of Science and Technology, Israel.

## References

[1] T. Brecht, D. Pariag, and L. Gammo, "*accept()able strategies for improving web server performance*". In *USENIX Ann. Tech. Conf.*, pp. 227–240, Jun 2004.

[2] L. Cherkasova and P. Phaal, "*Session-based admission control: A mechanism for peak load management of commercial web sites*". *IEEE Trans. Comput.* **51**(**6**), pp. 669–685, Jun 2002, DOI: 10.1109/TC.2002.1009151.

[3] D. G. Feitelson and E. Shmueli, "*A case for conservative workload modeling: Parallel job scheduling with daily cycles of activity*". In 17th *Modeling, Anal. & Simulation of Comput. & Telecomm. Syst.*, Sep 2009, DOI: 10.1109/MASCOT.2009.5366139.

[4] R. Hashemian, D. Krishnamurthy, and M. Arlitt, "*Web workload generation challenges – an empirical investigation*". *Softw. — Pract. & Exp.* **42**(**5**), pp. 629–647, May 2012, DOI: 10.1002/spe.1093.

[5] E. D. Lazowska, J. Zahorjan, G. S. Graham, and K. C. Sevcik, *Quantitative System Performance: Computer System Analysis Using Queueing Network Models*. Prentice-Hall, Inc., 1984.

[6] D. A. Menascé, V. A. F. Almeida, and L. W. Dowdy, *Performance by Design: Computer Capacity Planning by Example*. Prentice Hall, 2004.

[7] D. Pariag, T. Brecht, A. Harji, P. Buhr, and A. Shukla, "*Comparing the performance of web server architectures*". In *EuroSys*, pp. 231–243, Mar 2007, DOI: 10.1145/1272998.1273021.

[8] 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.

[9] 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.

[10] 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.

[11] 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.

[12] M. welsh and D. Culler, "*Adaptive overload control for busy Internet servers*". In 4th *Conf. Internet Technology & Syst.*, USENIX, Mar 2003.

[13] N. Zakay and D. G. Feitelson, "*Workload resampling for performance evaluation of parallel job schedulers*". *Concurrency & Computation — Pract. & Exp.* **26**(**12**), pp. 2079–2105, Aug 2014, DOI: 10.1002/cpe.3240.

[14] N. Zakay and D. G. Feitelson, "*Preserving user behavior characteristics in trace-based simulation of parallel job scheduling*". In 22nd *Modeling, Anal. & Simulation of Comput. & Telecomm. Syst.*, pp. 51–60, Sep 2014, DOI: 10.1109/MASCOTS.2014.15.

[15] N. Zakay and D. G. Feitelson, "*Semi-open trace based simulation for reliable evaluation of job throughput and user productivity*". In 7th *IEEE Intl. Conf. Cloud Comput. Tech. & Sci.*, pp. 413–421, Nov 2015, DOI: 10.1109/CloudCom.2015.35.

# Chapter 7

# Summary

In Chapter 2, we conducted a research on the correct way to identify user session boundaries in parallel workload logs. Identifying and characterizing the sessions is important in the context of workload modeling, especially if a user-based workload model is considered. Traditionally, sessions have been delimited by long think times, that is, by intervals of more than, say, 20 minutes from the termination of one job to the submittal of the next job. We show that such a definition is problematic in this context, because jobs may be extremely long. As a result of including each jobs execution in the session, we get unrealistically long sessions, and indeed, users most probably do not always stay connected and wait for the termination of long jobs. This result is presented in Chapter 2, Figures 9,10.

In this work, we compare multiple options to identify the sessions. For example, we tried to solve the problem of long sessions in the common definition that uses the jobs termination time to define the session, by limiting the inter-arrival time between jobs in the same session. However, the limit value appears to have a huge impact on the sessions, as demonstrated in Chapter 2, Figures 11-12.

Our research concludes that the correct way to identify session is based on proven user activity, namely the submittal of new jobs, regardless of how long they run. The common approaches in this area lead to unrealistic long sessions and the attempts to solve this problem lead to a strong relation between the sessions and the threshold value. Therefore, by elimination, we left with the arrival-time approach, which is common in other domains, such as concluding user's sessions in HTTP logs. In contrast to the rest of the approaches, the arrival-time approach leads to more realistic sessions length and distributions, which are presented in Figure 8 and Figure 11.

This work changes the way sessions are usually identified. Therefore, we provide more details regarding the sessions properties [Table 1, Figures 13-15]. The rest of our work in simulations (Chapters 3-5) uses this approach to identify sessions and to deduce users behavior from a trace.

The concept of resampling, presented in Chapter 3, suggests a new, novel way that combines the realism of real traces with the flexibility of models in order to create new workloads. This is done by dividing the trace into a sub-trace per user, and recombining them in various ways. Resampling has the following attributes:

- Retaining the complex internal structure of the original trace, including features that we do not know about.
- Allowing manipulations that affect specific properties that we know about and want to change as part of the evaluation.

Resampling creates multiple workloads based on a single recorded workload. We first show that resampling preserves the attributes of the original trace. In Chapter 3, Figures 4-5 show that the produced workloads have the same distributions as the original trace, such as daily and weekly cycles, users activity length, number of jobs and sessions per user, and many more. Figures 6-7 show that they also have the same self similarity values.

While resampling preserves the attributes of the original trace, it produces variations of the workload and allows manipulation of specific properties. Resampling provides the following applications:

- Verification of performance results. One of the problems with using a workload trace directly in simulations is that it provides a single data point. This has the obvious deficiency that it is impossible to calculate any kind of confidence intervals except perhaps by the method of batch means [49]. But with resampling we can create many resampled randomized versions of the workload, and evaluate the performance of the system with all of them, thus obtaining multiple data points that all adhere to the same underlying statistics. The distribution of these data points can then be used to compute confidence intervals for performance metrics. This is essentially an application of the well-known technique of bootstrapping used in statistical analysis [21]. In Chapter 6, Figures 8-9 show histograms of the performance metrics using resampling, which demonstrate this application.
- Extending a trace. While some of our workload traces are pretty long, with hundreds of thousands of jobs submitted over 2 years or more,

others are shorter. In addition, a significant part of the trace may be needed as a "warmup period" to ensure that the simulated system achieves its steady state [49]. Given only the raw traces, the length of the simulation may therefore be quite limited. But with resampling we can extend the simulation to arbitrary lengths. As an example, we extended a log to be five times longer, and showed in Chapter 6, Figure 10 that except the length, the rest still had similar properties as the original workload.

- Changing the load. An important aspect of systems performance evaluation is often to check the system's performance under different load conditions, and in particular, how performance degrades with increased load. Given a single trace, crude manipulations are typically used in order to change the load. For example, multiplying all arrival times or all required numbers of processors by a constant. However, these approaches have well known deficiencies, that lead to unrepresentative behavior. With resampling we can easily change the load, by increasing or decreasing the sampling-rate. We used resampling to multiply by 6 the load of a cluster with very low utilization, while we preserved the rest of the properties except the load (Chapter 6, Figure 11). We also used resampling to analyze the impact of the load on the system's performance (Chapter 5, Figure 12).
- Over-sampling rare behaviors. Workload logs sometimes contain unique users that behave anomalously in a specific period compared to the rest of the users. For example, a user may submit an inordinate number of jobs during a single week thus creating a flash crowd. With resampling we can check its influence, by over-sampling these behaviors. For example, we over-sampled flash crowds in order to check their impact on the system's performance (Chapter 6, Figures 12-16).

Pay attention that in the results of these applications we show two important things. The first is that the applications work as expected and achieve their goal (e.g. extending the load actually creates a new workload with higher load). The second is that the new workloads preserve the properties of the original trace and they have similar distributions. This means that resampling achieves its goals of retaining the complex internal structure of the original trace while allowing workload manipulations.

Resampling creates multiple instances of essentially the same workload. The next question that we dealt with, in Chapter 4, is the following: given a workload that represents a work on a certain system, how can it be used properly to evaluate the performance of a new system design? This workload may be recorded from a real system or produced by resampling.

The conventional approach is to use the workload directly on the simulated system. In Chapter 4, we showed that this in 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.

Instead we propose a novel approach — A simulation that deduces the user's logical structure of dependencies and relations between the jobs, and preserves them during the simulation. The simulation also simulates the users responses to the system's decisions, by sending jobs later or earlier. For example, if the system is overloaded, the users will send their next jobs later. Another example, if the system prioritizes a user, he will receive faster responses, and therefore will send his next jobs earlier. This creates for each system a unique workload that represents how the users that were taken from the original workload would have worked on the simulated system. Note that the simulation only changes the arrival time of the jobs, while it preserves all the rest of the properties (such as number of jobs per user, and each job's length and processors number). Chapter 4, Figure 3, demonstrates the differences between the conventional simulation and our feedback based simulation. Here we describe the three main components in the simulation.

1. Pre-simulation step: deducing batches and constraints. Given a workload, we first split it to a subtrace per user. Then we split each subtrace into batches — sets of overlapping jobs. The jobs in a batch don't depend on each other. Chapter 4, Figure 1, provides an example of batches. Finally, we deduce all the constraints of sending a batch to the system. For example, if a batch depends on the results of a previous batch, we create a constraint for the later batch. We practically build a DAG per user, where the nodes are the batches, and edges represent constraints. This is demonstrated in Chapter 4, Figures 2,4, that present our constraints.

2. Tracking batches constraints. In the conventional simulations, all the batches are sent at predefined (original) timestamps. In our simulation however, only the batches without constraints are sent to the sys-

tem with their original timestamps. During the simulation, we track batches' arrivals and terminations, which may lead to constraints removal, and we update the constraints accordingly. If a batch becomes without any constraints, it is released to the system. For example, if a batch B has a constraint on a batch A because B depends on A's results, when A is terminated during the simulation, we remove this constraint. If B had only one constraint (on A), then now B is released to the system.

3. Choosing job's arrival time. Given a batch that we want to release to the system, the question is what will be its arrival times. We examined multiple approaches. We show that the simple intuitive proposals, such as preserving the original inter-arrival times and think-times, destroy the daily and weekly cycles, and create unrealistic workloads (see Chapter 4, Figure 5,6). As as an alternative, we propose a new approach called Fluid. Fluid proposes to preserve the original sessions structure of the users, by sending batches only during the original users' sessions. Fluid creates much more realistic workloads, that have similar attributes as the original workload (see Chapter 4, Figure 8,9). For example, with Fluid, the simulated workload preserves the daily and weekly cycles.

Up until now we discussed the motivation for feedback, how it is done, as well as showing that the workload is representative and preserves the original workload's attributes. Our final goal is to show the importance of feedback and why it is needed. For this purpose, we evaluate the performance of two well known schedulers using our feedback-based simulation and the conventional simulation. The first scheduler is a basic First Come First Serve (FCFS) scheduler, and the second uses more sophisticated scheduler named EASY. The results are presented on Chapter 4, Figure 10. When we use the feedback-based simulation, EASY creates shorter queues (shorter wait times), and therefore it is better. However, both FCFS and EASY have reasonable queue lengths. On the other hand, when we evaluate the schedulers using a conventional simulation, we notice that the results of FCFS are unreasonably poor.dfgf There are hundreds of jobs in average in the waiting queue, which practically does not happen in reality. This demonstrates that without adapting the jobs submittal rate to the system state, the simulated workload might be not only unrepresentative, but highly unrealistic. This, of course, leads to unreliable evaluations in conventional simulations.

Until now we described two different works with goal of improving the workloads that are used in simulations.

- Resampling. This is a tool that creates multiple instances of the same workload and allows workload manipulations. Note that resampling proposes a new methodology for generating workloads and it is not a simulation.

- Feedback. This work proposes a new type of simulation, with a loop-back (feedback) from the system to the users. This affects the arrival times of the jobs that are sent to the system and creates a workload that is adapted to the simulated system.

The next natural step, described in Chapter 5, is to combine between the two works into a single complete simulation. We call this simulation the Trace Based User Oriented Simulation (TBOUS). Chapter 5, Figure 2 shows the flow of TBOUS, and describes how the combination between resampling and feedback is done. TBOUS preserves the properties of both resampling and feedback:

- It inherits from resampling the option to create multiple workloads based on a single recorded trace and allow workloads manipulations such as changing the load, while preserving the original workload attributes.

- It inherits from feedback the simulation of the users responses to the system decisions, preserves the logical structure of the users (such as dependencies and think-times), and adapts the transmission rate to the system's performance.

This combination between resampling and feedback does not only provide a combination between the properties of both approach, but has additional advantages. One interesting property is that feedback re-synchronizes between the users after resampling destroyed this natural synchronization. Resampling by its very nature mixes up the users from the original trace. Thus user activity that was originally performed when the system was highly loaded may be matched up with activity of another user that was originally performed when the system was lightly loaded [54]. This is an unnatural combination. TBUOS solves this problem by using feedback in addition to resampling. The feedback adjusts the job arrival rate to match the momentary conditions in the simulated system, and therefore this problem does not happen. Chapter 5, Figure 4 shows that when we use the resampled workloads in conventional simulations we receive high wait times in average. However, TBOUS solves this problem, and the wait times are much lower.

Another important property of TBOUS is that it is a semi open system model. It splits the provided workload into two classes: long term users that use the system as a closed-system model and temporary users that use the system as an open-system model (see illustration in Chapter 5, Figure 3). Note that TBOUS is a combination of an open-system simulation (feedback) with a tool for generating workloads (resampling), and nevertheless TBOUS is a semi-open system.

The fact that TBOUS is a semi-open system is probably its most interesting and important attribute. In conventional simulations, the arrival process is preordained, and is not affected by the state and performance of the simulated system. As a result the system throughput is dictated by the trace being used, so the simulation cannot be used to measure throughput or utilization. Therefore, performance in such simulations is measured by the average wait time and slowdown. In TBOUS, on the other hand, the throughput and utilization reflect the performance of the evaluated system. Therefore TBOUS produces different loads and throughputs for different scheduling algorithms or parametrizations. This allows evaluations of the users' productivity and the resources utilization in addition to the delays in the system.

Our next goal is to show that TBOUS indeed creates dynamic throughput in contrast to the fixed throughput in conventional simulations. For this purpose, we evaluate and compare the throughput of FCFS scheduler and EASY, using the conventional simulation and TBOUS. For each simulation type (resampling+conventional or TBUOS) and scheduler (FCFS or EASY) we run a hundred simulations and tabulate the throughputs achieved. The results are presented on Chapter 5, Fig 6. Using conventional simulation with resampling, the distributions of throughput under FCFS and EASY are similar to each other and to the original value. On the other hand, in TBUOS the EASY scheduler led to increased throughput relative to the FCFS scheduler for all the traces. The difference testifies to the more realistic simulation of the users in TBUOS, including a logical response to a poor system performance. This results in reduced throughput with the FCFS scheduler, sometimes less than in the original trace.

In this experiment, we used the ability of TBOUS to evaluate the throughput of a system in order to evaluate more reliably common, well-known scheduling policies. However, and even more importantly, TBOUS allows us to evaluate systems that could not be well analyzed with the current tools. User-aware schedulers try to anticipate user behavior. The scheduler's goal is then to improve productivity by facilitating more effective use of the system, allowing users to submit more jobs. This is done by trying to assess whether users are still active, and prioritizing those who have the highest probability of submitting more jobs. This may increase the average wait time, because jobs belonging to users who are thought to be inactive are delayed, but will hopefully also increase the system throughput and the productivity of active users. Evaluating such schedulers requires methodologies like TBUOS, because under conventional simulations the throughput is fixed in advance.

Our final goal is to demonstrate that TBOUS allows reliable evaluations of user aware schedulers in contrast to the conventional simulation that would suggest that they are useless. For this purpose, we introduce the User Priority Scheduler (UPS). This is a user-aware scheduler intended to motivate users to submit more jobs. Jobs submitted to UPS are kept in a priority queue. When a scheduling decision has to be made (e.g. when some job terminates and processors become available) the highest priority job is selected for execution. The priorities are based on a weighted sum of two terms, one of which reflects the probability that a user will submit more jobs, and the other reflecting the waiting time of the job. We analyzed two approaches for the first parameter. The first approach is load-based prioritization. It prioritizes users based on their current demands from the system. The concept is to serve first users with less demands (which can be done relatively quickly), so they will be able to submit the jobs that depend on them. The alternative approach is recency-based prioritization. It prioritizes users based on the arrival time of their last job. The concept is to serve the users that are still on the computer (before they will break their sessions).

Next, we compare between the throughput of UPS and EASY in simulations. Chapter 5, Figure 8 shows the histogram of the throughput when the conventional simulation is used. The throughput does not change noticeably between the different schedulers. This means that user awareness does not seem to provide any benefits. Chapter 5, Figure 9 shows the histogram of the throughput when we use TBOUS. It is easy to see that UPS facilitates higher throughputs in average than EASY, and the gap grows with higher emphasis on user prioritization. Moreover,load-based prioritization appears to be more effective than recency based prioritization.

Chapter 5, that presents TBOUS, actually combines all our previous work, and introduce them in a single complete simulation. Therefore, this actually describes our research's contribution to the area of simulations. The simulation inherits the properties of feedback and resampling, because it uses both of them. However, it also has new unique characteristics. TBOUS has the following properties:

- Receives a recorded workload and simulates the system with a new workload that is based on the provided workload and preserves its properties.
- Produces multiple data points based on a single recorded trace.
- Allows workload manipulations, such as changing the load, extending a trace, and over-sampling rare-events.
- Preserves the user behavior characteristics (such as dependencies and think-times) and adapts the jobs arrival rate to the system state.
- Re-synchronizes between users after the resampling mixed-up the users.
- Produces dynamic throughput and utilization that reflect the performance of the evaluated system.
- Supports realistic evaluations of new system designs that can not be evaluated reliably in conventional simulations, such as user-aware schedulers.

This properties of TBOUS demonstrates its contribution to the area of performance evaluations. Our methodology is a unique breakthrough. Until our work, people had to choose between using trace-based simulations, which provide relatively reliable evaluations, but only a single data-point per log, or statistical models, that provide flexibility but less-reliable evaluations. Until now, people developed and used trace-driven simulations for years without realizing that the workloads actually retain attributes, such as the arrival rate, that reflect the users' responses to the decisions made by the original system, and this should not be retained, but adapted to the proposed system design. Before our work, people could not evaluate the throughput with representative workloads, but only in a pure closed system. Until our work, there were no reliable ways to evaluate system designs that encourage people to submit more jobs, for example user aware schedulers such as UPS. TBOUS provides a simulation that solves all these problems. This demonstrates our contribution to the area of performance evaluation using simulations.

TBOUS opens the door for a less common system design: semi-open models. This raises questions regarding the relationship between the different performance metrics, and leads us to our final work — Evaluating the Throughput (Chapter 6). In this work, we research semi-open models in analytical modeling for computer systems evaluations. We propose two possible models that use semi-open systems: the **mixed model** and the **re-open model**. We illustrate the models' flow in Chapter 6. These models allow evaluations of the throughput, while, in many cases, they create much more representative workloads than the common way to evaluate the throughput of a system — a pure closed system. In this work we describe both models in details, including their concept and parameters. Then we analyze them, including the impact of the parameters on the performance metrics and the relationship between the different performance metrics with each other.

The first model is the **mixed model**. This model includes an open workload class and a closed workload class. It is an abstraction of the workload used in the TBUOS simulations.

1. Open users submit a single job to the system and leave. This is like in a conventional open system. Arrivals of these users are not affected by the system state, so the load they impose on the system is also not affected by the system state. In other words, they have a constant contribution to the system throughput.

2. Closed users who submit a job, wait for it to terminate, think, submit another job, and so on indefinitely. This is just like a conventional closed system, and arrivals are naturally affected by the system state.

We assume that the properties of the jobs, such as the distribution of runtimes, are the same for both classes. The interesting issue is the interaction between the two classes. As in TBOUS, the open class affects the performance of the closed class, and this effect can lead to changes in the system throughput.

This model represents a system that has both permanent users and temporary users. For example, a cloud service that has users that paid to host a persistent service and other users that try the system for a limited time only and then leave. This is highly relevant today due to the growing popularity of cloud systems.

We describe the common approach to analyze the performance of such system using the mean value analysis (MVA) algorithm. While MVA provides complete evaluation of the common performance metrics, it is a recursive technique that does not provide any intuition. For example, it is important to understand the impact of changing a certain parameter. To provide this, we did the following:

1. We execute many simulations of the MVA algorithm, and present the relationship between the response time and the throughput (see Chapter 6, Figure 3).

2. We analyze the relations between the different parameters and performance metrics mathematically by using operational analysis.

The second model is **re-open model**. This is an open model with repeated submittal of jobs: users arrive at a given arrival rate as in a conventional open model, but once they arrive they may submit several jobs one after the other with think times in between. If the total number of jobs (or the probability to submit additional jobs) is fixed, the system cannot affect the throughput. But if the probability to submit additional jobs depends on system performance (and specifically on the response time) then throughput is indeed affected.

This model may be suitable for a web server or similar systems. When users surf to a site they usually send some number of requests. However, if the performance is poor, a user may get discouraged and leave the site. On the other hand, if the server is highly responsive, the users may extend their activity and send more requests.

We first analyze the re-open model using a constant $p$ value. We calculate that the throughput is $\frac{\lambda}{1-p}$, which means that the performance does not affect the throughput. We next present various performance metrics in Chapter 6, Figures 4-6. Figure 4 shows that the Think Rate doesn't impact the performance, Figure 5 shows the impact of $p$ on all the rest of the metrics, and Figure 6 analyzes the relations between the throughput, throughput per user, and response time.

Then, we analyze the re-open model when $p$ is a decreasing function of the response time. In this case, the throughput is dynamic and depends on the performance of the system. Short response times lead to higher throughput. But higher throughput will lead to more contention and thus to higher response times, and hence to reduced throughput. Therefore, the throughput and the response times balance each other, and we have a stabilizing negative feedback effect.

We simulated the FCFS scheduler with two different functions that calculate $p$ based on the response time: The first function is an exponentially decreasing function. The second is based on Shmueli and Feitelson work [55, 56]. Our final goal is to evaluate both approaches. Figure 8 shows that in both models, long run-times lead to a bigger chance of leaving the system. Figure 7 shows the impact of the arrival rate on the performance with both approaches. We can observe that the throughput when using Shmueli's approach converges to 1 for lower $\lambda$ and the average wait time is longer. Moreover, in Shmueli's approach the throughput per user starts higher, which means that users take advantage of the empty system to send more jobs, but they keep sending a high number of jobs when the system is loaded, and this leads to delays and subsequently to low throughput per user when $\lambda$ is high.

This work introduces and analyzes models that evaluate the throughput of a proposed system design with reasonable workloads using analytical modeling. It shows that our work in the area of simulations (Chapters 2-5), with the final outcome of TBOUS, is relevant not only for performance evaluations of parallel workloads system, and even not only for simulations, but for varied aspects under the important domain of performance evaluations.

# Bibliography

[1] V. S. Adve and M. K. Vernon, "*Performance analysis of mesh interconnection networks with deterministic routing*". *IEEE Trans. Parallel & Distributed Syst.* **5(3)**, pp. 225–246, Mar 1994.

[2] J. Aikat, S. Hasan, K. Jeffay, and F. D. Smith, "*Towards traffic benchmarks for empirical networking research: The role of connection structure in traffic workload modeling*". In 20th *Modeling, Anal. & Simulation of Comput. & Telecomm. Syst.*, pp. 78–86, Aug 2012.

[3] M. A. Amer, A. Chervenak, and W. Chen, "*Improving scientific workflow performance using policy based data placement*". In 22nd *Policies for Distrib. Syst. & Netw.*, pp. 86–93, Jul 2012.

[4] R. Bagrodia, R. Meyer, and M. Takai, "*Parsec: a parallel simulation environment for complex systems*". *Computer* **31**, pp. 77–85, Oct 1998.

[5] N. Baldo, F. Maguolo, M. Miozzo, M. Miozzo, and M. Zorzi, "*ns2-miracle: a modular framework for multi-technology and cross-layer support in network simulator 2*". In *International Conference on Performance Evaluation Methodologies and Tools*, Feb 2016.

[6] P. Barford and M. Crovella, "*Generating representative web workloads for network and server performance evaluation*". In *SIGMETRICS Conf. Measurement & Modeling of Comput. Syst.*, pp. 151–160, Jun 1998.

[7] I. Baumgart, B. Heep, and S. Krause, "*Oversim: A scalable and flexible overlay framework for simulation and real network applications*". In *Peer-to-Peer Computing*, Sep 2009.

[8] O. Boiman and M. Irani, "*Detecting irregularities in images and in video*". In 10th *IEEE Intl. Conf. Comput. Vision*, vol. 1, pp. 462–469, Oct 2005.

[9] R. Buyya and M. Murshed, "*Gridsim: a toolkit for the modeling and simulation of distributed resource management and scheduling for grid computing*". *Concurrency and Computation — Practice and Experience* Jan 2003.

[10] R. N. Calheiros, R. Ranjan, A. Beloglazov, C. A. F. D. Rose, and R. Buyya, "*Cloudsim: a toolkit for modeling and simulation of cloud computing environments and evaluation of resource provisioning algorithms*". *Software: Practice and Experience* **41**, pp. 23–50, Aug 2010.

[11] H. Casanova, F. Desprez, G. S. Markomanolis, and F. Suter, "*Simulation of MPI applications with time-independent traces*". *Concurrency & Computation — Pract. & Exp.* **27(5)**, pp. 1145–1168, Apr 2015.

[12] V. Casolaa, A. Cuomob, M. Rakc, and U. Villanob, "*The cloudgrid approach: Security analysis and performance evaluation*". *Future Generation Computer Systems* **29**, pp. 387–401, Jan 2013.

[13] W. Chen and E. Deelman, "*WorkflowSim: A toolkit for simulating scientific workflows in distributed environments*". In 8th *IEEE Intl. Conf. E-Science*, pp. 1–8, Oct 2012.

[14] Y. Chen, S. Alspaugh, and R. Katz, "*Interactive analytical processing in big data systems: A cross-industry study of MapReduce workloads*". *Proc. VLDB Endowment* **5(12)**, pp. 1802–1813, Aug 2012.

[15] Y. Chen, A. Ganapathi, R. Griffith, and R. Katz, "*The case for evaluating MapReduce performance using workload suites*". In 19th *Modeling, Anal. & Simulation of Comput. & Telecomm. Syst.*, pp. 390–399, Jul 2011.

[16] 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, 2002. Lect. Notes Comput. Sci. vol. 2459.

[17] 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.

[18] S. Dubnov, Z. Bar-Joseph, R. El-Yaniv, D. Lischinski, and M. Werman, "*Synthesizing sound textures through wavelet tree learning*". *IEEE Comput. Graphics & Applications* **22(4)**, pp. 38–48, Jul 2002.

[19] M. R. Ebling and M. Satyanarayanan, "*SynRGen: An extensible file reference generator*". In *SIGMETRICS Conf. Measurement & Modeling of Comput. Syst.*, pp. 108–117, May 1994.

[20] B. Efron, "*Bootstrap methods: Another look at the jackknife*". *Ann. Statist.* **7(1)**, pp. 1–26, Jan 1979.

[21] B. Efron and G. Gong, "*A leisurely look at the bootstrap, the jackknife, and cross-validation*". *The American Statistician* **37(1)**, pp. 36–48, Feb 1983.

[22] C. Ernemann, B. Song, and R. Yahyapour, "*Scaling of workload traces*". In *Job Scheduling Strategies for Parallel Processing*, D. G. Feitelson, L. Rudolph, and U. Schwiegelshohn (eds.), pp. 166–182, Springer-Verlag, 2003. Lect. Notes Comput. Sci. vol. 2862.

[23] M. Etinski, J. Corbalan, J. Labarta, and M. Valero, "*Parallel job scheduling for power constrained HPC systems*". *Parallel Comput.* **38(12)**, pp. 615–630, Dec 2012.

[24] D. G. Feitelson and E. Shmueli, "*A case for conservative workload modeling: Parallel job scheduling with daily cycles of activity*". In 17th *Modeling, Anal. & Simulation of Comput. & Telecomm. Syst.*, Sep 2009.

[25] D. G. Feitelson and D. Tsafrir, "*Workload sanitation for performance evaluation*". In *IEEE Intl. Symp. Performance Analysis Syst. & Software*, pp. 221–230, Mar 2006.

[26] D. G. Feitelson, D. Tsafrir, and D. Krakov, "*Experience with using the Parallel Workloads Archive*". *J. Parallel & Distributed Comput.* **74(10)**, pp. 2967–2982, Oct 2014.

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

[28] 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.

[29] P. Garca, C. Pairot, R. Mondjar, J. Pujol, H. Tejedor, and R. Rallo, "*Planetsim: A new overlay network simulation framework*". In *International Workshop on Software Engineering and Middleware*, vol. 3437, pp. 123–136, Sep 2004.

[30] S. K. Garg and R. Buyya, “*Networkcloudsim: Modelling parallel applications in cloud simulations*”. In *Utility and Cloud Computing*, pp. 105–113, Dec 2011.

[31] E. Gul, B. Atakan, and O. B. Akan, “*Nanons: A nanoscale network simulator framework for molecular communications*”. *Nano Communication Networks* **1**, pp. 138–156, June 2010.

[32] F. Hernández-Campos, K. Jeffay, and F. D. Smith, “*Modeling and generating TCP application workloads*”. In 4th *Broadband Comm., Netw. & Syst.*, pp. 280–289, Sep 2007.

[33] 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.

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

[35] P. Kamath, K.-c. Lan, J. Heidemann, J. Bannister, and J. Touch, “*Generation of high bandwidth network traffic traces*”. In 10th *Modeling, Anal. & Simulation of Comput. & Telecomm. Syst.*, pp. 401–412, Oct 2002.

[36] D. Klusek, L. Matyska, and H. Rudov, “*Alea – grid scheduling simulation environment*”. In *Parallel Processing and Applied Mathematics*, vol. 4967, Sep 2007.

[37] L. Kovar, M. Gleicher, and F. Pighin, “*Motion graphs*”. *ACM Trans. Graph.* **21(3)**, pp. 473–482, Jul 2002.

[38] 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.

[39] V. Kwatra, A. Schödl, I. Essa, G. Turk, and A. Bobick, “*Graphcut textures: Image and video synthesis using graph cuts*”. *ACM Trans. Graph.* **22(3)**, pp. 277–286, Jul 2003.

[40] E. D. Lazowska, J. Zahorjan, G. S. Graham, and K. C. Sevcik, *Quantitative System Performance: Computer System Analysis Using Queueing Network Models*. Prentice-Hall, Inc., 1984.

[41] S.-H. Lim, B. Sharma, G. Nam, E. K. Kim, and C. R. Das, "*Mdcsim: A multi-tier data center simulation platform*". In *IEEE International Conference on Cluster Computing and Workshops*, Aug 2009.

[42] A. M. Lindsay, M. Galloway-Carson, C. R. Johnson, D. P. Bunde, and V. J. Leung, "*Backfilling with guarantees made as jobs arrive*". *Concurrency & Computation — Pract. & Exp.* 2012.

[43] 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.

[44] P. Magnusson, M. Christensson, and J. Eskilson, "*Simics: A full system simulation platform*". *Computer* **35**, pp. 50–58, Aug 2002.

[45] D. A. Menascé, V. A. F. Almeida, and L. W. Dowdy, *Performance by Design: Computer Capacity Planning by Example*. Prentice Hall, 2004.

[46] 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.

[47] S. Ostermann, K. Plankensteiner, R. Prodan, and T. Fahringer, "*Groudsim: An event-based simulation framework for computational grids and clouds*". In *European Conference on Parallel Processing*, vol. 6586, pp. 305–313, Aug 2010.

[48] S. Park, A. Savvides, and M. B. Srivastava, "*Sensorsim: a simulation framework for sensor networks*". In *ACM international workshop on Modeling, analysis and simulation of wireless and mobile systems*, pp. 104–111, Aug 2000.

[49] K. Pawlikowski, "*Steady-state simulation of queueing processes: A survey of problems and solutions*". *ACM Comput. Surv.* **22(2)**, pp. 123–170, Jun 1990.

[50] R. S. Prasad and C. Dovrolis, "*Measuring the congesion responsiveness of Internet traffic*". In 8th *Passive & Active Measurement Conf.*, pp. 176–185, Apr 2007.

[51] A. Rajbhandary, D. P. Bunde, and V. J. Leung, "*Variations of conservative backfilling to improve fairness*". In *Job Scheduling Strategies for Parallel Processing*, N. Desai and W. Cirne (eds.), pp. 177–191, Springer-Verlag, 2013. Lect. Notes Comput. Sci. vol. 8429.

[52] 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.

[53] 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)**, pp. 385–398, Oct 1990.

[54] 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.

[55] 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.

[56] 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.

[57] F. Singhoff, J. Legrand, L. Nana, and L. Marc, "*Cheddar: a flexible real time scheduling framework*". *SIGAda* **XXIV**, pp. 1–8, Dec 2004.

[58] J. Sommers and P. Barford, "*Self-configuring network traffic generation*". In 4th *Internet Measurement Conf.*, pp. 68–81, Oct 2004.

[59] 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.

[60] 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.

[61] R. Ubal, B. Jang, and P. Mistry, "*Multi2sim: A simulation framework for cpu-gpu computing*". In *International Conference on Parallel Architectures and Compilation Techniques*, vol. 19–23, Sep 2012.

[62] K. V. Vishwanath and A. Vahdat, "*Swing: Realistic and responsive network traffic generation*". *IEEE/ACM Trans. Networking* **17(3)**, pp. 712–725, Jun 2009.

[63] M. C. Weigle, P. Adurthi, F. Hernández-Campos, K. Jeffay, and F. D. Smith, "*Tmix: A tool for generating realistic TCP application workloads in ns-2*". *Comput. Commun. Rev.* **36(3)**, pp. 67–76, Jul 2006.

[64] T. Wenisch, R. Wunderlich, and M. Ferdman, "*Simflex: Statistical sampling of computer system simulation*". *IEEE Micro* **26**, pp. 18–31, Aug 2006.

[65] Y. Wexler, E. Schechtman, and M. Irani, "*Space-time video completion*". In *Conf. Comput. Vision & Pattern Recog.*, vol. 1, pp. 120–127, Jun 2004.

[66] W. Willinger, V. Paxson, and M. S. Taqqu, "*Self-similarity and heavy tails: Structural modeling of network traffic*". In *A Practical Guide to Heavy Tails*, R. J. Adler, R. E. Feldman, and M. S. Taqqu (eds.), pp. 27–53, Birkhäuser, 1998.

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

[68] N. Zakay and D. G. Feitelson, "*Workload resampling for performance evaluation of parallel job schedulers*". *Concurrency & Computation — Pract. & Exp.* **26(12)**, pp. 2079–2105, Aug 2014.

# מכתב תרומה

דוקטורט זה, אשר מוגש כאסופת מאמרים, מתבסס על המאמרים הבאים:

1. N. Zakay and D. G. Feitelson, "*On identifying user session boundaries in parallel workload logs* ". In J*ob Scheduling Strategies for Parallel Processing*, W. Cirne et al. (eds.), pp. 216–234, Springer-Verlag, 2012. Lect. Notes Comput. Sci. vol. 7698.
2. N. Zakay and D. G. Feitelson, "*Workload resampling for performance evaluation of parallel job schedulers* ". In *4th Intl. Conf. Performance Engineering*, pp. 149–159, Apr 2013.
3. N. Zakay and D. G. Feitelson, "W*orkload resampling for performance evaluation of parallel job schedulers* ". *Concurrency & Computation — Pract. & Exp*. 26(12), pp. 2079–2105, Aug 2014. This is an extended journal version of the original paper.
4. N. Zakay and D. G. Feitelson, "*Preserving user behavior characteristics in trace-based simulation of parallel job scheduling* ". In 22nd *Modeling, Anal. & Simulation of Comput. & Telecomm. Syst.*, pp. 51–60, Sep 2014.
5. N. Zakay and D. G. Feitelson, "*Semi-open trace based simulation for reliable evaluation of job throughput and user productivity* ". In 7th *IEEE Intl. Conf. Cloud Comput. Tech. & Sci.*, pp. 413– 421, Nov 2015.
6. N. Zakay and D. G. Feitelson, "*Models for Evaluating Throughput*", unpublished.

ישנם כחמישה מאמרים מפורסמים ועבודה אחת שטרם פורסמה. בכל המאמרים המחברים הם נתנאל זכאי ופרופסור דרור פייטלסון בלבד. כלומר, אין שותפים נוספים לעבודה.

המשתמשים לסיום של עבודות קודמות ולביצועי המערכת. לכן זה יותר חשוב לשמר את התנהגות המשתמשים והתלויות בין העבודות השונות מאשר את זמני ההגעה עצמם.

על מנת לעשות זאת, בתחילה נדרשנו להבין כיצד משתמשים מתנהגים. תכונה חשובה של משתמשים במערכת היא הצורה שבה הם עובדים. המשתמשים עובדים במערכת לזמן מה ואז הם עוצרים ועושים משהו אחר. זמן העבודה הרציף של משתמשים במערכת נקרא סשן (session). עומסי העבודה שלנו מכילים נתונים לגבי עבודות בודדות (כמו זהות המשתמש ששלח את העבודה), אך הם אינם כוללים פרטים על מקבצי עבודות כמו מבני הסשנס של המשתמשים. נתחנו שיטות שונות על מנת לזהות ולאפיין את הסשנס של משתמשים על פי המידע בעומס עבודה מוקלט [1]. השתמשנו בהבנות אלו כאשר נדרשנו להסיק את התנהגות המשתמשים מעומס עבודה מוקלט.

תכונה נוספת חשובה בהתנהגות המשתמשים היא שטף העבודה (workflow) שלהם. באמצעות הסקת תלויות מעומס עבודה מוקלט, אנחנו מראים כיצד הדמיה יכולה לשמר את התלויות בין העבודות. שימור התלויות נעשה באמצעות הוספת משוב מהמערכת אל המשתמשים. דבר זה מייצר מערכת חצי פתוחה (semi open system) אשר מכילה מידול של התנהגות המשתמשים. בהדמיות אלו, כמו במציאות, המשתמשים מתאימים את זמני ההגעה של העבודות שלהם לביצועי המערכת [4]. כתוצאה מכך, ההדמיה מייצרת עומסי עבודה שונים וכן תפוקה וניצולת שונים עבור אלגוריתמי תזמון שונים. בנוסף, הצענו מתזמן הנקרא User Priority Scheduler (UPS) אשר מספק שיפור ניכר בתפוקה של המערכת כאשר אומדים אותו באמצעות ההדמיה שלנו, אולם לא ניתן להבחין או לאמוד שיפורים אלו באמצעות ההדמיות שנפוצות כיום המתבססות על מערכת פתוחה [5].

מערכות פתוחות וסגורות נפוצות גם בתחום של ניתוח ביצועים אנליטי. אולם, מערכות חצי פתוחות כפי שאנחנו מציעים [5] הן פחות פופולריות, למרות שצורת עבודה זאת אופיינית במערכות רבות הקיימות במציאות. בנוסף, במערכות פתוחות התפוקה היא קלט של ההדמיה ולכן לא ניתן לאמוד את התפוקה של המערכת המוצעת. הפתרון הנפוץ כיום הוא להשתמש במערכת סגורה, אך היא לרוב לא ייצוגית. אנחנו מציעים אלטרנטיבה: שני מודלים של מערכות חצי פתוחות עם תפוקה דינמית. אנחנו מנתחים את הפרמטרים שלהם, התכונות שלהם, הדרך הנכונה לאמוד ביצועים באמצעותם, וכן אנחנו מציגים את האינטראקציה בין מדדי הביצועיים השונים (כמו זמן ההמתנה והתפוקה) עבור שני המודלים. תוצאות אלו משפרות את ההבנה של מערכות חצי פתוחות גם עבור ניתוח ביצועים אנליטי. עבודה זאת, בשמה המלא *מודלים לאומדן התפוקה* (*Models for evaluating throughput*) טרם פורסמה, אך מהווה חלק אינטגרלי מעבודת התזה שלנו.

## ביבליוגרפיה

[1] 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, 2012. Lect. Notes Comput. Sci. vol. 7698.

[2] N. Zakay and D. G. Feitelson, "*Workload resampling for performance evaluation of parallel job schedulers*". In *4th Intl. Conf. Performance Engineering*, pp. 149–159, Apr 2013.

[3] N. Zakay and D. G. Feitelson, "W*orkload resampling for performance evaluation of parallel job schedulers*". *Concurrency & Computation — Pract. & Exp.* 26(12), pp. 2079–2105, Aug 2014.

[4] N. Zakay and D. G. Feitelson, "*Preserving user behavior characteristics in trace-based simulation of parallel job scheduling*". In 22nd *Modeling, Anal. & Simulation of Comput. & Telecomm. Syst.*, pp. 51–60, Sep 2014.

[5] N. Zakay and D. G. Feitelson, "*Semi-open trace based simulation for reliable evaluation of job throughput and user productivity*". In 7th *IEEE Intl. Conf. Cloud Comput. Tech. & Sci.*, pp. 413– 421, Nov 2015.

# תקציר

## שיפור המציאותיות והייצוגיות של עומסי עבודה בכדי להשיג הערכות ביצועים אמינות יותר

## Improved Realism and Representativeness of Workloads to Improve Reliability of Performance Evaluations

סטודנט: נתנאל זכאי

מנחה: פרופסור דרור פייטלסון

כאשר מערכת חדשה מוצעת, זה לא פרקטי להטמיע אותה מיד במוצר בגלל התקורה הגבוהה. במקום זאת, בהתחלה מעריכים אותה בהדמיה (סימולציה), ורק אם היא משיגה שיפורים משמעותיים בביצועים היא הופכת למועמדת להטמעה. לכן, אמינות ההדמיות היא קריטית ובעלת השפעה מכרעת על המערכות שממומשות בסופו של דבר.

הביצועים של המערכת מושפעים מעומס העבודה (workload) שהמערכת מטפלת בו. לכן, הערכות ביצועים אמינות דורשות שעומס העבודה יהיה ייצוגי. כתוצאה מכך, בהדמיות מודרניות לעיתים תכופות משתמשים בלוגים - עומסי עבודה אשר הוקלטו במערכות אמיתיות על מנת לאמוד את הביצועים של מערכת מוצעת. המוטיבציה היא שלוגים של עומסי עבודה מוקלטים מכילים את כל מבנה העבודה הסבוך והמורכב של משתמשים אמיתיים.

כאשר משתמשים בלוג מוקלט, כל לוג מספק רצף אחד של עבודות, ולכן אומדן ביצועים אחד. אולם, בד"כ אנחנו רוצים לבדוק את הביצועים עבור מספר עומסי עבודה. למשל, על מנת להעריך את רווח הסמך של הביצועים יש צורך במספר רב של עומסי עבודה בעלי תכונות דומות. הדרך הנפוצה להשיג צורך זה היא להשתמש במודלים סטטיסטים (אשר מבוססים על מידע מעומסי עבודה מוקלטים). אולם, בעוד שמודלים אלו אכן מספקים מגוון וגמישות עבור הערכות ביצועים, הם לאו דווקא משמרים את התכונות החשובות של עומסי עובדה מוקלטים.

על מנת שעומסי העבודה יהיו יותר מייצגים, אנחנו מציעים *לשלב את המציאותיות של עומסי עבודה מוקלטים עם הדינמיות של מודלים.* זה נעשה על ידי מידול רק של החלקים בעומסי העבודה שרוצים לשנות, ודגימה-מחדש של שאר המידע מלוגים מוקלטים [2, 3].

באמצעות שימוש בדגימה מחדש על עומס עבודה מוקלט, אנחנו יכולים להשיג את הדברים הבאים: מספר עומסי עבודה דומים סטטיסטית (על מנת לחשב רווח סמך), שינוי העומס הממוצע בעומס העבודה (על מנת לבדוק איך העומס משפיע על הביצועים במערכת), הארכת משך עומס העבודה (על מנת להבטיח התכנסות של הערכות הביצועים), וכן עודף דגימה של מאורעות מסוימים (עוזר לחקור את ההשפעה של אירועים כאלו על הביצועים).

מה שייחודי בדגימה מחדש זה שהיא מאפשרת שינוי של תכונות מסוימות (כמו אורך, עומס, ועוד) אך היא משמרת את המרכיבים הבסיסים של המבנה של עומס העבודה, כמו תכונות הריצה של העבודות ומבנה הסשנס (sessions) ומחזור העבודה היומי והשבועי של משתמשי המערכת.

ההדמיות הנפוצות בשימוש מריצות את העבדות על המערכת בצורה פתוחה (open system model). המשמעות של זה היא שבקשות מגיעות רק לפי זמן ההגעה שלהן בעומס העבודה המוקלט, בלי תלות בלוגיקה של התנהגות המשתמשים במערכת שגרמה להן להגיע בעיתוי זה. בנוסף, ביצועי מערכת כזאת נמדדים על פי זמני המתנה ושהייה במערכת, תחת תנאי העומס והניצולת הקבועים שהיו במערכת המוקלטת. הדמיות אלו לא יכולות להעריך את השפעת המערכת המוצעת על התפוקה והניצולת.

גם דגימה מחדש לא משנה מצב זה. עומסי עבודה (מוקלטים או שנוצרו באמצעות דגימה מחדש) משמרים את זמני ההגעה של העבודות במערכת המוקלטת. אולם, בפועל, זמנים אלו לרוב נובעים מתגובות

עבודה זאת נעשתה בהדרכתו של פרופסור דרור פייטלסון.

# שיפור המציאותיות והייצוגיות של עומסי עבודה בכדי להשיג הערכות ביצועים אמינות יותר

חיבור לשם קבלת תואר דוקטור לפילוסופיה

מאת

**נתנאל זכאי**

הוגש לסנט האוניברסיטה בירושלים
ספטמבר 2017