Resampling with Feedback: A New Paradigm of Using Workload Data for Performance Evaluation

(Extended Version)

Dror G. Feitelson

School of Computer Science and Engineering
The Hebrew University of Jerusalem, 91904 Jerusalem, Israel
feit@cs.huji.ac.il ORCID: 0000-0002-2733-7709

Abstract. Reliable performance evaluations require representative workloads. This has led to the use of accounting logs from production systems as a source for workload data in simulations. But using such logs directly suffers from various deficiencies, such as providing data about only one specific situation, and lack of flexibility, namely the inability to adjust the workload as needed. Creating workload models solves some of these problems but creates others, most notably the danger of missing out on important details that were not recognized in advance, and therefore not included in the model. Resampling solves many of these deficiencies by combining the best of both worlds. It is based on partitioning real workloads into basic components (specifically the job streams contributed by different users), and then generating new workloads by sampling from this pool of basic components. The generated workloads are adjusted dynamically to the conditions of the simulated system using a feedback loop, which may change the throughput. Using this methodology analysts can create multiple varied (but related) workloads from the same original log, all the time retaining much of the structure that exists in the original workload. Resampling with feedback thus provides a new way to use workload logs which benefits from the realism of logs while eliminating many of their drawbacks. In addition, it enables evaluations of throughput effects that are impossible with static workloads.
This paper reflects a keynote address at JSSPP 2021, and provides more details than a previous version from a keynote at Euro-Par 2016 [18]. It summarizes my and my students' work and reflects a personal view. The goal is to show the big picture and the building and interplay of ideas, at the possible expense of not providing a full overview of and comparison with related work.

1 Introduction

Performance evaluation is a basic element of experimental computer science [3]. It is used to compare design alternatives when building new systems, to tune parameter values of existing systems, and to assess capacity requirements when

setting up systems for production use. Lack of adequate performance evaluations can lead to bad decisions, which imply either not being able to accomplish mission objectives or inefficient use of resources. A good evaluation study, on the other hand, can be instrumental in the design and realization of an efficient and useful system.

It is widely accepted that the performance of a computer system depends on its design and implementation. This is why performance evaluations can be used to judge designs and assess implementations. But performance also depends on the workload to which the system is subjected. Evaluating a system with the wrong workload will most probably lead to erroneous results, that cannot be relied upon [10, 17]. It is therefore imperative to use representative and reliable workloads to drive performance evaluation studies. However, workloads can have complicated structures and distributions, so workload characterization can be a hard task to perform [5].

The problem is exacerbated by the fact that workloads may interact with the system and even with the performance metrics in non-trivial ways [12, 13]. Thus it may not be enough to use a workload model that is generally correct, and it may be important to get minute details correct too. But it is not always clear in advance which details are the important ones. This suggests that the workload should be comprehensive and include *all* possible attributes [26].

In the field of parallel job scheduling, the workload is the sequence of jobs submitted to the system. Early research in this field, in the 1980s, lacked data on which to base workloads. Instead studies were based on what were thought to be reasonable assumptions, or compared possible distributions — for example, a uniform distribution of job sizes, a distribution over powers of 2, and a harmonic distribution [23, 24]. But it was not known which of these is the most realistic.

Since the mid 1990s workload logs became available (starting with [22]) and were collected in the Parallel Workloads Archive [34, 28]. This enabled the substitution of assumptions with hard data [16, 14]. In particular, using logged data to drive simulations became the norm in evaluations of parallel job schedulers. But experience with this methodology exposed problems, especially in the context of matching the workload to the simulated system. This is described below in Section 3.

The suggested solution to these problems is to use resampling with feedback, as described in Section 4. The idea is to partition the workload into basic components, specifically the individual job streams produced by different users, and sample from this pool of components to create multiple alternative workloads [49]. At the same time, feedback from the simulated system is used to pace the workload generation process as would occur in reality [19, 37, 39, 51]. In practical terms, this means that we no longer just simulate the behavior of the system under study — rather, we study the dynamics of how the system interacts with its environment, and in particular, with its users.

The fact that jobs are submitted by simulated users might seem innocuous at first, but in fact it has major implications. The feedback effect works in both directions. On the one hand, the volume of the workload can grow as far as

the system capacity allows. But on the other hand we also have a stabilizing *negative* feedback, where extra load causes the generation of additional load to be throttled [19]. This reduces the risk of system saturation.

The resulting methodology enables evaluations that are not possible when using logs as they were recorded — and in particular, evaluations of how system design may affect throughput and utilization. This is extremely important, as low utilization of large-scale systems translates to the loss of a sizable fraction of the investment in the system. Notably, this methodology applies to any system type, not only to the context of parallel job scheduling.

2 Background

Our discussion is couched in the domain of parallel job scheduling. Parallel jobs are composed of multiple interacting processes which run on distinct processors. They can therefore be modeled as rectangles in $processors \times time$ space, where the height of a rectangle represents the number of processors used, and its width represents the duration of use. This representation rests on the assumption that processors are allocated to jobs on an exclusive basis, and jobs are run to completion on dedicated processors. This enables the parallel processes which constitute the job to communicate with each other efficiently using the machine's high-speed network, and avoids memory contention problems.

Scheduling parallel jobs is the decision of when each job will run. This can be considered as packing the rectangles which represent the jobs. An important goal is to create a dense packing, and minimize the space between job rectangles, because such spaces represent resources that are not utilized. Note that scheduling is an on-line problem: at each instant we do not know of future job arrivals, and even once a job arrives, we do not know for sure how long it will run.

The simplest scheduling algorithm is First-Come-First-Serve (FCFS), which simply schedules the jobs in the order that they are submitted to the system (Fig. 1). An alternative is EASY, named after the Extensible Argonne Scheduling sYstem which introduced it [31, 33]. The idea here is to optimize the schedule by taking small jobs from the back of the queue, and using them to fill in holes that were left in the schedule, an operation known as *backfilling*. This reduces fragmentation and improves throughput.

But note that the utility of backfilling depends on the workload. For example, if all the jobs require more than half the processors, two jobs can never run at the same time, and backfilling cannot be used. Thus if EASY is evaluated with such a workload, the result would be that the backfilling optimization is useless. But if real workloads actually do include many small jobs, then this conclusion would be wrong. Therefore workloads used in evaluations must be representative of real workloads. In particular, we need to correctly specify the job arrival patterns, and the job resource demands (processors and runtime).

An important observation is that, in the context of job scheduling (as opposed to task scheduling), there is a human in the loop. Jobs are submitted by human users of the system, and they often wait for the results of one job

Fig. 1. *Illustration of a sequence of parallel jobs (the workload) and how it would be scheduled by FCFS and EASY up to time T. Hatched areas represents wasted resources.*

before submitting another job. This creates a feedback loop from the system performance to the job submittal process. Our work is about how to achieve representativeness, and at the same time, how to incorporate this feedback.

3 Using Workload Logs and Models to Drive Simulations

There are two common ways to use a logged workload to analyze or evaluate a system design: (1) use the logged data directly to drive a simulation, or (2) create a model from the log and use the model for either analysis or simulation. As we'll show, both have deficiencies that may lead to problems in evaluations. The idea of resampling can be thought of as combining the two in order to enjoy the best of both worlds.

3.1 Workload Modeling

The generation of a workload model proceeds as follows:

1. Identify the workload attributes that need to be modeled. These are the important attributes that are expected to influence the outcome, for example the job arrival times and resource requirements.
2. Collect data, namely workload logs from production systems.

3. Fit the data regarding the identified workload attributes to mathematical distributions.

The model is then the description of the workload using these distributions. It can be used for random variate generation as input to simulations, or else the mathematical distributions can be used directly in a mathematical analysis.

Workload models have a number of advantages over logs. Some of the most salient ones are [17, Sect. 1.3.2]:

- The modeler has full knowledge of workload characteristics. For example, it is easy to know which workload parameters are correlated with each other because this information is part of the model. Such knowledge increases our understanding, and can lead to new designs based on this understanding. Workload logs, on the other hand, may include unknown features that nevertheless have a significant influence on the results. These cannot be exploited and may lead to confusion.
- It is possible to change model parameters one at a time, in order to investigate the influence of each one, while keeping other parameters constant. This allows for direct measurement of system sensitivity to the different parameters. In particular, it is typically easy to check different load levels. It is also possible to select model parameters that are expected to match the specific workload at a given site.
- A model is not affected by policies and constraints that are particular to the site where a log was recorded. For example, if a site configures its job queues with a maximum allowed duration of 4 hours, it forces users to break long jobs into multiple short jobs. Thus, the observed distribution of durations in a log will be different from the "natural" distribution users would have generated under a different policy, and the log — despite being "real" — is actually unrepresentative. In a model we can recreate the postulated real distribution.
- Logs may be polluted by bogus data. For example, a log may include records of jobs that were killed because they exceeded their resource bounds. Such jobs impose a transient load on the system, and influence the arrival process. However, they may be replicated a number of times before completing successfully, and only the successful run represents "real" work. In a model, such jobs can be avoided (but they can also be modeled explicitly if so desired).
- Models have better statistical properties: they are usually stationary, so evaluation results converge faster [9], and they allow multiple statistically equivalent simulations to be run so as to support the calculation of confidence intervals. Logs, on the other hand, provide only a single data point, which may be based on an unknown mixture of conditions.

These advantages have led to the creation and use of several workload models (e.g. [30, 4, 2, 32]), and even a quest for a general, parameterized workload model that can serve as a canonical workload in all evaluations [25].

3.2 Problems with Models

By definition, models include only what you know about in advance, and decide to incorporate in the model. As the result they do not include workload features that you think are not important, or that you don't know about. But such workload features may actually be important, in the sense that they have an effect on scheduler performance.

Over the years several examples of important attributes that were not anticipated in advance have been discovered. Three interesting ones are the following.

User Runtime Estimates: Perhaps the most interesting feature of parallel job workloads — in terms of its unexpected importance — is user runtime estimates. Many schedulers (including EASY) require users to provide estimates of job runtime when submitting a job; these estimates are then used by the scheduler to plan ahead. Simulations often assumed that perfect estimates are available, based on the assumption that users are motivated to provide them: shorter estimates increase the chances to backfill, but too short estimates would cause the job to be killed when it overruns its allocated time. This turned out to be wrong on two counts: first, estimates are actually very inaccurate, and second, it actually matters.

Typical runtime estimate accuracy data is shown in Fig. 2 [33]. These are histograms of the fraction of the user runtime estimates that was actually used (that is, of the quotient of the true runtime divided by the user's estimate). Ideally we would like to see all jobs near 100%, implying near-perfect estimates. But the actual peak at 100% is due to jobs that were killed because they exceeded their estimate. For other jobs except the very shortest ones the histogram is flat, implying that the estimates provide little information about the actual runtimes.

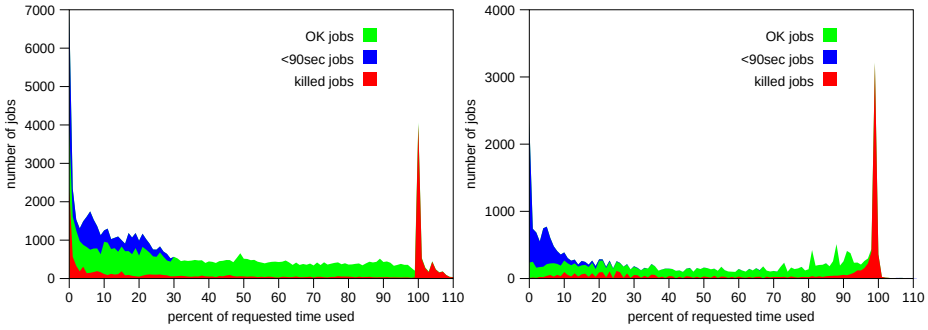

Fig. 2. *Histograms of user runtime estimate accuracy, from logs from the CTC and KTH SP2 machines [33].*

the effect of these inaccurate estimates is that unnecessary holes are left in the schedule. These holes allow for backfilling, and in particular, short holes

allow for the backfilling of short jobs [45]. This leads to a shortest-job-first like effect, which improves scheduling metrics like the average response time. As a result it appears that worse estimates lead to better performance [20, 52, 33].

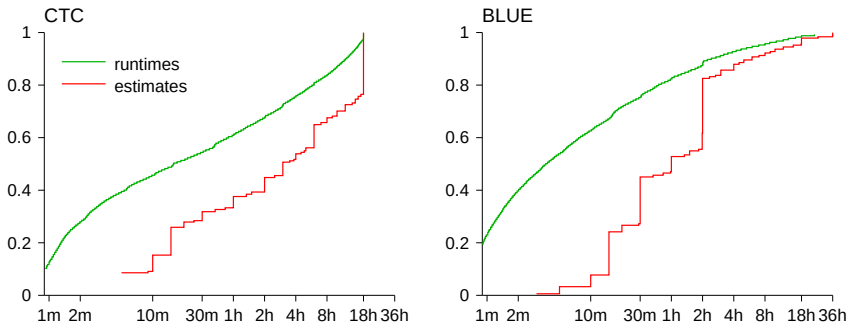

Fig. 3. *Comparison of the CDFs of the distributions of user estimates and actual runtimes, from logs from the CTC and BLUE machines [42].*

In order to achieve more reliable evaluations we need a better model of user runtime estimates and their (in)accuracy. This turned out to be highly non-trivial [42]. As shown in Fig. 3, users tend to use round figures such as 30 minutes or 2 hours as their estimates. And they tend to over-estimate by a large amount, often using the maximal allowed estimate, so as to avoid the risk of having their jobs killed. Together, this means that estimates provide much less information than they could.

In retrospect we now have a good model of user runtime estimates [42], and a good understanding of the interactions between estimates and other features of the workload, and the conditions under which one scheduler is better than another [45, 13]. But the more important result is the demonstration that performance evaluation results may be swayed by innocent-looking workload details, and that a very detailed analysis is required in order to uncover such situations.

The Daily Cycle of Activity: Another example is that real workloads are obviously non-stationary: they have daily, weekly, and even yearly cycles. In many cases this is ignored in performance evaluations, with the justification that only the high load at prime time is of interest. While this is reasonable in the context of network communication, where the individual workload items (packets) are very small, it is very dubious in the context of parallel jobs, that may run for many hours.

Consider the "user-aware" scheduler, which attempts to promote interactive work by prioritizing short jobs that have been submitted recently [39, 51]. This idea is based on the observation that short response times are correlated with short think times [38]: if a job's response time is short, chances are that the

user will submit another job soon after. But if the response time is long, the think time till the next job will also be long, implying that the user may leave the system and take a break. Consequently, by prioritizing recent short jobs, we better support users' workflows and facilitate the extension of their active sessions.

A comprehensive evaluation of this idea turned out to require two unusual elements. One was a user model which responds to feedback from the system performance, and adjusts the submittal of jobs based on their response time. This is discussed at length below in Sect. 4.3. The other was that the workload include a daily cycle of activity, as described at the end of Sect. 4.4.

Note that we did not explicitly design the user-aware scheduler to exploit the daily cycle of activity. But its performance turned out to depend on the existence of such a cycle [26]. The reason was that the prioritization of recent short jobs led to the starvation of long and non-recent jobs. If the workload exhibited a daily cycle, the non-critical jobs submitted during prime time were delayed and executed later during the night hours. As a result the system backlog was drained, and the full capacity was available for the interactive work of the next day. But if the workload did not have a daily cycle, there was no non-prime time, and thus no alternative time to execute the non-critical jobs. They therefore had to compete, and interfere with, the interactive workload — making it seem that the user-aware scheduler provides no benefits.

Workload Locality: Yet another effect that is prevalent in logs but usually absent from models is locality [15]. The locality properties of real workloads are especially important for the evaluation of adaptive and predictive systems (for example, it may be possible to predict job runtimes and compensate for inaccurate estimates [43]). Such features are becoming more commonplace with the advent of self-tuning and self-management. The idea is that the system should be able to react to changing conditions, without having to be reconfigured by a human operator [21]. But in order to study such systems, we need workloads with changing conditions as in real workload logs. A model based on random sampling from a distribution will not do, as it creates a stationary workload. This can be solved by employing "localized sampling" from the distribution [15], but a better solution is to use user-based modeling (or resampling), as described below.

3.3 Using Logs Directly

The perception that workload models may be over-simplified and unjustified has led many researchers to prefer real workload logs. Logs are used to generate the workload for simulations in a straightforward manner: jobs just arrive according to the timestamps in the log, and each job requires the number of processors and runtime as specified in the log.

The advantage of using a traced log directly as the input to a simulation is that it is the most "real" test of the simulated system: the workload reflects

a real workload precisely, with all its complexities, even if they are not known to the person performing the analysis [10, 17]. Consequently, this is the current best practice and is widely used.

The first such log to be made available came from the iPSC/860 hypercube machine installed at NASA Ames Research Center, and included all jobs executed on the system in the fourth quarter of 1993 [22]. This provided a wealth of hitherto unknown information regarding the distributions of job sizes and runtimes and patterns of user activity. Over the years many additional logs have been collected in the Parallel Workloads Archive [34, 28]. This resource is widely used as shown in Fig. 4.

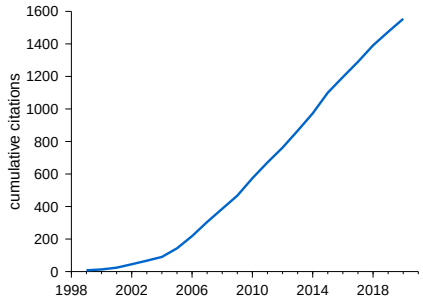

Fig. 4. *Number of citations to the Parallel Workloads Archive on Google Scholar.*

Contributing to the popularity of the Parallel Workloads Archive is the fact that each log is accompanied by copious metadata concerning the system and the logged data. In addition, all the logs are converted to a "standard workload format" [1]. Thus if a simulator can read this format, it can immediately run simulations using all the logs in the archive.

3.4 Drawbacks of Using Logs

While using logs "as is" avoids the problems associated with models, logs too have their drawbacks. Three major issues are the following.

Evaluation Flexibility: Each log reflects only one specific workload, and can only provide a single data point to the evaluation. But evaluations often require multiple simulations with related workloads. For example, the calculation of confidence intervals is best done by running multiple simulations with distinct but statistically identical workloads. This is easy with a workload model but impossible with a log.

More specifically, it is not possible to manipulate logs to adjust the workload to the simulated system and conditions, and even when it is possible, it can be problematic. In particular, it is often desirable to evaluate the performance of a

system under different load conditions, e.g. to check its stability or the maximal load it can handle before saturating. Thus a single load condition (as provided by a log) is not enough, and we need a tunable parameter that allows for the generation of different load conditions.

In log-based simulations it is common practice to increase the load on the system by reducing the average interarrival time. For example, if a log represents a load of 70% of system capacity, multiplying all interarrival times by a factor of $7/8 = 0.875$ will increase the load to 80%. But this practice has the undesirable consequence of shrinking the daily load cycle as well. The alternative of increasing the runtime to increase the load is not much better: jobs that originally came one after the other, and maybe even depended on each other, may now overlap. And increasing the number of processors to increase load is even worse. For example, if job sizes tend to be powers of 2 (which they are) then they pack well together. Increasing them by say 10% is not always possible (a 4-processor job can only be increased in increments of 25%), and when possible it has detrimental effects on the packing of the jobs onto processors.

Dirty Data: While logged data represents real activity, it is dubious whether *all* real data is worth using. Potential problems include errors in data recording, workload evolution and non-stationarity, multi-class workload mixtures, and abnormal activity.

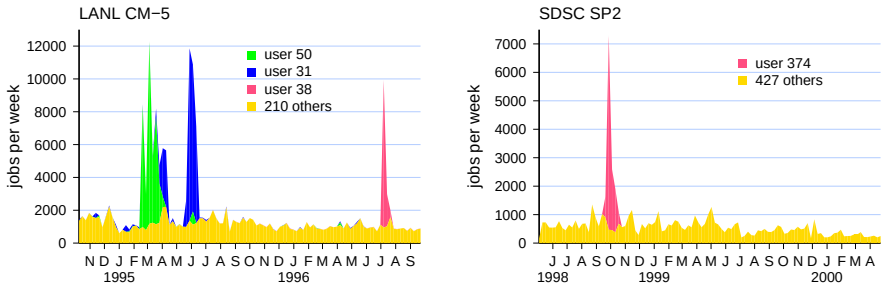

Fig. 5. *Arrivals per week on two parallel supercomputers, showing flurries of activity due to single users [27, 44].*

One type of unrepresentative activity is huge short-lived surges of activity by individual users, called flurries (Fig. 5). Such flurries may single-handedly affect workload statistics, especially the statistics of job arrivals, and perhaps also job attributes (a flurry with numerous similar jobs will create a mode in the distribution). While the existence of flurries is not uncommon (many logs exhibit them up to a few times a year), they are very different from the normal workload between them, and also different from each other. They should therefore be removed from the workload logs before they are analyzed and used in simulations [27, 44].

An example of the effect of a workload flurry is shown in Fig. 6. This is a rather small flurry that occurred in the CTC SP2 log. The log records data about 79,302 jobs submitted by 678 users over the course of 11 months. One of these users, user no. 135, had a flurry of 2080 jobs, nearly all of them during one day. The log was used in simulations of the EASY scheduler for different load conditions, where the load was changed in steps of 1% by modifying the job interarrival times as described above. Surprisingly, the results where erratic: the average bounded slowdown showed considerable fluctuations between neighboring load conditions. Even more surprisingly, these fluctuations disappeared when user 135's flurry was removed from the log, leaving a curve similar to what is expected in queueing analysis results [44]. In other words, a flurry that contained just 2.6% of the jobs and existed largely during a single day out of 11 months led to unreliable results.

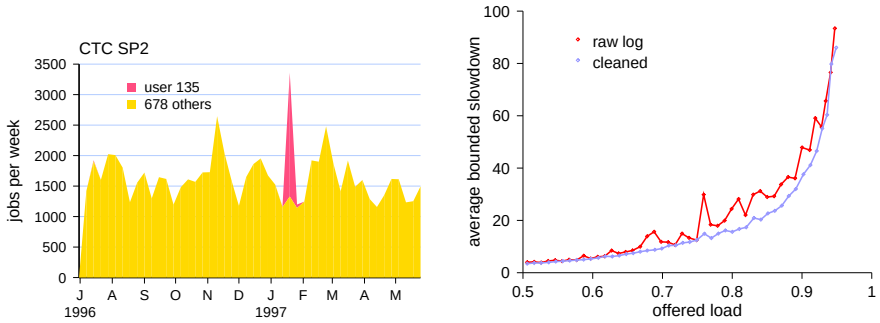

Fig. 6. *The effect of the CTC flurry on slowdown results.*

A different type of example is shown in Fig. 7. This is not a short-lived flurry, but rather the voluminous activity of a single user throughout the recorded log. In fact, user no. 2 in the HPC2N log accounted for no less than 57% of the jobs that appear in the log. Thus, if this log is used to analyze schedulers, we risk optimizing the scheduler to the behavior of a single user on a certain cluster in Sweden. An alternative interpretation is that this is not a real user, but rather represents a funnel for jobs from some external source, e.g. a multi-national grid system. This is just as bad, as it means that all data about individual users is lost.

The above examples suggest that workload data should be chosen carefully, and unrepresentative data should be removed. But this is a controversial suggestion: after all, if we start manipulating the data, aren't we harming the scientific validity of the results? In my opinion extolling the use of data as is reflects ignorance. Workload data that does not reflect typical system usage should be identified and removed [27]. However, we need to be transparent about the considerations for such cleaning, and what exactly was done to clean the data. In the parallel workloads archive, many logs have a cleaned version where flurries

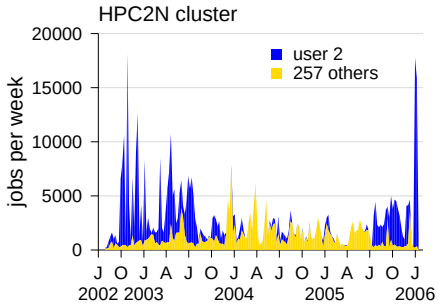

Fig. 7. *Arrivals per week on the HPC2n cluster, showing voluminous activity by user no. 2.*

have been removed, and the documentation contains the details of the filters used [28].

Signature of the Logged System: At a deeper level, we find that logged workloads actually contain a "signature" of the logged system. In other words, there is no such thing as a "real" workload which is the right one for general use: *every workload observed on a real system is the result of the interaction between that particular system and its users*. If the system behaves differently, for example if we decide to use a different scheduler, the users change their behavior as well.

This has major implications. It means that using a workload from one system to evaluate a different system is wrong, because the workload will not fit the simulation conditions [37, 38, 39]. We demonstrated this using a pair of cross-simulations of the two schedulers described above (Fig. 1) [37]. The first is the well known FCFS scheduler, which is inefficient and leads to wasted resources, because processors are left idle until the first queued job can run. The second is the optimizing EASY scheduler, which optimizes the schedule by taking small jobs from down the queue and backfilling them into holes left between earlier jobs. This allows EASY to sustain a heavier load. Using these schedulers, we conducted a site-level simulation of each one, namely a full simulation of the interaction between the scheduler and its users (this is explained below in Sect. 4.3). Importantly, in these simulations the workload is generated on-the-fly. These generated workloads were recorded, creating logs like those that are obtained from real supercomputer sites.

We then used each log as the input to a conventional simulation of the other scheduler (Fig. 8). As expected, the simulation of FCFS using a workload generated by an EASY simulation led to system saturation and overloading: FCFS could not handle the load that was generated when users interacted with EASY. Conversely, simulation of EASY using a workload generated by an FCFS simulation failed to show that EASY had a significant advantage, because the workload was not challenging enough.

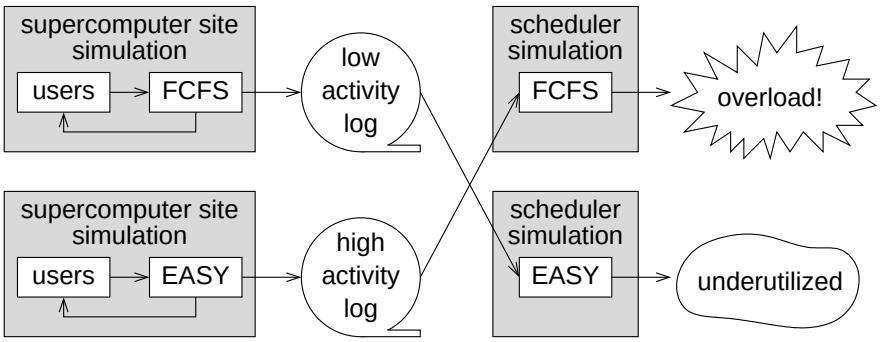

Fig. 8. *Experimental setup to demonstrate the effect of a system's signature on the workload.*

Taken together, all these problems seem to imply that workload logs are actually not any better than workload models. Resampling and feedback are designed to solve these problems and facilitate reliable evaluations.

4 Resampling and Feedback

The root cause of many of the problems with using logs is that a log represents unique conditions that were in effect when it was recorded, and may not be suitable for the desired evaluation. At the same time logs contain significant structure that we want to retain. Resampling is a way to provide flexibility while preserving the structure. And feedback adds just the necessary level of adjustment to the conditions that hold during the evaluation.

4.1 Before Resampling: Input Shaking and User-Based Modeling

The idea of resampling grew out of the ideas of input shaking and user-based modeling.

Input shaking was also an innovative use of logs in simulations [46]. The idea was to "shake" the job stream, meaning that in general the workload remained the same as in the log, but some of the jobs were adjusted to a small degree. For example, their arrival time could be changed by a small random amount. This enabled many simulations with similar but not identical workloads, and facilitated the identification of situations where the original results were actually due to some artifact and therefore not representative.

User-based modeling is based on the observation that the workload on a multi-user system is composed of the workloads by multiple individual users [8, 5]. The idea is then to *turn this into a generative approach* to creating workloads: first create independent workloads representing the different users, and then combine them together.

This idea was first proposed as a mechanism to generate locality of sampling [11], based on the fact that different users submits jobs with different characteristics [8, 5]. Since the community of active users changes over time, the number of active users in a given week — and the number of different programs they run — will be relatively small. The short-term workload will therefore tend to include repetitions of similar jobs, and will consequently tend to have more locality and be more predictable. But over a longer period this will change, because the set of active users has changed.

The essence of user-based modeling is an attempt to capture this structure using a multi-level model of the user population and the behavior of individual users. The top level is a model of the user population, including the arrival of new users and the departure of previous users. The second level models the activity of individual users as a sequence of sessions synchronized with the time of day (again based on data extracted from logs [48]). The lowest level includes repetitions of jobs within each session.

Remarkably, user-based modeling makes significant progress towards solving the problems outlined above:

- The workload will naturally have locality provided that the job models of different users are different from each other. During the tenure of each set of users the job stream will reflect the behavior of those users.
- The load on the system can be modified by changing the number of active users, or in other words, by changing parameters of the user population model. More users would generate higher load, but do it "in the right way".
- The generated workload can include non-stationary elements such as a daily cycle, by virtue of the model of when users engage in sessions of activity [39].
- As a special case, unique events such as workload flurries can be included or excluded at will, by including or excluding users with such unique behaviors.
- By using heavy-tailed session durations (and inter-session breaks) one can generate self similarity [47], which has been found in many types of workloads including parallel jobs [41, 49].

The drawback of this approach is the need to collect data and construct all these models. But maybe all this modeling takes us too far from the original log data? Resampling was designed to retain the original data as much as possible, and modify only whatever is needed for a specific purpose.

4.2 Resampling from a Log

Resampling is a powerful technique for statistical reasoning in situations where not enough empirical data is available [6, 7]. The idea is to use the available data sample as an approximation of the underlying population, and resample from it. Applying this to workloads, we can create workloads using the following procedure [49]:

1. Partition a workload log into its basic components, taken to be the individual job streams submitted by different users.

2. Sample from this pool of users.
3. Combine the job streams of the selected users to create a new version of the workload.

When looking at individual user traces, we find that some of them are active throughout much of the log's duration, while others are active only during a relatively short interval (a few weeks or months). We therefore distinguish between long-term users and temporary users (Fig. 9), and use them differently in the resampling. Users whose entire activity is too close to either end of the log are excluded.

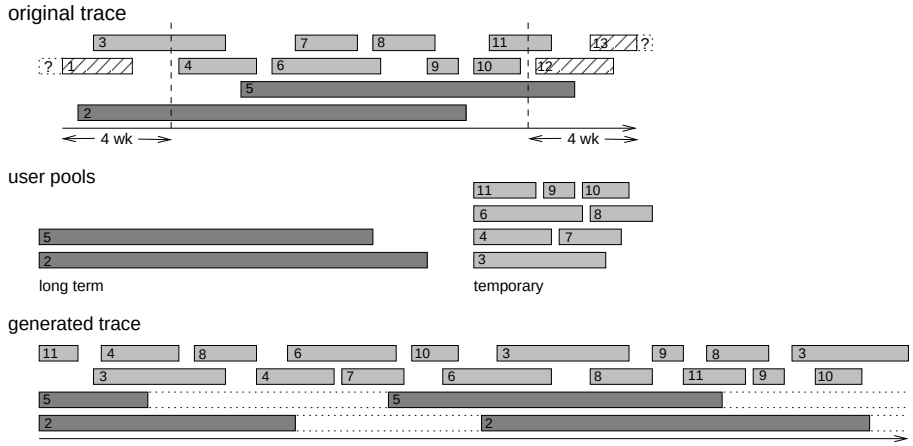

Fig. 9. *Conceptual framework of dividing users into long-term and temporary, and reusing them in a generated workload [49]. Each rectangle represents the full extent of activity by a certain user.*

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

- **Initialization:** We initialize the active users set with some temporary users and some long-term users. The defaults are the number of long-term users in the original log, and the average number of temporary users present in a single week of the original 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 log.
- **Temporary users:** In each new week of the simulation, a certain number of new temporary users are added (and a similar number are expected to leave,

on average). The exact number is randomized around the target number, which defaults to the average rate at which temporary users arrived in the original log. 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. Naturally the regenerations are also synchronized correctly with the time and day.

Each active user submits jobs to the system exactly as in the log (except that their timing may vary to reflect feedback as explained below). The flow of the simulation is shown in Fig. 10.

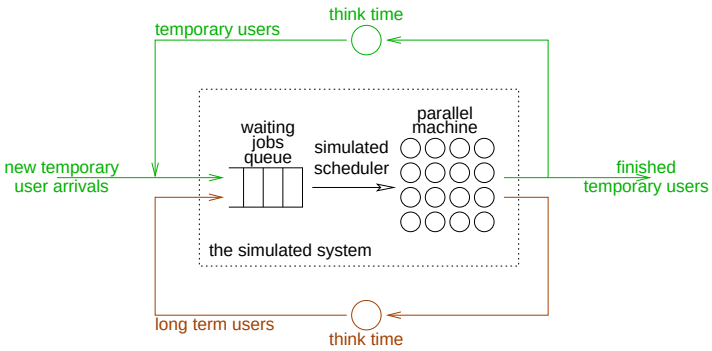

Fig. 10. *Queueing model of long term and temporary users in the simulation, leading to a semi-open system [51].*

Using resampled workloads enjoys all the benefits listed above for user-based workloads. In addition to that, it has the following advantages over using a log directly:

– We can create multiple different workloads with the same underlying structure and statistics, and
– We can extend the workload by continued resampling beyond the original length of the log.

However, recall that logs contain a signature of the interaction between the users and the scheduler (Sect. 3.4). In addition, each user's job stream contains a signature of interactions with the activity of other users: when one user loads the system, others tend to back off (Fig. 11). So if we just remix them randomly we might create unrealistic conditions. To solve these problems we need to add feedback.

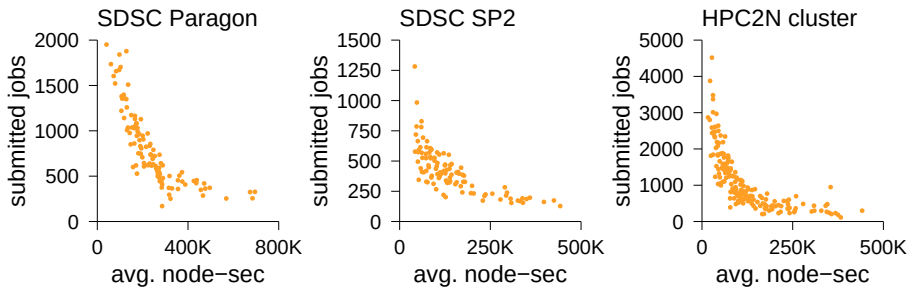

Fig. 11. *Evidence of interactions between users. In these scatter plots, each dot represents a week's activity. In weeks with jobs that require a lot of resources, there are relatively few jobs. When there are many jobs, all of them, by all users, are small.*

4.3 Adding Feedback

Conventional simulations that replay a log to recreate the workload essentially assume an open systems model: they assume the jobs arrive from an external, large user population, which is insensitive to the system's performance. As a result the jobs' arrivals are independent of each other. However, real computer systems are often more like closed systems [35, 36]: They have a limited user population, and new jobs are submitted only after previous ones terminate. This leads to a feedback from the system's performance to the workload generation.

We therefore suggest that it is not enough to simulate the computer system in isolation — we should also simulate the system's environment, namely the users who interact with the system, create its input, and wait for its response [37]. With resampling we introduce this explicitly by including a changing user community in the simulation. It is these (simulated) users who create the (simulated) jobs submitted to the (simulated) system. We need to add a feedback effect to these users' behavior.

Another way to look at the resulting system dynamics is as follows. Conventional performance evaluations, whether based on simulations or queueing analysis, attempt to assess the system's performance as a function of the offered load. But at the same time, the additional load generated by the users depends on the system's performance. By studying both effects together we can find the equilibrium point where workload generation and performance match each other (Fig. 12) [10]. This is the actual expected performance for this system and these users.

The fact that realistic workloads need to be paced according to system performance has also been noted in the context of computer networks [29]. In this case the feedback is caused by link congestion, which may not be readily visible in packet traces.

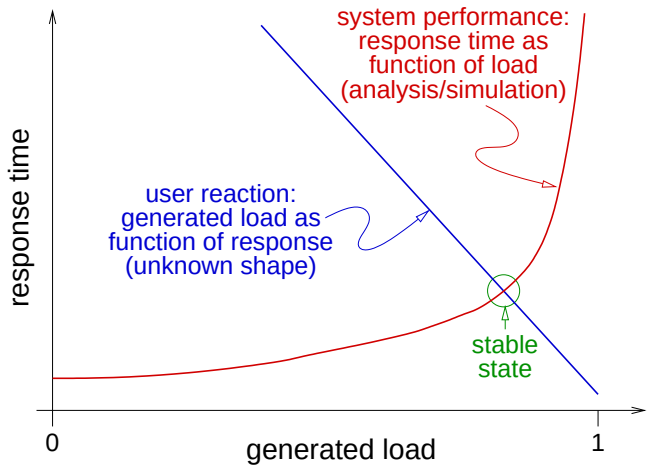

Fig. 12. *The observed workload on a system reflects the equilibrium between generating more jobs when performance is better and reduced performance when there are more jobs.*

User Models with Feedback: The way that feedback is incorporated into the simulations is via user models. As noted above there are three such models. The top level, the population model, is handled by the resampling as explained in sect. 4.2. The bottom level, that of jobs, is actually taken directly from the logs. The middle model, which concerns user activity patterns, is where the feedback comes in.

The way this works is shown in Fig. 13. In its simplest form, each user's activity is modeled by transitions between three states:

- Thinking and submitting a job,
- Waiting for a job to terminate, and
- Being away from the system.

When a job is submitted the user transitions to the waiting state. The feedback signal informs the user that the job has terminated, causing a transition back to the think and submit state. Alternatively, if the wait is too long, a transition to the "away" state may occur. Such a transition can also occur due to the passage of time. Returning to the system is also usually associated with the passage of time, e.g. simulating coming to work in the morning.

Recovering Dependencies from Logs: The idea of feedback is to pace the job submittal rate. Additional jobs will be submitted (by the simulated users) only after the jobs that they depend on have terminated (on the simulated system).

In other words, when we want to evaluate a new scheduling policy using a representative workload, the workload should reflect the user-level logic and not just parrot a previous workload. The logic is embodied in the dependencies

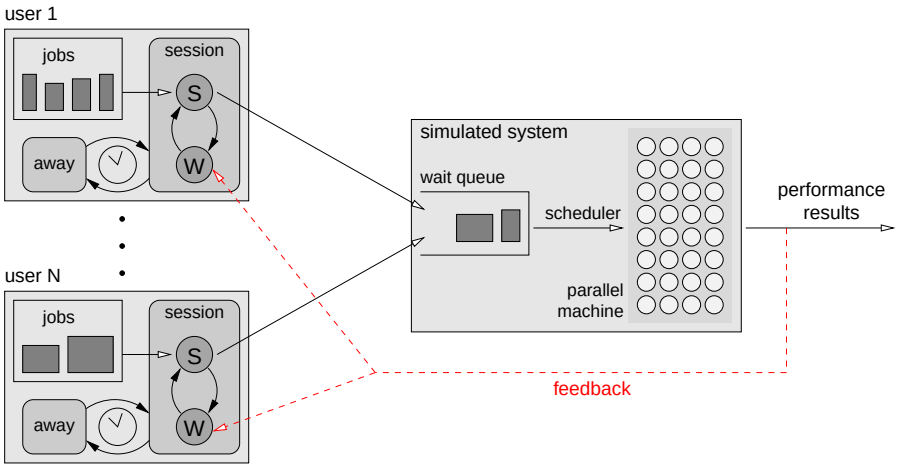

Fig. 13. *Illustration of a user-based simulation with feedback. When users are in session, they alternate between submitting jobs (S) and waiting for feedback regarding previous jobs (W). The sessions themselves often depend on (simulated) time of day.*

between jobs. Note that there is a tradeoff involved: the traditional approach is to replay the log using the exact timestamps of job arrivals as recorded in the log. This may violate the user logic if jobs are delayed in the simulated system. the alternative is to retain the logical dependencies, at the expense of losing the recorded timestamps. We argue that *it is more important to preserve the logic of the users' behavior than to repeat the exact timestamps that appear in the original log.*

The problem is that accounting logs used as data sources do not include explicit information regarding the dependencies between jobs. We therefore need to identify user sessions and extract dependencies between the jobs in each session [37, 50]. Two types of dependencies have been suggested [50]:

- Order constraint: jobs should start in the same order in the simulation as in the original log. In any log, the arrival times of all the jobs submitted by the same user form a sequence. Each job in the sequence should therefore depend on the previous job.
- Potential dependence constraint: every job potentially depends on all the jobs that have terminated before it started to run. As the jobs in each session form a sequence, it is enough to enforce dependencies on the last terminated job in each previous session.

These dependencies are illustrated in Fig. 14.

The way to integrate such considerations into log-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 [50]. Specifically, each job's submit time is adjusted to reflect feedback from the system performance to the user's behavior.

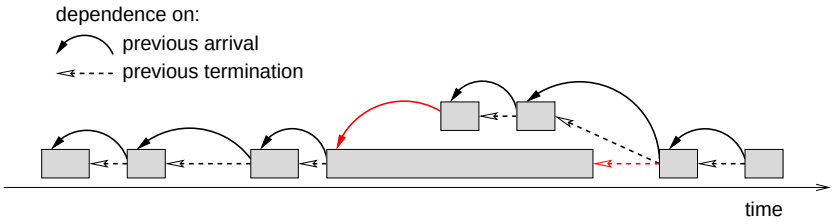

Fig. 14. *Dependencies on previous job arrivals and terminations. The ones shown in red show that the two types are not redundant.*

However, a job cannot arrive immediately when all its constraints are removed. Rather, its arrival should reflect reasonable user behavior (for example, users often go to sleep at night). 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 jobs to flow from one session to another according to the feedback. To do this, we keep each session's start and end timestamps from the original log. The think times between successive jobs are also retained from the original log. But if a job's execution is delayed in the simulation, leading to the next arrival falling beyond the end of the session, the next job will be delayed even more and arrive only at the beginning of the next session [50]. Contrariwise, if jobs terminate sooner in the simulation, jobs that were submitted originally in the next session may flow forward to occur in the current one.

4.4 Applications and Benefits

So what can we do with this new tool of workload resampling with feedback? Here are some results that would be hard or impossible to achieve with conventional simulations that just replay an existing log.

Validation: The first and foremost is to validate simulation results. Simulating with a given log provides a single data point. But with resampling we can get a distribution based on statistically similar workloads. In most cases the value which is obtained using a conventional simulation is within the body of this distribution, and the result is verified (Fig. 15). The width of the distribution can then be used to estimate the result's accuracy. But in some cases (e.g. the Blue log on the right) the distribution is shifted, indicating a mismatch between the behavior of the users in the original log and the expected behavior in the simulated system. This is a warning that a direct simulation with the original log may not be suitable for an evaluation of this scheduler.

Enhanced Simulations: Other benefits are more technical in nature. One of them is the ability to extend a log and create a longer workload, with more jobs in total, so as to facilitate better convergence of the results and reduce the

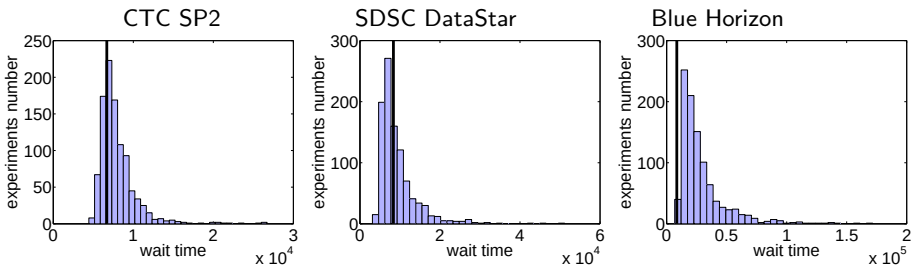

Fig. 15. *Histograms of the average waiting time in a thousand simulations of EASY on resampled workloads, compared to a simulation using the original logs (vertical line).*

detrimental effects of the initial "warmup" period. This is easy to achieve: we just continue resampling on and on for as long as we wish.

In principle it may seem that we can also change the load on the system by increasing or decreasing the number of active users. This is done by changing the number of long-term users in the initialization, and the number of new temporary users which arrive in each simulated week. Such a manipulation facilitates the study of how the system responds to load, and enables the generation of response curves similar to those obtained from queueing analyses. However, as noted above, increasing the load on the system is at odds with the throttling effect that comes with feedback. This is further discussed below.

Nevertheless, adding users to increase the load does work as long as the system does not saturate. This has the benefit of making low-load logs usable. Some logs were recorded on very low-load systems, with a utilization of only 25% of capacity or so. These workloads are not interesting as they do not tax the system to any appreciable degree. But by using resampling to increase their load they become interesting.

Finally, we note that once we partition the workload into individual users we can also look at different user classes in isolation. One example noted before is the workload flurries occasionally produced by some users. We can then evaluate the effect of such flurries by oversampling these users, and thus causing more and more flurries to occur (Fig. 16).

Throughput and Capacity Evaluation: Using results from resampling and feedback for verification hinges on the claim that such simulations are more valid to begin with. As noted above, using a log to drive a simulation suffers from the possible mismatch between the behavior of the users in the logged system and the behavior that would be observed for the simulated system. In particular, if the simulated system is more powerful, the users would be expected to submit more jobs, and vice versa. In simulations with feedback this indeed happens automatically, as demonstrated in Fig. 17. Note that this result is impossible to achieve using conventional log-based simulations. The ability of EASY to support a higher load is only observable when using feedback-based simulations.

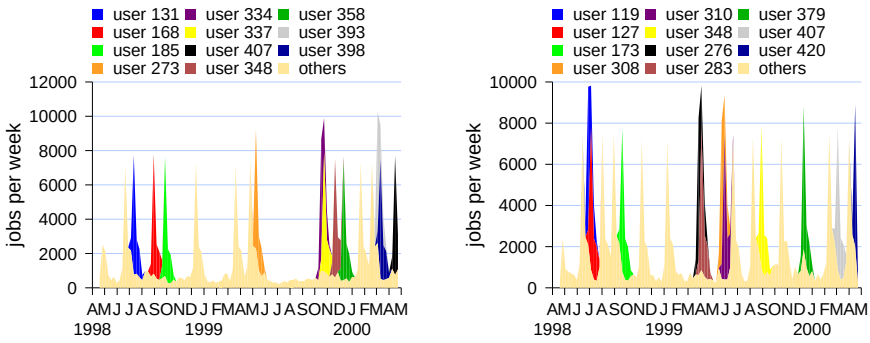

Fig. 16. *Examples of workloads with repeated flurries based on the SDSC SP2 workload from Fig. 5.*

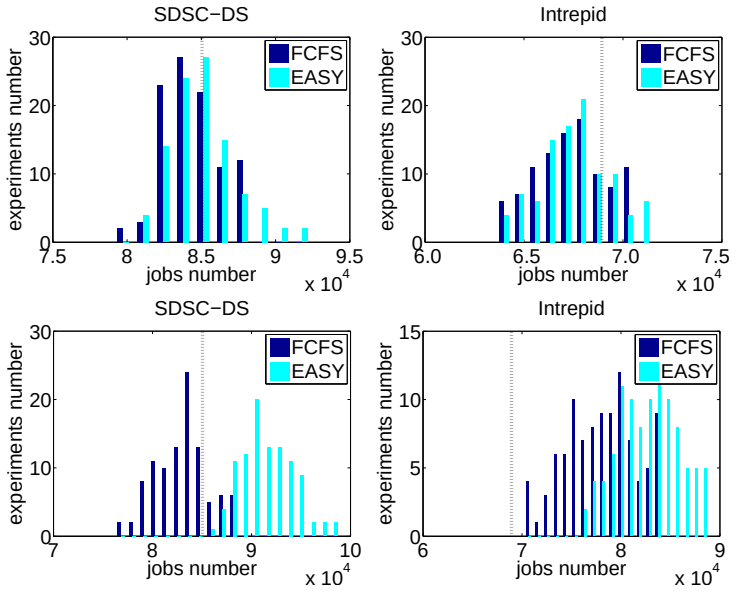

Fig. 17. *Histograms of the throughput achieved in one hundred simulations of the EASY and FCFS schedulers using the SDSC DataStar and Intrepid logs [51]. Top: in simulation without feedback the throughput is determined by the input, and a better scheduler has no effect. Bottom: in simulation with feedback EASY is seen to support a higher throughput than FCFS. The vertical line represents the throughput in the original log.*

However, note that increasing the load on the system is at odds with the throttling effect that comes with feedback. As the load increases, the system will naturally take longer to process each job. Simulated users who are waiting for a job to terminate before submitting their next job will therefore be delayed. So having more users will eventually cause each of these users to produce additional work at a slower rate, and the load will cease to increase! This is good because it is assumed that such an effect exists in real systems, where users abandon the system if it is too slow. But it frustrates our attempt to control the load and increase it at will.

Importantly, as a result of these dynamics we also have the additional benefit of being able to measure the maximal load that the system is expected to support. Resampling and user-based modeling facilitate the use of throughput as a performance metric, because the users become part of the simulation. But every system has a maximal capacity, and if the load exceeds this capacity the system saturates. In real systems this hardly happens, because of the feedback, and the system settles into a slightly lower load. Identifying this maximal expected load is an important part of a performance evaluation. Likewise, if we add system abandonment to the user behavior model, we can add the metric of the number of frustrated users.

Simulating User Aware Schedulers: Given that simulations based on resampling and feedback can exhibit changes in throughput, they can specifically be used to evaluate adaptive systems that are designed to enhance throughput (and thus productivity) rather than response time [39, 51].

For example, we can design a scheduler that prioritizes jobs based on their expected response time (mentioned above in Sect. 3.2). The idea is that if the response time is short, there is a good chance that the user will continue to submit more jobs. Note, however, that this depends on a model of what users care about. Regrettably, we typically do not have any explicit information about a user's motivation and considerations. But still some information can be gleaned by analyzing logs. For example, the question of what annoys users more and causes them to abort their interactive work has been investigated by tabulating the probability to submit another job as a function of the previous job's response time or its slowdown [38]. The result was that response time was the more meaningful metric (Fig. 18).

Based on this insight, we define think times of 20 minutes to be the boundary between continuously submitting jobs in the same session (short think times and rapid additional jobs) and session breaks (a long time before the next job, so the user probably took a break). With this definition, we find that the longer a job's response time, the lower the probability that the session will continue — namely, that the think time till the next job is submitted will be shorter than 20 minutes [38, 39].

With this background we can design schedulers that prioritize jobs that are expected to have short response times [39, 51]. To avoid starvation this is balanced with prioritizing based on waiting time. At one extreme all the weight is

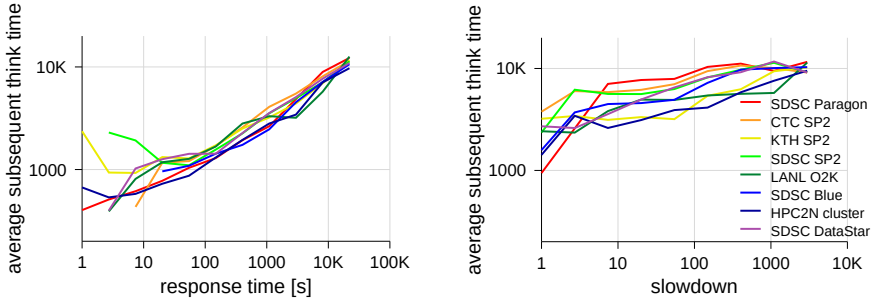

Fig. 18. *A job's performance as measured by the response time is a better predictor of subsequent user behavior (think time till the next job) than the job's slowdown.*

placed on responsiveness, accepting the risk of starvation. At the other extreme all the weight is placed on waiting time and none on responsiveness, which is equivalent to EASY. Different mixes lead to a spectrum of algorithms.

Trying to evaluate this idea with conventional workloads and simulations is useless — such simulations cannot evaluate productivity, and might even show that average response time is actually increased. But with a dynamic user-based simulation we can compare the resulting dynamics as the competition for resources intensifies. The results are shown in Fig. 19. When priority is given to the interactive jobs, sessions retain their length despite increasing load and the overall system throughput increases.

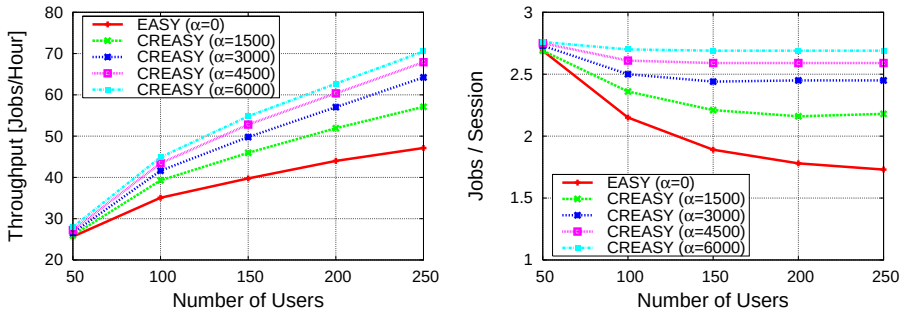

Fig. 19. *Average job throughput and session length are higher the more emphasis is placed by the scheduler on responsiveness (as reflected by the parameter α; CREASY is a version of EASY with priority to critical interactive jobs).*

It is also interesting to see how these results are achieved. Fig. 20 shows an analysis of the performance achieved by different job classes, where the classification is based on runtime: short is up to 1 minute, medium is 1 to 10 minutes, and long is above 10 minutes. As seen in the graph, the user-aware scheduler

achieves a significant reduction in response time for short and medium jobs, which are prioritized, while suffering a significant increase in response time for the long jobs, which were delayed.

Fig. 20. *The improved performance of short jobs comes at the expense of long jobs, which are delayed.*

Interestingly, this only worked when the simulation included a daily cycle of activity [26]. In retrospect, the reason is simple: the daily cycle allows the jobs that were delayed to run later, when the load on the system abates. A comparison of the (simulated) scheduler queue lengths is shown in Fig. 21. For both EASY and the user aware scheduler the queue builds up in the morning. But with EASY it quickly drains in the evening, leaving the system idle during the night. The user-aware scheduler, in contradistinction, overcommits during the day. As a result it has much more work left for the night, and utilizes the resources that EASY leaves idle. In simulations without daily cycles this effect cannot occur, and both schedulers display the same level of performance.

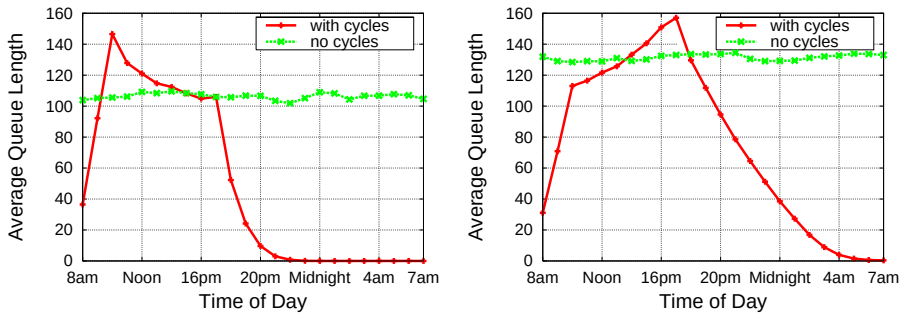

Fig. 21. *Queue lengths in simulations with and without daily cycles; EASY on the left and user-aware on the right. The simulation used a simple model where jobs arrive only between 8 AM and 6 PM.*

5 Conclusions

Resampling with feedback provides a new way to use workload logs in simulations, enabling the generation of varied and dynamically adjusted workloads that are specifically suited to evaluate the simulated system. This combines the realism of real log data with the flexibility of models. Basing the simulations as closely as possible on real logs reflects the importance of using hard data rather than assumptions. Adjusting the workload to the specific conditions during the simulation reflects the importance of the interaction between the users and the system. Without this, evaluation results are of unknown relevance, and might pertain to only irrelevant situations which do not occur in practice.

Particularly, by applying resampling and feedback to real workloads we achieve the following:

- Retain all important (including possibly unknown) details of the workload as they exist in logs recorded from production systems, with as little modifications as possible. At the same time, avoid the "signature" of the system on which the log was recorded.
- Enable evaluations of throughput and user satisfaction in addition to (or instead of) being limited to the response time and slowdown metrics. This also leads to natural support for assessing the saturation limit of the system.
- Provide a new interpretation of the goal of "comparing alternatives under equivalent conditions": this is not to process exactly the same job stream, but rather to face the same workload generation process (users). This acknowledges the realization that there is no such thing as a generally correct workload — rather, the workload depends on system.

Table 1. *Performance metrics under oblivious workloads and workloads with resampling and feedback.*

Metric	*Oblivious*	*Resampling+feedback*
load	compared with offered load to find the threshold beyond which the system saturates	measured to find the maximum utilization achievable — a performance metric
throughput	if not saturated, throughput is completely determined by the workload	achieved throughput is the main metric of system performance
response time	response time is the main metric of system performance	quoting response time is misleading without taking into account the throughput and user frustration
user satisfaction	assumed to be measured by the response time or slowdown	measured directly by the feedback model, e.g. in the form of frustrated user departures

The shift to using resampling with feedback has very significant implications. First, performance metrics change (Table 1). The new interpretation of good performance is that a better scheduler facilitates the execution of more jobs. Therefore the most important metric becomes the throughput. This can not be measured using conventional log-based simulations, where the workload is oblivious to the simulated system conditions, because in that context the throughput is dictated by the log and does not change in the simulation. Moreover, a consequence of achieving higher throughput may be that the average response time is also somewhat higher. In conventional simulations this would be taken as indicating that performance is worse, but now it is a side-effect of an improvement.

Related to this change in metrics is a change in simulation dynamics. In conventional oblivious simulations one must worry about system saturation. If the system is saturated, the simulation results are unreliable. A thorough methodology then has to check various load conditions to find the system capacity limit. But with resampling and feedback we do not need to do this any more. The feedback counteracts our efforts to increase the load by adding users and jobs, and leads to stability. The simulation then naturally settles to the throughput that reflects the correct balance between user behavior and system performance.

Workload manipulations such as those embodied in resampling with feedback are important tools in the performance analyst's toolbox, that have not received 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 logs, and enabling researchers to achieve better control over their properties, as needed for different evaluation scenarios.

Naturally, there are many opportunities for additional research regarding resampling and feedback. One element that is still largely missing is the user population model, and especially the issue of leaving the system when performance is inadequate. Another is the distribution of user types and behaviors. Resolving these issues requires not only deep analysis of workload logs, but also a collaboration with researchers in psychology and cognition [40]. After all, computer systems are used by humans.

Acknowledgments

The work described here was by and large performed by several outstanding students, especially Edi Shmueli, Netanel Zakay, and Dan Tsafrir. Our work was supported by the Israel Science Foundation (grants no. 219/99 and 167/03) and the Ministry of Science and Technology, Israel.

References

[1] S. J. Chapin, W. Cirne, D. G. Feitelson, J. P. Jones, S. T. Leutenegger, U. Schwiegelshohn, W. Smith, and D. Talby, "*Benchmarks and standards for the evaluation of parallel job schedulers*". In *Job Scheduling Strategies for Parallel Processing*, D. G. Feitelson and L. Rudolph (eds.), pp. 67–90, Springer-Verlag, 1999, DOI: 10.1007/3-540-47954-6_4. Lect. Notes Comput. Sci. vol. 1659.

[2] W. Cirne and F. Berman, "*A comprehensive model of the supercomputer workload*". In 4th *Workshop on Workload Characterization*, pp. 140–148, Dec 2001, DOI: 10.1109/WWC.2001.990753.

[3] P. J. Denning, "*Performance analysis: Experimental computer science at its best*". *Comm. ACM* **24(11)**, pp. 725–727, Nov 1981, DOI: 10.1145/358790.358791.

[4] A. B. Downey, "*A parallel workload model and its implications for processor allocation*". *Cluster Comput.* **1(1)**, pp. 133–145, 1998, DOI: 10.1023/A:1019077214124.

[5] A. B. Downey and D. G. Feitelson, "*The elusive goal of workload characterization*". *Performance Evaluation Rev.* **26(4)**, pp. 14–29, Mar 1999, DOI: 10.1145/309746.309750.

[6] B. Efron, "*Bootstrap methods: Another look at the jackknife*". *Ann. Statist.* **7(1)**, pp. 1–26, Jan 1979, DOI: 10.1214/aos/1176344552.

[7] 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.

[8] D. G. Feitelson, "*Memory usage in the LANL CM-5 workload*". In *Job Scheduling Strategies for Parallel Processing*, D. G. Feitelson and L. Rudolph (eds.), pp. 78–94, Springer-Verlag, 1997, DOI: 10.1007/3-540-63574-2_17. Lect. Notes Comput. Sci. vol. 1291.

[9] D. G. Feitelson, "*Metrics for parallel job scheduling and their convergence*". In *Job Scheduling Strategies for Parallel Processing*, D. G. Feitelson and L. Rudolph (eds.), pp. 188–205, Springer-Verlag, 2001, DOI: 10.1007/3-540-45540-X_11. Lect. Notes Comput. Sci. vol. 2221.

[10] D. G. Feitelson, "*The forgotten factor: Facts; on performance evaluation and its dependence on workloads*". In *Euro-Par 2002 Parallel Processing*, B. Monien and R. Feldmann (eds.), pp. 49–60, Springer-Verlag, Aug 2002, DOI: 10.1007/3-540-45706-2_4. Lect. Notes Comput. Sci. vol. 2400.

[11] 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.

[12] D. G. Feitelson, "*Metric and workload effects on computer systems evaluation*". *Computer* **36(9)**, pp. 18–25, Sep 2003, DOI: 10.1109/MC.2003.1231190.

[13] 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.

[14] D. G. Feitelson, "*Experimental computer science: The need for a cultural change*". URL http://www.cs.huji.ac.il/~feit/papers/exp05.pdf, 2005.

[15] 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.

[16] D. G. Feitelson, "*Looking at data*". In 22nd *Intl. Parallel & Distributed Processing Symp.*, Apr 2008, DOI: 10.1109/IPDPS.2008.4536092.

[17] D. G. Feitelson, *Workload Modeling for Computer Systems Performance Evaluation*. Cambridge University Press, 2015.

[18] D. G. Feitelson, "*Resampling with feedback — a new paradigm of using workload data for performance evaluation*". In *European Conference on Parallel Processing (Euro-Par)*, P.-F. Dutot and D. Trystram (eds.), pp. 3–21, Springer-Verlag, Aug 2016, DOI: 10.1007/978-3-319-43659-3_1. Lect. Notes Comput. Sci. vol. 9833.

[19] D. G. Feitelson and A. W. Mu'alem, "*On the definition of "on-line" in job scheduling problems*". *SIGACT News* **36(1)**, pp. 122–131, Mar 2005, DOI: 10.1145/1052796.1052797.

[20] D. G. Feitelson and A. Mu'alem Weil, "*Utilization and predictability in scheduling the IBM SP2 with backfilling*". In 12th *Intl. Parallel Processing Symp.*, pp. 542–546, Apr 1998, DOI: 10.1109/IPPS.1998.669970.

[21] D. G. Feitelson and M. Naaman, "*Self-tuning systems*". *IEEE Softw.* **16(2)**, pp. 52–60, Mar/Apr 1999, DOI: 10.1109/52.754053.

[22] D. G. Feitelson and B. Nitzberg, "*Job characteristics of a production parallel scientific workload on the NASA Ames iPSC/860*". In *Job Scheduling Strategies for Parallel Processing*, D. G. Feitelson and L. Rudolph (eds.), pp. 337–360, Springer-Verlag, 1995, DOI: 10.1007/3-540-60153-8_38. Lect. Notes Comput. Sci. vol. 949.

[23] D. G. Feitelson and L. Rudolph, "*Distributed hierarchical control for parallel processing*". *Computer* **23(5)**, pp. 65–77, May 1990, DOI: dx.doi.org/10.1109/2.53356.

[24] D. G. Feitelson and L. Rudolph, "*Evaluation of design choices for gang scheduling using distributed hierarchical control*". *J. Parallel & Distributed Comput.* **35(1)**, pp. 18–34, May 1996, DOI: 10.1006/jpdc.1996.0064.

[25] D. G. Feitelson and L. Rudolph, "*Metrics and benchmarking for parallel job scheduling*". In *Job Scheduling Strategies for Parallel Processing*, D. G. Feitelson and L. Rudolph (eds.), pp. 1–24, Springer-Verlag, 1998, DOI: 10.1007/BFb0053978. Lect. Notes Comput. Sci. vol. 1459.

[26] 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.

[27] 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.

[28] 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.

[29] 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.

[30] 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, DOI: 10.1007/3-540-63574-2_18. Lect. Notes Comput. Sci. vol. 1291.

[31] D. Lifka, "*The ANL/IBM SP scheduling system*". In *Job Scheduling Strategies for Parallel Processing*, D. G. Feitelson and L. Rudolph (eds.), pp. 295–303, Springer-Verlag, 1995, DOI: 10.1007/3-540-60153-8_35. Lect. Notes Comput. Sci. vol. 949.

[32] 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.

[33] A. W. Mu'alem and D. G. Feitelson, "*Utilization, predictability, workloads, and user runtime estimates in scheduling the IBM SP2 with backfilling*". *IEEE Trans. Parallel & Distributed Syst.* **12(6)**, pp. 529–543, Jun 2001, DOI: 10.1109/71.932708.

[34] "*Parallel Workloads Archive*". URL http://www.cs.huji.ac.il/labs/parallel/workload/.

[35] 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.

[36] B. Schroeder and M. Harchol-Balter, "*Web servers under overload: How scheduling can help*". *ACM Trans. Internet Technology* **6(1)**, pp. 20–52, Feb 2006.

[37] 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.

[38] 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.

[39] 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.

[40] M. Snir, "*Computer and information science and engineering: One discipline, many specialties*". *Comm. ACM* **54(3)**, pp. 38–43, Mar 2011, DOI: 10.1145/1897852.1897867.

[41] D. Talby, D. G. Feitelson, and A. Raveh, "*Comparing logs and models of parallel workloads using the Co-plot method*". In *Job Scheduling Strategies for Parallel Processing*, D. G. Feitelson and L. Rudolph (eds.), pp. 43–66, Springer-Verlag, 1999, DOI: 10.1007/3-540-47954-6_3. Lect. Notes Comput. Sci. vol. 1659.

[42] D. Tsafrir, Y. Etsion, and D. G. Feitelson, "*Modeling user runtime estimates*". In *Job Scheduling Strategies for Parallel Processing*, D. G. Feitelson, E. Frachtenberg, L. Rudolph, and U. Schwiegelshohn (eds.), pp. 1–35, Springer-Verlag, 2005, DOI: 10.1007/11605300_1. Lect. Notes Comput. Sci. vol. 3834.

[43] D. Tsafrir, Y. Etsion, and D. G. Feitelson, "*Backfilling using system-generated predictions rather than user runtime estimates*". *IEEE Trans. Parallel & Distributed Syst.* **18(6)**, pp. 789–803, Jun 2007, DOI: 10.1109/TPDS.2007.70606.

[44] 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.

[45] D. Tsafrir and D. G. Feitelson, "*The dynamics of backfilling: Solving the mystery of why increased inaccuracy may help*". In *IEEE Intl. Symp. Workload Characterization*, pp. 131–141, Oct 2006, DOI: 10.1109/IISWC.2006.302737.

[46] 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.

[47] W. Willinger, M. S. Taqqu, R. Sherman, and D. V. Wilson, "*Self-similarity through high-variability: Statistical analysis of Ethernet LAN traffic at the source level*". In *ACM SIGCOMM Conf.*, pp. 100–113, 1995.

[48] 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.

[49] 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.

[50] 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.

[51] 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.

[52] D. Zotkin and P. J. Keleher, "*Job-length estimation and performance in backfilling schedulers*". In 8th *Intl. Symp. High Performance Distributed Comput.*, pp. 236–243, Aug 1999, DOI: 10.1109/HPDC.1999.805303.