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

Dror G. Feitelson(✉)

School of Computer Science and Engineering,
The Hebrew University of Jerusalem, 91904 Jerusalem, Israel
feit@cs.huji.ac.il

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 (e.g. the jobs 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 adjust 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 was written to accompany a keynote address at EuroPar 2016. 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. 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

P.-F. Dutot and D. Trystram (Eds.): Euro-Par 2016, LNCS 9833, pp. 3–21, 2016.
DOI: 10.1007/978-3-319-43659-3_1

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 [9,15]. 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 [4].

The problem is exacerbated by the fact that workloads may interact with the system and even with the performance metrics in non-trivial ways [11,12]. 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 workload should be comprehensive and include *all* possible attributes [22].

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 [19,20]. But it was not known which of these is the most realistic.

Since the mid 1990s workload logs became available (starting with [18]) and were collected in the Parallel Workloads Archive [24,29]. This enabled the substitution of assumptions with hard data [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 Sect. 3.

The suggested solution to these problems is to use resampling with feedback, as described in Sect. 4. The idea is to partition the workload into basic components, and sample from this pool of components to create multiple alternative workloads [42]. At the same time, feedback from the simulated system is used to pace the workload generation process as would occur in reality [30,32,44]. The resulting methodology enables evaluations that are not possible when using logs as they were recorded. And it 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 the rectangle represents the number of processors used, and its width represents the duration of use.

Scheduling parallel jobs is the decision of when each job 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 [26,28]. 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.

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.

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. Our work is about how to achieve this goal.

3 Using Workload Logs and Models to Drive Simulations

There are two common ways to use a measured workload to analyze or evaluate a system design: (1) use the logged workload 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

Workload models have a number of advantages over logs. Some of the most salient ones are [15, 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 h, 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.
- 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 [8], 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. [2,3,25,27]), and even a quest for a general, parameterized workload model that can serve as a canonical workload in all evaluations [21].

3.2 Problems with Models

But models include only what you know about in advance, and decide to incorporate in the model. Over the years several examples of important attributes that were missed have been uncovered.

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. But simulations often assumed that perfect estimates are available. This turned out to be wrong on two counts: first, estimates are actually very inaccurate (Fig. 2) [28], and second, it actually matters [12,38]. In retrospect we can now fully understand the interactions between estimates and other features of the workload, and the conditions under which one scheduler is better than the other. We can also model realistic (inaccurate) estimates [35]. 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.

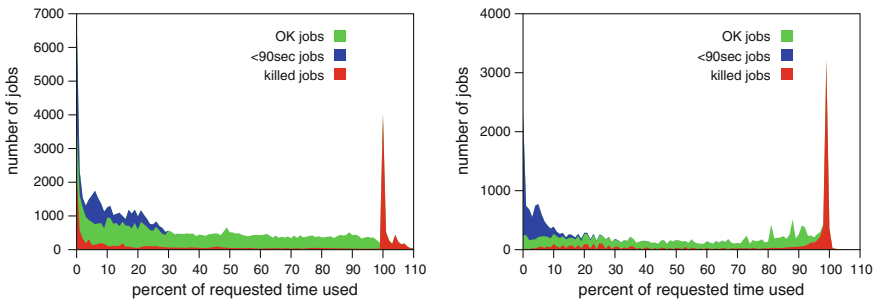

Fig. 2. Histograms of user runtime estimates as a fraction of the actual runtimes, from logs from the CTC and KTH SP2 machines [28]. The peak at 100 % is jobs killed because they exceeded their estimate; for other jobs except the very shortest ones the histogram is flat. (Color figure online)

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. And in fact

we have found that optimizing schedulers may actually depend on the existence of the daily cycle, because they try to delay non-critical jobs submitted during prime time and execute them at night [22]. If there is no daily cycle there is no non-prime time, and thus no alternative to executing these jobs at once.

Yet another effect that is prevalent in logs but usually absent from models is locality [13]. 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 [36]). 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 [17]. 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 [13], 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. 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 [9,15].

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 [18]. Over the years many additional logs have been collected in the Parallel Workloads Archive [24,29]. This resource is widely used, and as of the middle of 2016 a Google Scholar search for the archive's URL (www.cs.huji.ac.il/labs/parallel/workload) led to nearly one thousand hits.

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. The most noticeable ones are as follows:

– 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.
– Another drawback is the need for workload cleaning. Real workloads sometimes include unrepresentative activity, like huge short-lived surges of activity by individual users (flurries, Fig. 3). 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 [23,37].

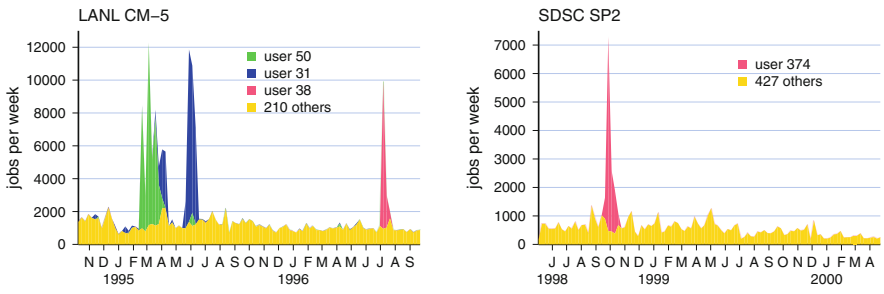

Fig. 3. Arrivals per week on two parallel supercomputers, showing flurries of activity due to single users [23,37]. (Color figure online)

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, the users change their behavior as well.

This has grave 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 [30,31]. We demonstrated this using a pair of cross-simulations of two different schedulers. The first is the well known FCFS scheduler, which is inefficient and leads to wasted resources as 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. And indeed, simulation of FCFS using a workload generated by an EASY simulation (using the same underlying system) 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 a FCFS simulation failed to show that EASY had any advantage, because the workload was not challenging enough.

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 the 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 [39]. 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 a generative approach to creating workloads, which was first proposed as a mechanism to generate locality of sampling [10]. The

idea was to view the workload as being composed from the activities of many individual users, each of which submits jobs with different characteristics [4,7]. 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 [41]). The lowest level includes repetitions of jobs within each session.

Note, however, that user-based modeling is not easy, as 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 [31]. The result was that response time was the more meaningful metric (Fig. 4).

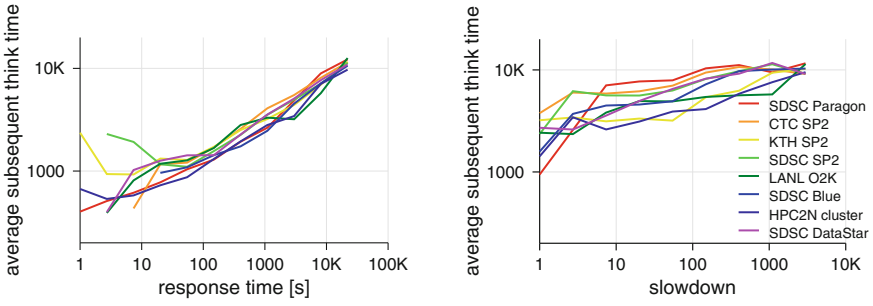

Fig. 4. A job's performance as measured by the response time is a better predictor of subsequent behavior (think time till the next job) than the job's slowdown. (Color figure online)

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 [32].
- 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 [40], which has been found in many types of workloads including parallel jobs [34].

But on the other hand, maybe all this modeling is too far removed 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 [5,6]. 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 partition a workload log into its basic components and re-group them in different ways to achieve the desired effects. In the context of parallel job scheduling, we suggest that the resampling be done at the level of users. Thus we first partition the workload into individual subtraces for the different users, including all the jobs submitted by each user throughout the logging period. We then sample from this pool of users to create a new workload [42].

When looking at individual user traces, we find that some of then 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. 5), and use them differently in the resampling. Users whose entire activity is too close to either end of the log are excluded.

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.

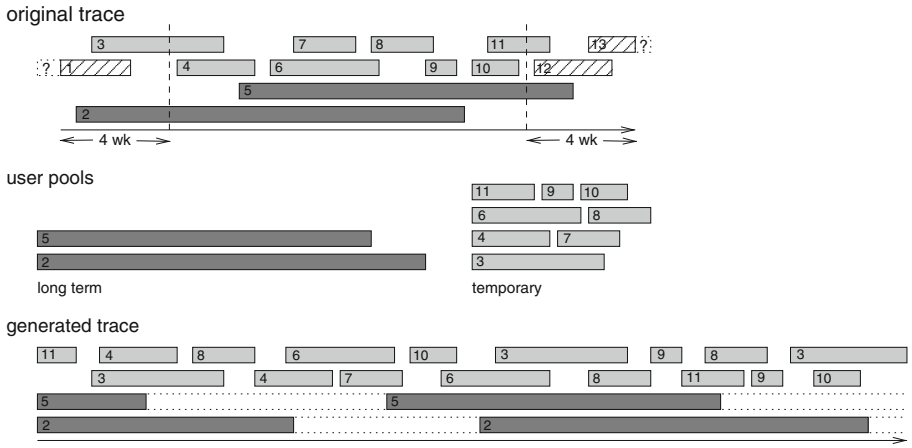

Fig. 5. Conceptual framework of dividing users into long-term and temporary, and reusing them in a generated workload [42]. Each rectangle represents the full extent of activity by a certain user.

- **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. 6.

4.3 Adding Feedback

Computer systems are not closed systems. Rather, they interact with their environment, and in particular with their users. 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 [30]. 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.

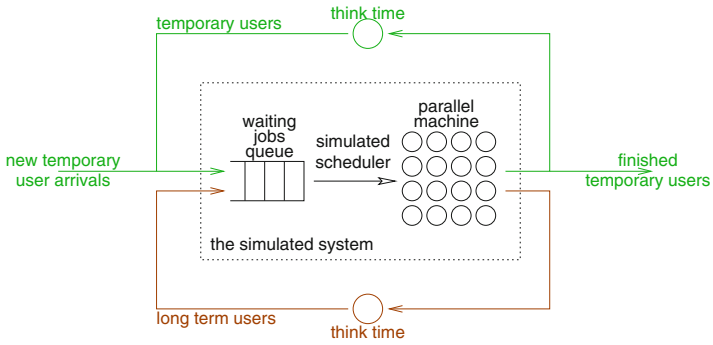

Fig. 6. Queueing model of long term and temporary users in the simulation, leading to a semi-open system [44].

The fact that jobs are submitted by simulated users might seem innocuous at first, but in fact it has grave implications. When users wait for the system before deciding that to do next they introduce a feedback loop (Fig. 7). And such feedback implies a pacing of the workload — it is a stabilizing *negative* feedback, where extra load causes the generation of additional load to be throttled [16]. This reduces the risk of system saturation.

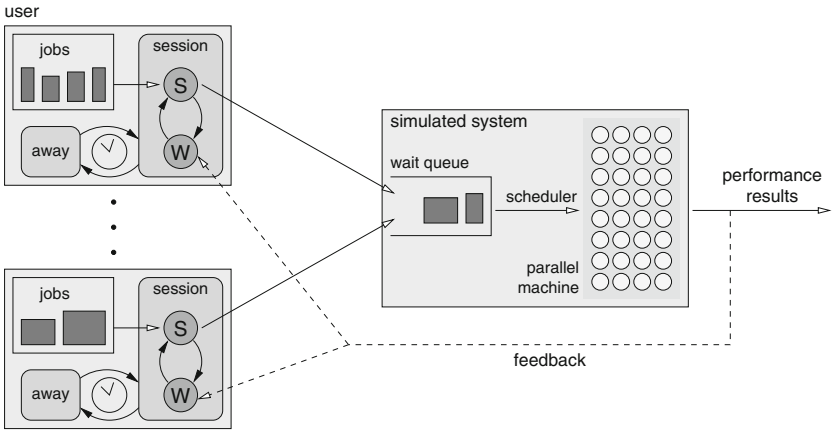

Fig. 7. Illustration of a user-based simulation with feedback. When users are in session, they alternate between submitting jobs (S) and waiting for feedback regarding previous jobs (W).

The problem with modeling the effect of feedback 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 [30,43]. These dependencies are

then used during the simulation 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. This logic is embodied in the dependencies between jobs. 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 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 [43]. Specifically, each job's submit time is adjusted to reflect feedback from the system performance to the user's behavior.

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 that, 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 [43]. 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.

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 this distribution is centered on the value which is obtained using a conventional simulation, and the result is verified (Fig. 8). 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.

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 appropriate 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. 9.

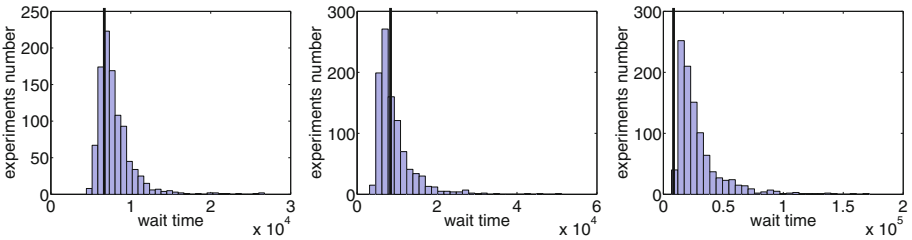

Fig. 8. Histograms of the average waiting time in a thousand simulations of EASY on resampled workloads, compared to a simulation using the original logs (vertical line). Left to right: CTC SP2, SDSC Datastar, and Blue Horizon logs. (Color figure online)

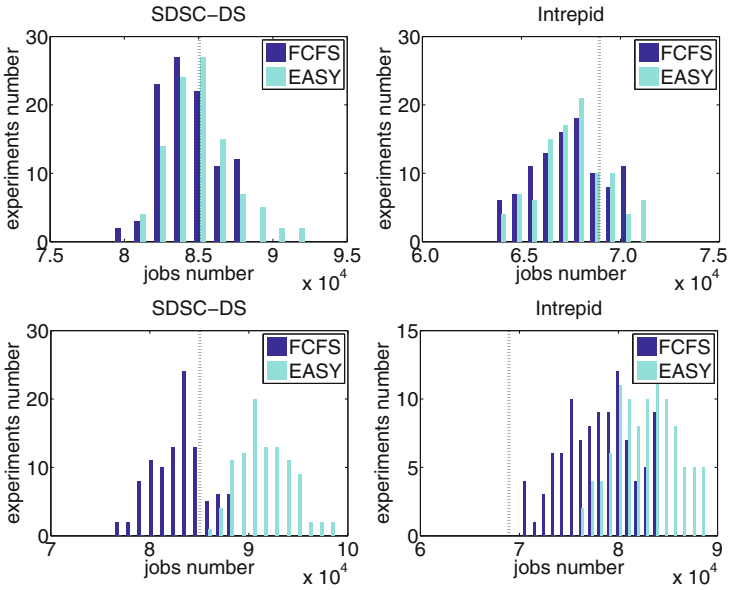

Fig. 9. Histograms of the throughput achieved in one hundred simulations of the EASY and FCFS schedulers using the SDSC DataStar and Intrepid logs [44]. Top: in simulation without feedback the throughput is determined by the input, and a better scheduler has no effect. Bottom: in simulation with feedback EASY is seen to support a higher throughput than FCFS. Vertical line represents the throughput in the original log. (Color figure online)

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 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.

We can also change the load on the system (as has been noted above) 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, 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.

Nevertheless, adding users to increase the load has two additional benefits. One is the ability to make low-load logs usable. Some logs were recorded on very low-load systems, with a utilization of only 25 % or 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.

The other and more important benefit is the ability to measure the maximal load that can be sustained before the system saturates. Resampling and user-based modeling support 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. Identifying this maximal capacity 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 number of frustrated users.

In a related vein, simulations based on resampling and feedback can be used to evaluate adaptive systems that are designed to enhance throughput (and thus productivity) rather than response time [32,44]. For example, we can design a scheduler that prioritizes jobs based on the elapsed time since the same user's previous job has terminated. The idea is that if this interval is short, there is a good chance that the user is waiting for this job. Therefore prioritizing the awaited job will enhance the productivity of this user. 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 do away with such averages, and focus on the users and the service they receive.

Finally, we note that once we partition the workload into individual users we can also look and 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.

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 the all important (but possibly unknown) details of the workload as they exist in logs recorded from production systems, with as little modifications as possible.
- 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.

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 [33]. 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. Chapin, S.J., Cirne, W., Feitelson, D.G., Jones, J.P., Leutenegger, S.T., Schwiegelshohn, U., Smith, W., Talby, D.: Benchmarks and standards for the evaluation of parallel job schedulers. In: Feitelson, D.G., Rudolph, L. (eds.) JSSPP 1999. LNCS, vol. 1659, pp. 67–90. Springer, Heidelberg (1999). doi:10.1007/3-540-47954-6_4
2. Cirne, W., Berman, F.: A comprehensive model of the supercomputer workload. In: 4th Workshop on Workload Characterization, pp. 140–148, December 2001. doi:10.1109/WWC.2001.990753
3. Downey, A.B.: A parallel workload model and its implications for processor allocation. Cluster Comput. **1**(1), 133–145 (1998). doi:10.1023/A:1019077214124
4. Downey, A.B., Feitelson, D.G.: The elusive goal of workload characterization. Perform. Eval. Rev. **26**(4), 14–29 (1999). doi:10.1145/309746.309750
5. Efron, B.: Bootstrap methods: another look at the jackknife. Ann. Statist. **7**(1), 1–26 (1979). doi:10.1214/aos/1176344552
6. Efron, B., Gong, G.: A leisurely look at the bootstrap, the jackknife, and cross-validation. Am. Stat. **37**(1), 36–48 (1983). doi:10.2307/2685844
7. Feitelson, D.G.: Memory usage in the LANL CM-5 workload. In: Feitelson, D.G., Rudolph, L. (eds.) JSSPP 1997. LNCS, vol. 1291, pp. 78–94. Springer, Heidelberg (1997). doi:10.1007/3-540-63574-2_17
8. Feitelson, D.G.: Metrics for parallel job scheduling and their convergence. In: Feitelson, D.G., Rudolph, L. (eds.) JSSPP 2001. LNCS, vol. 2221, pp. 188–205. Springer, Heidelberg (2001). doi:10.1007/3-540-45540-X_11
9. Feitelson, D.G.: The forgotten factor: facts; on performance evaluation and its dependence on workloads. In: Monien, B., Feldmann, R.L. (eds.) Euro-Par 2002. LNCS, vol. 2400, pp. 49–60. Springer, Heidelberg (2002). doi:10.1007/3-540-45706-2_4
10. Feitelson, D.G.: Workload modeling for performance evaluation. In: Calzarossa, M.C., Tucci, S. (eds.) Performance 2002. LNCS, vol. 2459, pp. 114–141. Springer, Heidelberg (2002). doi:10.1007/3-540-45798-4_6
11. Feitelson, D.G.: Metric and workload effects on computer systems evaluation. Computer **36**(9), 18–25 (2003). doi:10.1109/MC.2003.1231190
12. Feitelson, D.G.: Experimental analysis of the root causes of performance evaluation results: a backfilling case study. IEEE Trans. Parallel Distrib. Syst. **16**(2), 175–182 (2005). doi:10.1109/TPDS.2005.18
13. Feitelson, D.G.: Locality of sampling and diversity in parallel system workloads. In: 21st International Conference on Supercomputing, pp. 53–63, June 2007. doi:10.1145/1274971.1274982
14. Feitelson, D.G.: Looking at data. In: 22nd International Parallel & Distributed Processing Symposium, April 2008. doi:10.1109/IPDPS.2008.4536092
15. Feitelson, D.G.: Workload Modeling for Computer Systems Performance Evaluation. Cambridge University Press, Cambridge (2015)
16. Feitelson, D.G., Mu'alem, A.W.: On the definition of "on-line" in job scheduling problems. SIGACT News **36**(1), 122–131 (2005). doi:10.1145/1052796.1052797
17. Feitelson, D.G., Naaman, M.: Self-tuning systems. IEEE Softw. **16**(2), 52–60 (1999). doi:10.1109/52.754053
18. Feitelson, D.G., Nitzberg, B.: Job characteristics of a production parallel scientific workload on the NASA Ames iPSC/860. In: Feitelson, D.G., Rudolph, L. (eds.) JSSPP 1995. LNCS, vol. 949, pp. 337–360. Springer, Heidelberg (1995). doi:10.1007/3-540-60153-8_38

19. Feitelson, D.G., Rudolph, L.: Distributed hierarchical control for parallel processing. Computer **23**(5), 65–77 (1990). doi:10.1109/2.53356
20. Feitelson, D.G., Rudolph, L.: Evaluation of design choices for gang scheduling using distributed hierarchical control. J. Parallel Distrib. Comput. **35**(1), 18–34 (1996). doi:10.1006/jpdc.1996.0064
21. Feitelson, D.G., Rudolph, L.: Metrics and benchmarking for parallel job scheduling. In: Feitelson, D.G., Rudolph, L. (eds.) JSSPP 1998. LNCS, vol. 1459, pp. 1–24. Springer, Heidelberg (1998). doi:10.1007/BFb0053978
22. Feitelson, D.G., Shmueli, E.: A case for conservative workload modeling: parallel job scheduling with daily cycles of activity. In: 17th Modeling, Analysis & Simulation of Computer and Telecommunication Systems, September 2009. doi:10.1109/MAS-COT.2009.5366139
23. Feitelson, D.G., Tsafrir, D.: Workload sanitation for performance evaluation. In: IEEE International Symposium on Performance Analysis of Systems & Software, pp. 221–230, March 2006. doi:10.1109/ISPASS.2006.1620806
24. Feitelson, D.G., Tsafrir, D., Krakov, D.: Experience with using the Parallel Workloads Archive. J. Parallel Distrib. Comput. **74**(10), 2967–2982 (2014). doi:10.1016/j.jpdc.2014.06.013
25. Jann, J., Pattnaik, P., Franke, H., Wang, F., Skovira, J., Riodan, J.: Modeling of workload in MPPs. In: Feitelson, D.G., Rudolph, L. (eds.) JSSPP 1997. LNCS, vol. 1291, pp. 95–116. Springer, Heidelberg (1997). doi:10.1007/3-540-63574-2_18
26. Lifka, D.: The ANL/IBM SP scheduling system. In: Feitelson, D.G., Rudolph, L. (eds.) JSSPP 1995. LNCS, vol. 949, pp. 295–303. Springer, Heidelberg (1995). doi:10.1007/3-540-60153-8_35
27. Lublin, U., Feitelson, D.G.: The workload on parallel supercomputers: modeling the characteristics of rigid jobs. J. Parallel Distrib. Comput. **63**(11), 1105–1122 (2003). doi:10.1016/S0743-7315(03)00108-4
28. Mu'alem, A.W., Feitelson, D.G.: Utilization, predictability, workloads, and user runtime estimates in scheduling the IBM SP2 with backfilling. IEEE Trans. Parallel Distrib. Syst. **12**(6), 529–543 (2001). doi:10.1109/71.932708
29. Parallel Workloads Archive. http://www.cs.huji.ac.il/labs/parallel/workload/
30. Shmueli, E., Feitelson, D.G.: Using site-level modeling to evaluate the performance of parallel system schedulers. In 14th Modeling, Analysis & Simulation of Computer and Telecommunication Systems, pp. 167–176, September 2006. doi:10.1109/MAS-COTS.2006.50
31. Shmueli, E., Feitelson, D.G.: Uncovering the effect of system performance on user behavior from traces of parallel systems. In 15th Modeling, Analysis & Simulation of Computer and Telecommunication Systems, pp. 274–280, October 2007. doi:10.1109/MAS-COTS.2007.67
32. Shmueli, E., Feitelson, D.G.: On simulation and design of parallel-systems schedulers: are we doing the right thing? IEEE Trans. Parallel Distrib. Syst. **20**(7), 983–996 (2009). doi:10.1109/TPDS.2008.152
33. Snir, M.: Computer and information science and engineering: one discipline, many specialties. Comm. ACM **54**(3), 38–43 (2011). doi:10.1145/1897852.1897867
34. Talby, D., Feitelson, D.G., Raveh, A.: Comparing logs and models of parallel workloads using the co-plot method. In: Feitelson, D.G., Rudolph, L. (eds.) JSSPP 1999. LNCS, vol. 1659, p. 43. Springer, Heidelberg (1999). doi:10.1007/3-540-47954-6_3
35. Tsafrir, D., Etsion, Y., Feitelson, D.G.: Modeling user runtime estimates. In: Feitelson, D.G., Frachtenberg, E., Rudolph, L., Schwiegelshohn, U. (eds.) JSSPP 2005. LNCS, vol. 3834, pp. 1–35. Springer, Heidelberg (2005). doi:10.1007/11605300_1

36. Tsafrir, D., Etsion, Y., Feitelson, D.G.: Backfilling using system-generated predictions rather than user runtime estimates. IEEE Trans. Parallel Distrib. Syst. **18**(6), 789–803 (2007). doi:10.1109/TPDS.2007.70606
37. Tsafrir, D., Feitelson, D.G.: Instability in parallel job scheduling simulation: the role of workload flurries. In: 20th International Parallel & Distributed Processing Symposium, April 2006. doi:10.1109/IPDPS.2006.1639311
38. Tsafrir, D., Feitelson, D.G.: The dynamics of backfilling: Solving the mystery of why increased inaccuracy may help. In: IEEE International Symposium on Workload Characterization, pp. 131–141, October 2006. doi:10.1109/IISWC.2006.302737
39. Tsafrir, D., Ouaknine, K., Feitelson, D.G.: Reducing performance evaluation sensitivity and variability by input shaking. In: 15th Modelling, Analysis & Simulation of Computer and Telecommunication Systems, pp. 231–237, October 2007. doi:10.1109/MAS-COTS.2007.58
40. Willinger, W., Taqqu, M.S., Sherman, R., Wilson, D.V.: Self-similarity through high-variability: statistical analysis of Ethernet LAN traffic at the source level. In: ACM SIGCOMM Conference, pp. 100–113 (1995)
41. Zakay, N., Feitelson, D.G.: On identifying user session boundaries in parallel workload logs. In: Cirne, W., Desai, N., Frachtenberg, E., Schwiegelshohn, U. (eds.) JSSPP 2012. LNCS, vol. 7698, pp. 216–234. Springer, Heidelberg (2013). doi:10.1007/978-3-642-35867-8_12
42. Zakay, N., Feitelson, D.G.: Workload resampling for performance evaluation of parallel job schedulers. Concurrency Comput. - Pract. Exp. **26**(12), 2079–2105 (2014). doi:10.1002/cpe.3240
43. Zakay, N., Feitelson, D.G.: Preserving user behavior characteristics in trace-based simulation of parallel job scheduling. In: 22nd Modelling, Analysis & Simulation of Computer and Telecommunication Systems, pp. 51–60, September 2014. doi:10.1109/MAS-COTS.2014.15
44. Zakay, N., Feitelson, D.G.: Semi-open trace based simulation for reliable evaluation of job throughput and user productivity. In: 7th IEEE International Conference on Cloud Computing Technology and Science, pp. 413–421, November 2015. 10.1109/CloudCom.2015.35