Workload Resampling for Performance Evaluation of Parallel Job Schedulers

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

ABSTRACT

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

Categories and Subject Descriptors

C.4 [**PERFORMANCE OF SYSTEMS**]: Design studies; I.6.5 [**SIMULATION AND MODELING**]: Model Development; D.4.8 [**OPERATING SYSTEMS**]: Performance—*Simulation*; D.4.1 [**OPERATING SYSTEMS**]: Process Management—*Scheduling*

General Terms

Performance

Keywords

Workload trace; Resampling; Simulation

1. INTRODUCTION

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

Replaying a trace only provides a single data point of performance for one workload. But in many evaluations, several related workloads are needed. For example, in order to compute confidence intervals, one needs multiple instances of the same basic workload. The common way to satisfy this need is to create multiple synthetic workloads based on statistical workload models (which, in turn, are based on the traced data) [13, 2, 20, 24, 28]. While models provide the required variability and flexibility for evaluations, they also suffer from not necessarily including all the important features of the real workload [1] — in fact, they include only those of which the modeler was aware.

To improve the representativeness of evaluation workloads we propose to *combine the realism of real traces with the flexibility of models*. This will be done by modeling only the part of the workload that needs to be manipulated, and resampling from the real data to fill in the remaining details. Technically this is done by partitioning workload traces into their basic components and regrouping them in different ways to achieve the desired effects.

The domain of our work is parallel job scheduling. Parallel systems are increasingly relevant today, with the advent of multi-core processors (parallelism on the desktop), clusters and blade servers (parallelism at the enterprise level), and grids (parallelism across multiple locations). The jobs that run on parallel systems are composed of multiple processes that need to run on distinct processors (in large clusters and supercomputers the number of processes and processors can be in the thousands). When a job is submitted the user specifies how many processors are needed, and often also for how much time. The scheduler then determines the order in which jobs will be executed, and which processors will be allocated to each one. Accounting logs from large-scale systems are available in the Parallel Workloads Archive [11], and provide data about the workloads they served. (The logs we use in this work are listed in Table 1.) In particular, logs typically contain information about the submit time of each job, it's runtime and number of processes, the user who submitted it, and more. These logs can therefore be used to simulate the behavior of new scheduler designs and compare them with each other.

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

Log	File	Period	PEs	Users	Jobs
LANL CM5	LANL-CM5-1994-3.1-cln	10/94–09/96	1024	213	201,387
CTC-SP2	CTC-SP2-1996-2.1-cln	06/96–05/97	338	679	77,222
KTH-SP2	KTH-SP2-1996-2	09/96–08/97	100	214	28,489
SDSC-SP2	SDSC-SP2-1998-3.1-cln	04/98–04/00	128	437	59,725
OSC-clust	OSC-Clust-2000-3.1-cln	01/00–11/01	178	253	36,097
SDSC-BLUE	SDSC-BLUE-2000-3.1-cln	04/00–01/03	1152	468	243,314
HPC2N	HPC2N-2002-1.1-cln	07/02–01/06	240	257	202,876
SDSC-DS	SDSC-DS-2004-1	03/04–04/05	1664	460	96,089
Intrepid	ANL-Intrepid-2009-1	01/09–09/09	163,840	236	68,936

Table 1: *Logs from the Parallel Workloads Archive (www.cs.huji.ac.il/labs/parallel/workload/) that were used in this study.*

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

Importantly, while the resampled workloads differ from the original in length, statistical variation, or load, they nevertheless retain important elements of the internal structure such as sessions and the relationship between the sessions and the daily work cycle.

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

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

2. WHY USE RESAMPLING

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

Resampling is a powerful technique for statistical reasoning in such situations, when not enough empirical data is available [6, 7]. The idea is to use the available data sample as an approximation of the underlying population, and resample from it. This enables multiple, quasi-independent samples to be created, which are then used to compute confidence intervals or other metrics of interest that depend on the unknown underlying distribution.

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

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

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

We note that while we believe such resampling to be relatively novel in the context of computer workloads and performance evaluation, analogies from other fields of computer science do exist. One analogy comes from computer graphics, where texture mapping is often done by replicating a small patch of texture, with certain variations to give an impression of perspective, conform to lighting conditions, and avoid an obvious tiling effect [17]. More relevant to our work on workloads, such replication, modification, and patching together has also been done for temporal signals, such as movement specification [15] and sound [4]. Another analogy comes from the joint time-space analysis of video. Here the idea is to partition a video into patches, and then replace certain patches with others, e.g. to reconstruct missing frames or add or remove objects [30]. This technique can also be used for anomaly detection: if a piece of a new video cannot be reconstructed from snippets that exist in the system's database, then it is anomalous [3].

To the best of our knowledge resampling-based workload manipulations as we propose here have been used only in a very limited manner. The closest related work we know of is the Tmix tool, used for the generation of networking traffic. This tool extracts communication vectors describing different connections from a traffic log (sequences of $\langle \text{req\_bytes}, \text{resp\_bytes}, \text{think\_time} \rangle$) and then replays them subject to feedback from the simulated system's performance [29]. A subsequent paper also mentions the possibility of resampling traces to create diverse load conditions [12], but their approach is simpler than ours as they do not use the concept of sessions nor retain phenomena like the daily cycle. A similar construction was proposed by Krishnamurthy et al. in the context of evaluating e-commerce systems [16]. In this case they reuse sequences of user operations in order to ensure that illegal sequences are not generated by mistake by the workload modeling procedure. In the domain of parallel job scheduling, Ernemann et al. resize and replicate jobs in order to make a trace suitable for simulations with a larger machine [8]. Our goal, in contrast, is to use the resampled traces to perform better and more comprehensive evaluations. Kamath et al. have suggested to merge several traces and simulate a queueing mechanism in order to increase load [14]. However, this is limited to load values that are the sums of loads from existing traces. Ebling and Satyanarayanan created micro-models of application file behaviors based on a trace, and then combined them stochastically to create test workloads [5]. Again this is similar in concept, except that we prefer to use real data directly at the bottom layer rather than to risk models that may miss important details.

3. GRANULARITY OF RESAMPLING

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

Resampling at the job level is similar to resampling in statistics, e.g. as applied in the bootstrap method [6], which is similar to using the empirical distribution as a model. Note, however, that by resampling complete jobs we retain the correlations between job attributes (e.g. job size and runtime), which would be lost if we resampled from each marginal distribution independently.

Resampling is all about restructuring the workload. But at the same time, we wish to retain at least some of the local structure. Specifically, we typically want to retain the locality properties exhibited by normal work practices. Also, it may be important to retain the structure of batches of related jobs or sessions. For example, this is necessary for the evaluation of adaptive systems that learn about their workload and adapt to changing workload conditions [25, 9]; without locality and structure, such systems don't have what to exploit.

To motivate the use of resampling at the user level, we studied the correlation between each user's jobs. First we divided the work of each user into sessions [31]. Then we compared the jobs in one session with each other and with the jobs in subsequent sessions, using three attributes: the number of processors used, the jobs' runtimes, and their estimated runtimes. The metric for comparison was the ratio of the smaller value as part of the larger one: $r = \min\{j1.att, j2.att\}/\max\{j1.att, j2.att\}$. This is by definition in the range $[0, 1]$, with 0 indicating a large difference and 1 indicating identity.

Fig. 1 shows a sample of the results, using logs available from the Parallel Workloads Archive [21]. The X axis is the distance in sessions between the compared jobs, and the Y axis shows the average or median of the level of similarity. Obviously jobs in the same session tend to be more similar to each other, and the degree of similarity is reduced with distance. In effect, this testifies to the existence of locality in these workloads, which we want to retain.

To validate this result, we used bootstrapping to compare the results shown above with results that would be obtained if we sample jobs independently. To do so we retain the structure of sessions for each user, but mix the jobs randomly among the sessions. This is repeated 1000 times, and each time the degrees of similarity between jobs in the same session are computed as above. Fig. 2 shows a sample of the results. Obviously, the similarity among jobs that

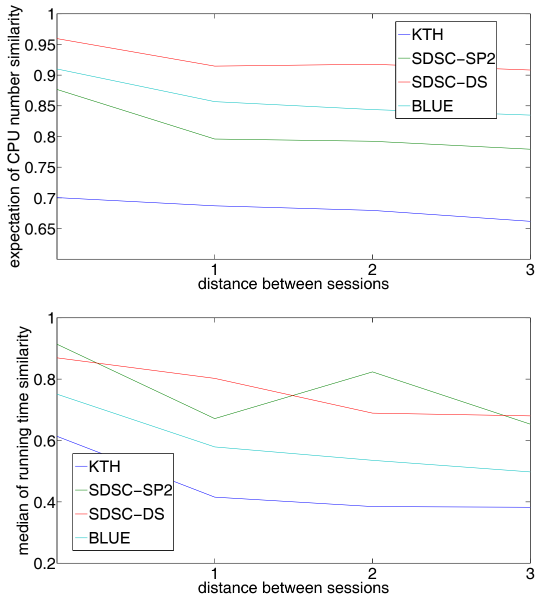

Figure 1: *Similarity between jobs as a function of the distance between them in sessions.*

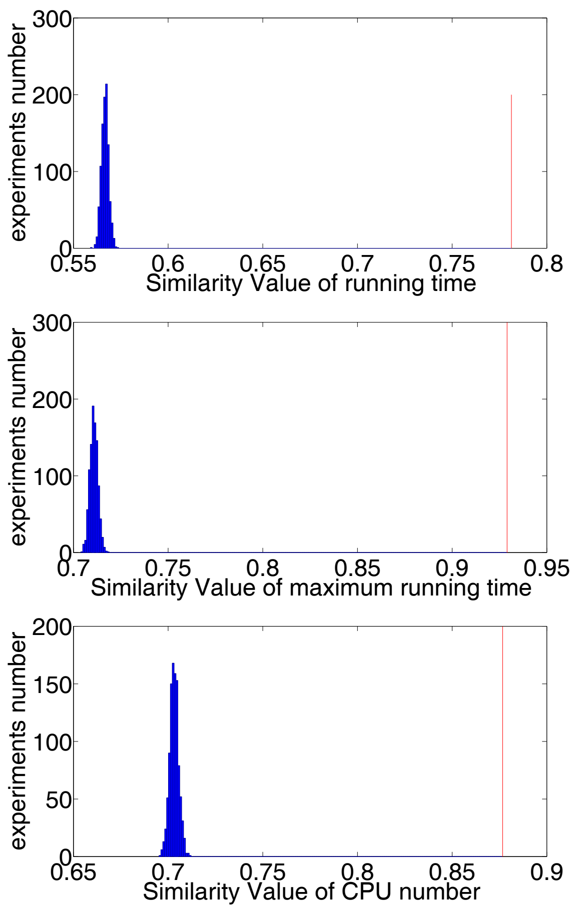

Figure 2: *Comparison of the similarity between jobs in the same session as computed from the SDSC-SP2 log (vertical line), and the distribution of similarity levels that are seen when the jobs are randomized.*

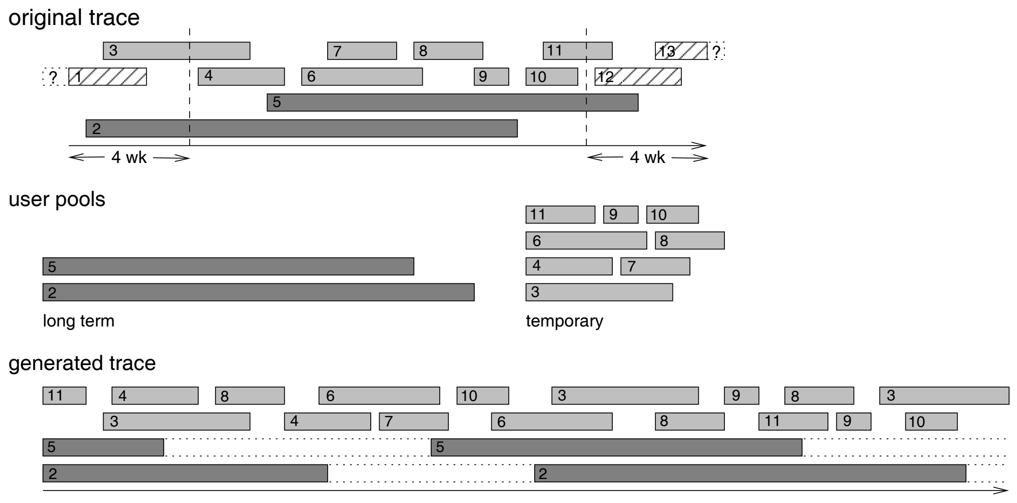

Figure 3: *Conceptual framework of dividing users into long-term and temporary, and reusing them in a generated trace.*

appear together in the original log is much higher than the similarity that would be obtained if jobs are randomized, as would happen if we resample individual jobs.

An implicit assumption in our resampling procedure is that users are independent. This is not strictly valid because users affect each other: if one user overloads the system, others may feel this and reduce their own activity. However, a large part of such interactions is due to all users operating on the same daily cycle, and we take care to retain this correlation between the resampled users. Moreover, resampling at the user level rather than at the session level allows for more sophisticated user behavior models. Specifically, we can introduce feedback effects whereby a user may decide to terminate a session because system performance is poor, and submit his subsequent jobs in a later session. Our work on incorporating such feedback effects will be reported separately; for now the main point is that if we resample at the session granularity such effects will be effectively excluded.

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

4. MECHANICS OF RESAMPLING

Creating a new workload by resampling users means that we dissect the given trace into sub-traces representing different users, and then recombine these sub-traces in different ways. Note that we do not manipulate each user's sub-trace. Thus the sequence of jobs representing each user will be the same as in the original trace, and the intervals between them will also be the same. This guarantees the same locality properties as in the original trace, as noted above. We also take care to synchronize the resampled users using a common timeframe, so that jobs always start on the same day of the week and the same time of the day as in the original trace. This ensures that the daily cycle of activity is retained in the produced workload, which may be important [32, 10].

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

	long-term		temporary	
Log	users	jobs	users	jobs
CTC-SP2	314	63,287	236	10,625
KTH-SP2	102	25,202	66	2,349
SDSC-SP2	173	44,251	206	8,790
SDSC-BLUE	426	221,745	31	1,435
HPC2N	178	194,429	66	7,949
SDSC-DS	230	74,764	192	9,012

Table 2: *Results of classifying users in the different logs.*

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

In analyzing the log, we distinguish between temporary users and long-term users according to the interval between their first job and their last job in the log. If the interval is long enough (above 12 weeks in our implementation), the user is classified as long-term. Otherwise the classification is temporary. The threshold of 12 weeks is chosen based on observation of the distribution of periods of activity by different users. We found that for most users their period of activity was up to about 12 weeks; these are the temporary users. For the rest there was a uniform distribution from 12 weeks to the full length of the log. This is interpreted as representing long-term users whose activity was arbitrarily intersected with the logging period. The numbers of temporary and long-term users found in different logs, and the jobs that they submitted, are shown in Table 2. There tend to be somewhat more long-term users than temporary ones, but as may be expected, the long-term users submit the vast majority of the jobs.

Data about the different users is kept in separate user pools, one for temporary users and the other for long-term users. However, temporary users whose full period of activity falls within a short time (4 weeks) from the beginning or the end of the logging period are discarded. The reason for doing so is that there is a high probability that the activity of these users was truncated, but we

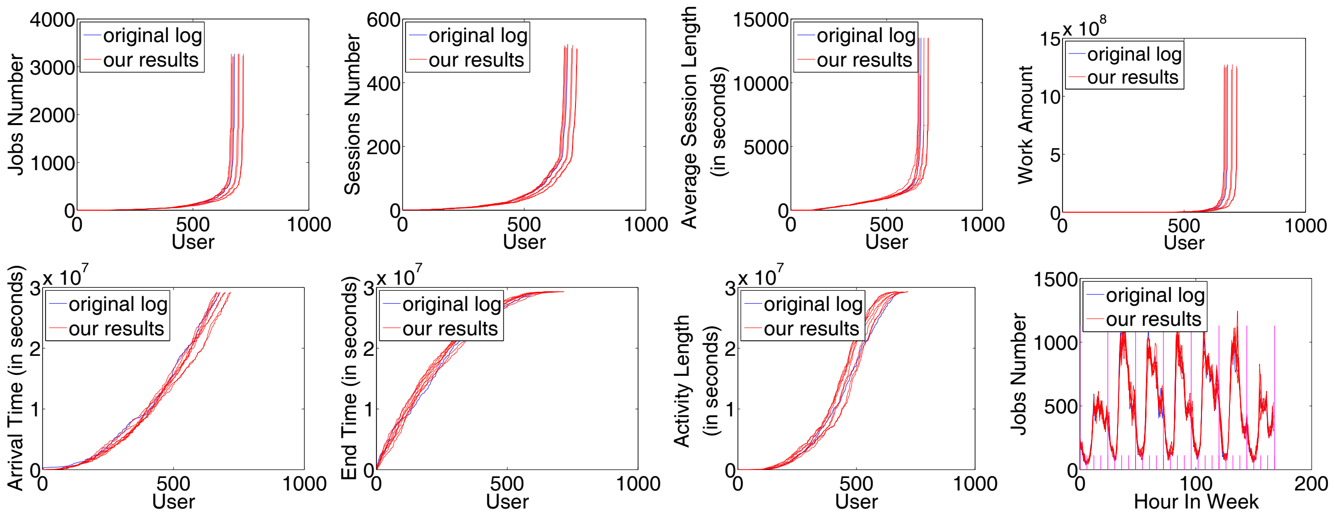

Figure 4: *Comparison between the original CTC SP2 log and 8 generated logs based on it. The last plot includes marks every 6 hours and a longer one at midnight.*

cannot know for sure. The threshold of 4 weeks is chosen because when plotting the cumulative number of users observed as a function of the number of weeks into the log, in the first few weeks the graph climbs at a higher rate. This is interpreted as being influenced by first observations of users that have already been active before. Then, when the increase settles on a lower and relatively constant average rate, this is interpreted as predominantly representing the arrivals of new users.

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

- **Initialization:** We initialize the trace with some temporary users and some long-term users. The numbers of users to use are parameters of the trace generation, and can be used to change the load or the ratio of temporary to long-term users (the defaults are the numbers of long-term users in the original log, and the average number of temporary users present in a single week of the original log). The probability to select each temporary user is proportional to the number of weeks during which the user was active in the log. Users are not started with their first job in the trace. Rather, each user is started in some arbitrary week of his traced activity. However, care is taken that jobs start on the same day of the week and time of the day in the simulation as in the original trace.
- **Temporary users:** In each new week of the simulation, a certain number of new temporary users are added. The exact number is randomized around the target number, which is a parameter of the trace generation (the default is the average rate at which temporary users arrived in the original trace). The randomization uses a binomial distribution, with a probability p equal to the fraction of temporary users expected to start every week. The selected users are started from their first traced jobs. A user can be selected from the pool multiple times, but care is taken not to select the same user twice in the same week.
- **Long-term users:** The population of long-term users is constant and consists of those chosen in the initialization. When the traced activity of a long-term user is finished, it is simply regenerated after a certain interval. While such repetitions are not very realistic, they allow us to extend the work of the long-term users as needed. We also note that repetitions only occur after rather long intervals, because logs are typically at least a year long. The interval between the regenerations corresponds to the sum of the intervals between the user's period of activity and the full logging period. Naturally the regenerations are also synchronized correctly with the time and day.

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

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

In order to perform resampling and implement the applications described in the next section it is enough to just create a new workload trace that is composed of the jobs of the different users as described above. However, we actually perform a full simulation of also scheduling these jobs. This enables us to directly use the generated workloads to evaluate various parallel job schedulers. In subsequent work we also consider adding feedback, whereby the system performance influences user behavior and may affect when subsequent jobs are submitted [23]. In any case, the simulation also creates a log file which contains the new workload. Comparing this generated workload with the original one allows us to validate the resampling process.

To validate this procedure we compare the generated workload to the original one. An example is shown in Fig. 4 based on the CTC SP2 log. The different panels show the following distributions:

- Number of jobs submitted by different users
- Number of sessions performed by users
- Average session length for different users
- Total amount of CPU time (work) used by users in all their jobs
- The users' first arrival times

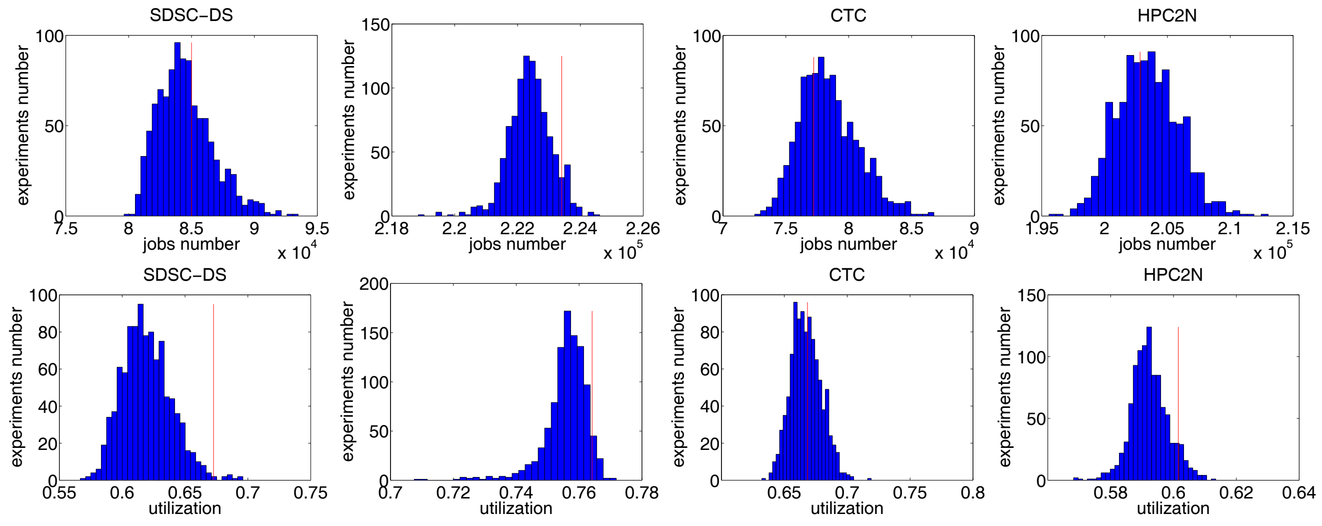

Figure 5: *Histograms of the throughput and utilization in a thousand simulations with resampled workloads compared to using the original workload (vertical red line).*

- The users' final departure times
- The users' periods of activity
- The distribution of job arrivals across days of the week, for all users

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

5. APPLICATIONS OF RESAMPLING

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

5.1 Verification of Performance Results

As noted above, one of the problems with using a workload trace directly is that it provides a single data point. This has the obvious deficiency that it is impossible to calculate any kind of confidence intervals. But with resampling we can create many resampled randomized versions of the workload, and evaluate the performance of the system with all of them, thus obtaining multiple data points that all adhere to the same underlying statistics. The distribution of these data points can then be used to compute confidence intervals for performance metrics. This is essentially an application of the well-known technique of bootstrapping used in statistical analysis [7].

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

We performed the checks on eight different logs from the Parallel Workloads Archive, and results for four of them are shown in Fig. 5. The top row shows that the throughput (that is, average number of jobs per unit time) was typically distributed around the original value. the result for the BLUE log was the largest deviation observed; with this log 92% of the variants had a lower throughput than the original log, but the difference between the median throughput and the original was only 0.45%. For utilization (bottom row) the results were more diverse, and varied between distributions around the original value — as for CTC — and distributions that are generally below the original value — as for DS, which was the most extreme. This may indicate some systematic bias which we do not understand yet. But note that the four logs that were checked and are not shown were all less extreme than BLUE or HPC2N.

Accepting the generated workload distributions as reasonable, we turn to check the results of evaluations of the EASY scheduler, which is probably the most commonly used backfilling scheduler [18]. The results for waiting time and slowdown are shown in Fig. 6 (results for response time exhibit similar behavior to wait time). The slowdown results are the most varied. In five of the eight logs we checked, the distribution was more or less around the value obtained using the original log. This is exemplified by the BLUE and CTC logs in the figure. But in other cases the original result was at the very end of the distribution, either higher or lower than nearly all the others (as in DS or HPC2N, respectively, which were the two most extreme cases). The results for slowdown were more one-sided, being distributed either around the original values (as for DS and CTC) or largely above them (as for BLUE and HPC2N). The explanation appears to be that in some logs there are more sparse days, in which very few jobs arrive and therefore all wait times are short or nil. Our resampling tends to distribute users and jobs somewhat more evenly.

For both metrics, these results underscore the importance of using the resampling methodology to identify cases where the result

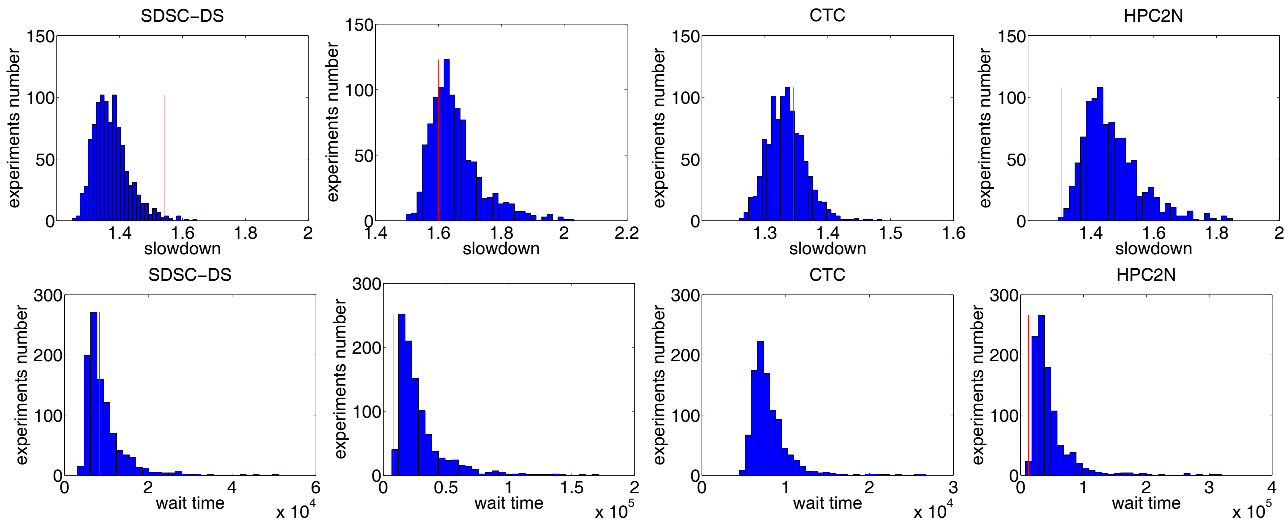

Figure 6: *Histograms of the average slowdown and waiting time in a thousand simulations of EASY on resampled workloads, compared to a simulation using the original logs (vertical red line).*

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

Note that this application of bootstrapping serves only to provide confidence intervals for evaluations based on a single log. We consider the possibility of extending this by mixing data from multiple logs in Section 5.4.1. Such mixing will provide confidence intervals for more general evaluations that are based on all the available data.

5.2 Extending a Trace

Another simple use of workload resampling is in order to extend a trace. While some of our workload traces are pretty long, with hundreds of thousands of jobs submitted over 2 years or more, others are shorter. In addition, a significant part of the trace may be needed as a "warmup period" to ensure that the simulated system achieves its steady state [22]. Given only the raw traces, the length of the simulation may therefore be quite limited.

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

To check the resulting extended workloads, we studied three repetitions of extending given traces to five times their original length. For example, given a trace that represented one year's worth of activity, we used it to create three traces that are each five years long. We then compared the original trace with the first year, the third year, and the fifth year of each repetition. The results for the BLUE log are shown in Fig. 7, using the same distributions as in Fig. 4.

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

5.3 Changing the Load

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

- Multiplying all arrival times by a constant, thus causing jobs to arrive at a faster rate and increasing the load, or causing them to arrive at a slower rate and decreasing the load. However, this also changes the daily cycle, for example causing jobs that were supposed to terminate during the night to extend into the next day. An alternative approach that has a similar effect is to multiply all runtimes by a constant. This has the deficiency of creating an artificial correlation between load and response time.
- Multiplying all job sizes (here meaning the number of processors they use) by a constant, and rounding to the nearest integer. This has two deficiencies. First, many jobs and machine sizes are powers of two. After multiplying by some constant in order to change the load, they will not be powers of two, which may have a strong effect on how they pack, and thus on the observed fragmentation. This effect can be much stronger than the performance effects we are trying to measure [19]. Second, small jobs cannot be changed with suitable fidelity as the sizes must always be integers. An alternative approach that has essentially the same effect is to modify the machine size. This at least avoids the problem presented by the small jobs.

With resampling, however, manipulating the load is relatively easy: One can simply increase or reduce the average number of active users. This changes the load while retaining all other attributes of the workload and avoiding the introduction of any artifacts. In particular, some logs have a very low utilization, in the range of 10–30%, which makes them uninteresting in terms of evaluating

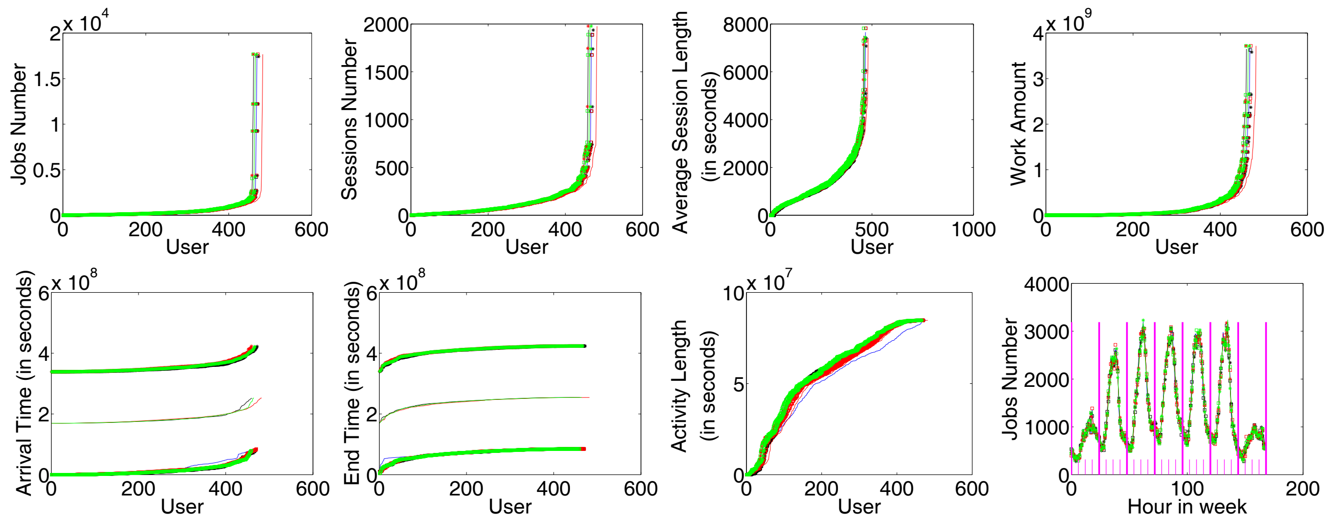

Figure 7: *Comparison between the original BLUE log to the first, third, and fifth parts of an extended resampled log that is 5 times longer.*

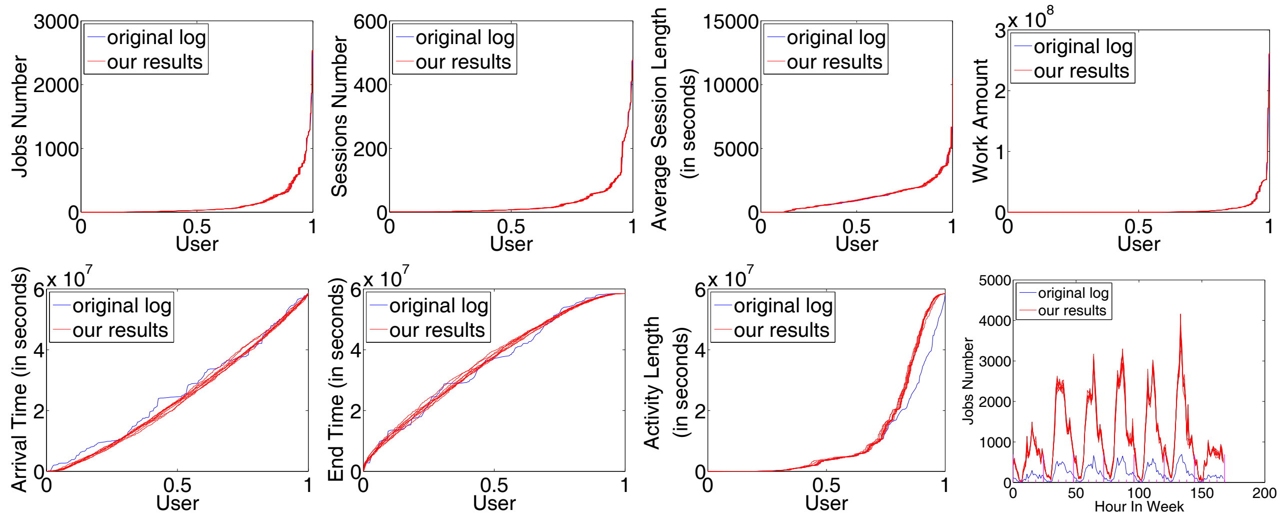

Figure 8: *Comparison between the original OSC-Cluster log to resampled workloads where the load is increased by a factor of 6.*

schedulers for parallel machines (because there are seldom enough concurrent jobs for the scheduler to have to make any decisions). Using resampling we can increase the load significantly and make these logs usable.

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

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

1. Tabulate the number of outstanding jobs at the beginning of each week of the simulation.
2. If the number of outstanding jobs grows due to a load fluctuation, but then decreases again, this does not indicate saturation. Therefore we replace each weekly count by the minimum count from that week to the end of the simulation, leading to a non-decreasing sequence of counts.
3. Delete the last 20% of the values, to avoid false positives based on fluctuations that occur towards the end of the simulation.
4. Use linear regression to fit a straight line to the remaining counts. If the slope is lower than 1 (meaning that on average the number of outstanding jobs grows by no more than one job per week) the simulation is declared stable. If it is higher, the simulated system is saturated.

Verifying that resampling with a modified number of users leads to reasonable workloads shows that indeed all the distributions are similar to those of the original traces (but taking into account that

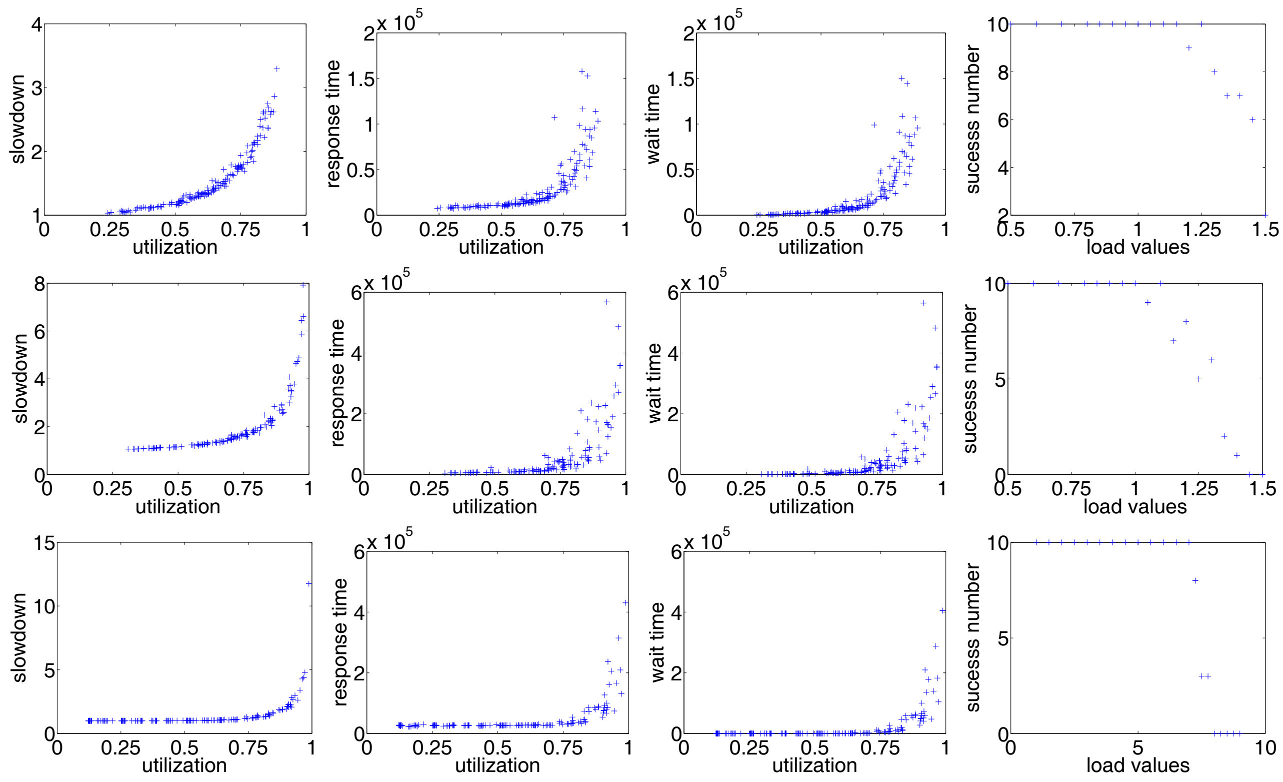

Figure 9: *The performance of EASY under different load conditions for different logs: SDSC-DS, BLUE, and OSC-Cluster respectively from top to bottom. Recall that the original OSC cluster log has very low load, so the load had to multiplied by higher factors to reach the range of interest.*

the number of users is different). Fig. 8 shows the results for one extreme case, based on the OSC cluster log. The average utilization of this log is only 12.8%, making it unusable for evaluations of parallel job schedulers. We therefore increased the number of users by a factor of 6, targeting an average utilization of approximately 76.8%. In the graphs, the user numbers on the X axis are normalized to the range $[0, 1]$ to enable comparison with the original log that has much fewer users. It is easy to see that the high-load simulations produce distributions that are very similar to the original log. The main difference is in the arrival time and end time distributions, which are smoother, because in our simulations users arrive at a constant average rate. Also, in the last graph portraying the weekly cycle of activity, one can see the big difference in the number of jobs that are being used.

The goal of all these workload manipulations is to enable the evaluation of parallel job schedulers, and in particular, their performance under different load conditions. To check this we again used simulations of the EASY backfilling scheduler [18]. For each log, we multiplied the number of users by various factors in the range 0.8 to 1.5, and performed 10 independent simulations (with different randomized resampling) for each load value. For the OSC cluster log, the range was from 1 to 9, because the original utilization of this log is very low as noted above. A sample of the results in terms of slowdown, response time, and waiting time are shown in Fig. 9.

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

As the results in Fig. 9 show, the performance profiles are exactly as one might expect from queueing analysis. At low loads performance is good, and increasing the load has little effect. But as the system approaches saturation, the performance deteriorates precipitously. Interestingly, different systems (as represented by the logs of their workloads) have different saturation points. SDSC-DS seems to saturate at less than 90% utilization, whereas BLUE and OSC come close to 100%. This reflects the ability of the scheduler to pack jobs together and reduce fragmentation, and depends both on the scheduler and on the workload statistics.

5.4 Additional Applications

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

5.4.1 Mixing Traces

In many cases we have more than one trace at our disposal, typically coming from different locations or different times. To obtain generally applicable results, data from all these traces should be used. This can be done either by performing evaluations based on each trace individually, or by mixing the traces, that is by resampling from a number of traces rather than from only one trace. This mechanism has been used in the past in order to reduce the dependence of analysis results on a single trace [32], or to increase the load [14]. It was also suggested for Tmix [29].

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

5.4.2 *Improving Stationarity*

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

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

5.4.3 *Improved Shaking to Reduce Sensitivity*

Simulations of system performance are sometimes very sensitive to the exact value of some input parameter. For example, we have found a specific case where changing the runtime of one job from 18 hours and 30 seconds to exactly 18 hours caused the average bounded slowdown of *all* the jobs in the trace to change by about 8% [26]. We developed "shaking" as a general methodology to overcome such mishaps [27].

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

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

5.4.4 *Selective Manipulation of the Workload*

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

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

5.4.5 *Reducing or Enhancing Locality*

Finally, a special case of manipulating the workload is changing its locality properties. Locality can be very meaningful for adaptive systems that learn about their workload and adjust accordingly [9].

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

Enhancing locality can be done by introducing repetitions, i.e. specifically selecting the same jobs over and over again [9]. However, this also needs to be done subject to stability constraints, and subject to the overall statistical properties of the workload.

6. CONCLUSIONS

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

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

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

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

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

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

7. REFERENCES

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

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

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

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

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

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

[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.

[8] C. Ernemann, B. Song, and R. Yahyapour, "*Scaling of workload traces*". In *Job Scheduling Strategies for Parallel Processing*, pp. 166–182, Springer Verlag, 2003. Lect. Notes Comput. Sci. vol. 2862.

[9] D. G. Feitelson, "*Locality of sampling and diversity in parallel system workloads*". In 21st *Intl. Conf. Supercomputing*, pp. 53–63, Jun 2007.

[10] 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.

[11] D. G. Feitelson, D. Tsafrir, and D. Krakov, "*Experience with the parallel workloads archive*", 2012. (In preparation).

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

[13] J. Jann, P. Pattnaik, H. Franke, F. Wang, J. Skovira, and J. Riodan, "*Modeling of workload in MPPs*". In *Job Scheduling Strategies for Parallel Processing*, pp. 95–116, Springer Verlag, 1997. Lect. Notes Comput. Sci. vol. 1291.

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

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

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

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

[18] D. Lifka, "*The ANL/IBM SP scheduling system*". In *Job Scheduling Strategies for Parallel Processing*, pp. 295–303, Springer-Verlag, 1995. Lect. Notes Comput. Sci. vol. 949.

[19] V. Lo, J. Mache, and K. Windisch, "*A comparative study of real workload traces and synthetic workload models for parallel job scheduling*". In *Job Scheduling Strategies for Parallel Processing*, pp. 25–46, Springer Verlag, 1998. Lect. Notes Comput. Sci. vol. 1459.

[20] 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.

[21] "*Parallel workloads archive*". URL http://www.cs.huji.ac.il/labs/parallel/workload/.

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

[23] 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.

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

[25] D. Talby and D. G. Feitelson, "*Improving and stabilizing parallel computer performance using adaptive backfilling*". In 19th *Intl. Parallel & Distrib. Processing Symp.*, Apr 2005.

[26] D. Tsafrir and D. G. Feitelson, "*Instability in parallel job scheduling simulation: The role of workload flurries*". In 20th *Intl. Parallel & Distrib. Processing Symp.*, Apr 2006.

[27] 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.

[28] K. V. Vishwanath and A. Vahdat, "*Realistic and responsive network traffic generation*". In *ACM SIGCOMM Conf.*, pp. 111–122, Sep 2006.

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

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

[31] N. Zakay and D. G. Feitelson, "*On identifying user session boundaries in parallel workload logs*". In *Job Scheduling Strategies for Parallel Processing*, pp. 216–234, Springer-Verlag, 2012. Lect. Notes Comput. Sci. vol. 7698.

[32] J. Zilber, O. Amit, and D. Talby, "*What is worth learning from parallel workloads? a user and session based analysis*". In 19th *Intl. Conf. Supercomputing*, pp. 377–386, Jun 2005.