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

Workload Resampling for Performance Evaluation of Parallel Job Schedulers

Netanel Zakay Dror G. Feitelson*

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

SUMMARY

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

Received ...

KEY WORDS: Workload trace; Resampling; Simulation

1. INTRODUCTION

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

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

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

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

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

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

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

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

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

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

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

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

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

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

2. WHY USE RESAMPLING

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

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

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

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

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

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

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

3. GRANULARITY OF RESAMPLING

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

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

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

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

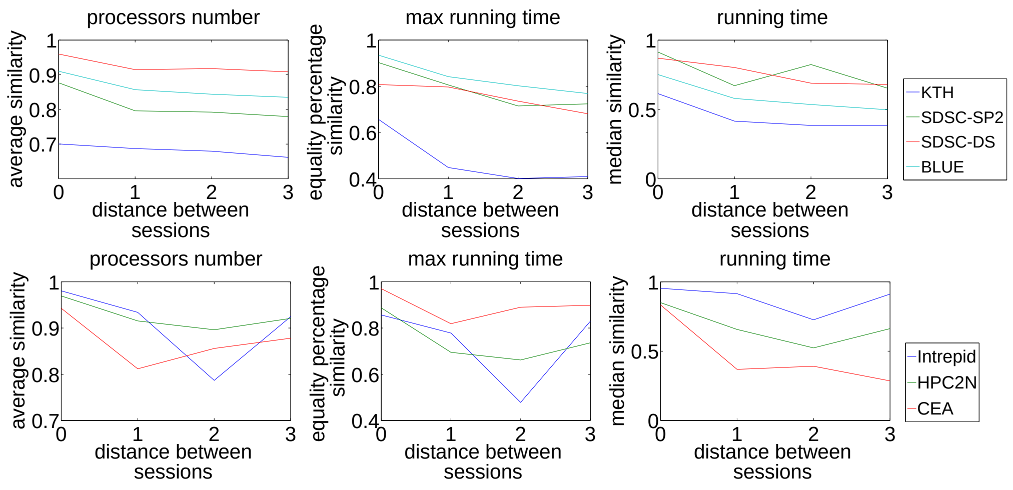

Figure 1. Similarity between jobs as a function of the distance between them in sessions.

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

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

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

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

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

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

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

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

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

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

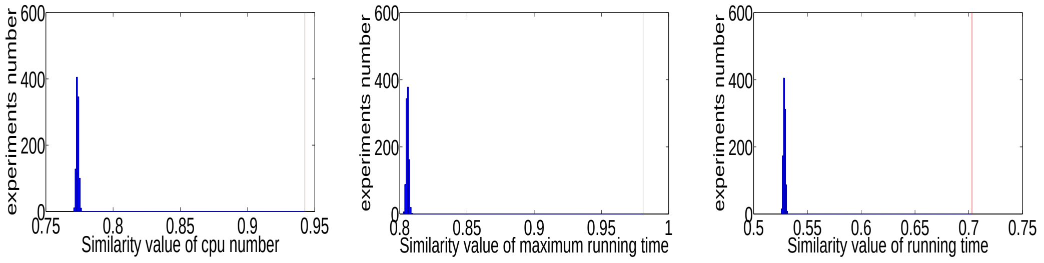

Figure 2. Comparison of the similarity between jobs in the same session as computed from the CEA-Curie log (vertical line), and the distribution of similarity levels that are seen when the jobs are randomized.

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

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

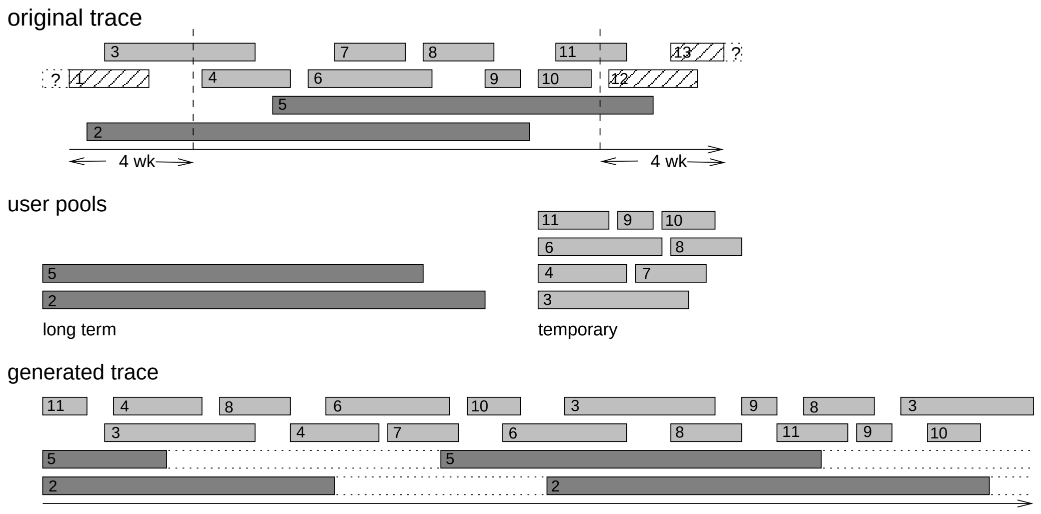

Figure 3. Conceptual framework of dividing users into long-term and temporary, and reusing them in a generated trace.

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

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

4. MECHANICS OF RESAMPLING

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

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

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

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

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

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

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

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

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

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

a probability p equal to the fraction of temporary users expected to start every week. The selected users are started from their first traced jobs. A user can be selected from the pool multiple times, but care is taken not to select the same user twice in the same week.
- **Long-term users:** The population of long-term users is constant and consists of those chosen in the initialization. When the traced activity of a long-term user is finished, it is simply regenerated after a certain interval. While such repetitions are not very realistic, they allow us to extend the work of the long-term users as needed. We also note that repetitions only occur after rather long intervals, because logs are typically at least a year long. The interval between the regenerations corresponds to the sum of the intervals between the user's period of activity and the full logging period. Naturally the regenerations are also synchronized correctly with the time and day.

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

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

5. RESAMPLING VALIDATION

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

5.1. Marginal Distributions

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

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

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

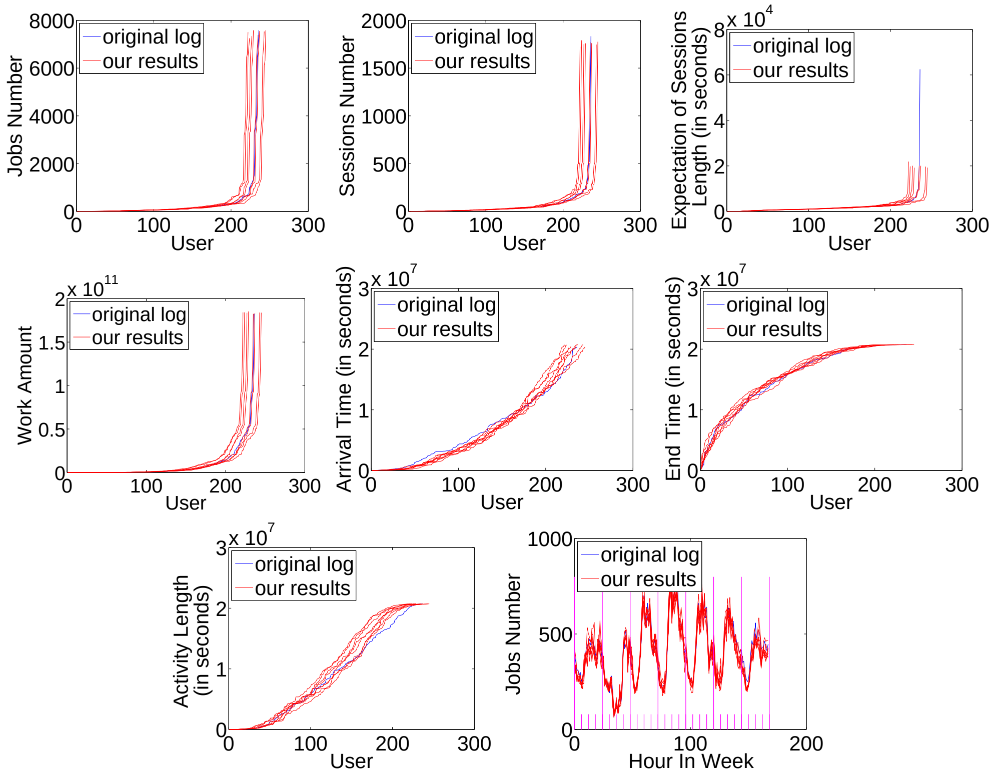

Figure 4. Comparison between various distributions of the original ANL-Intrepid log and 8 generated logs based on it. The last plot includes marks every 6 hours and a longer one at midnight.

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

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

5.2. *Locality*

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

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

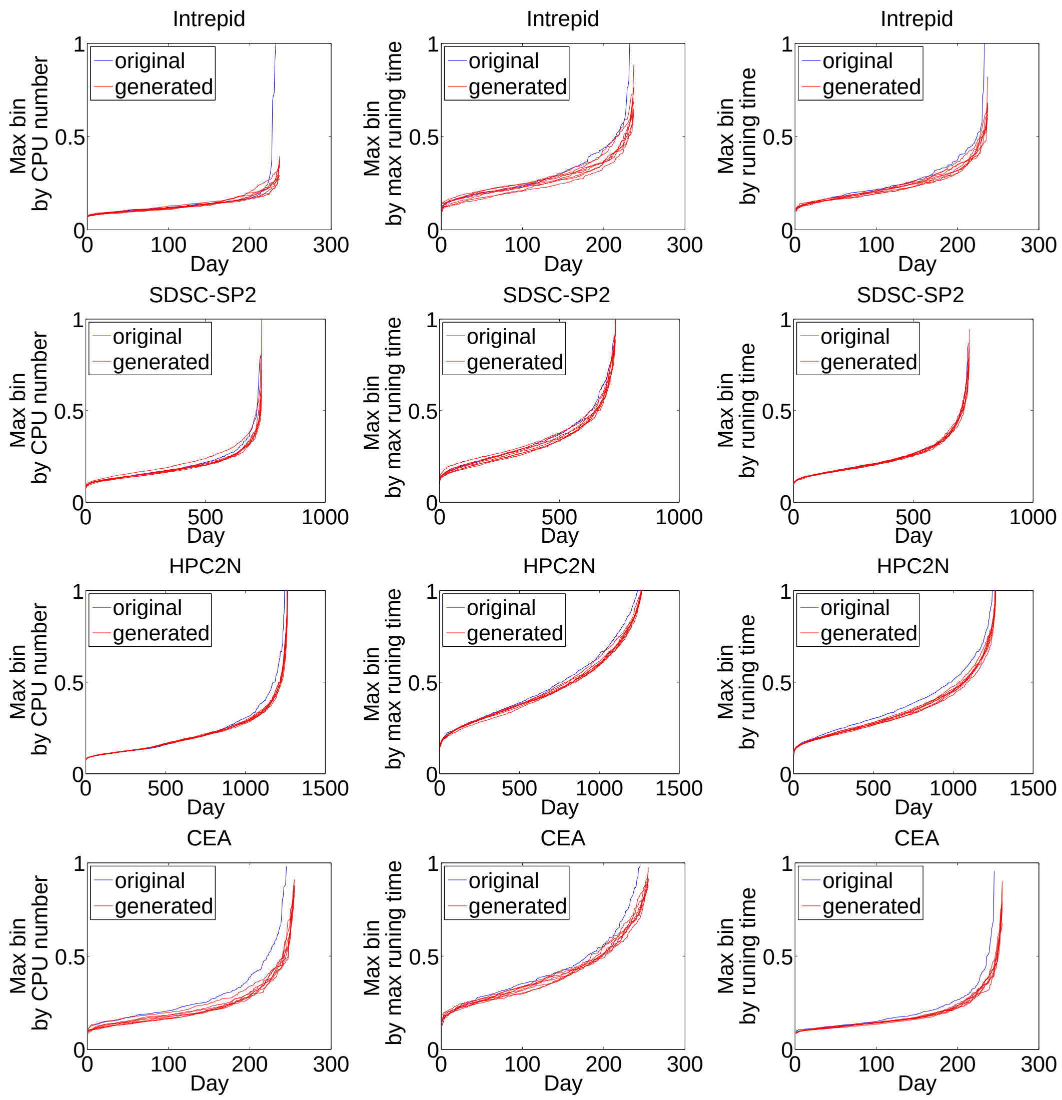

Figure 5. Using the bins-based algorithm to check locality. The graphs show the fraction of jobs in each day that are concentrated into one bin; if there was no locality, all values should be $\frac{1}{16} = 0.062$.

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

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

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

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

5.3. Self Similarity

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

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

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

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

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

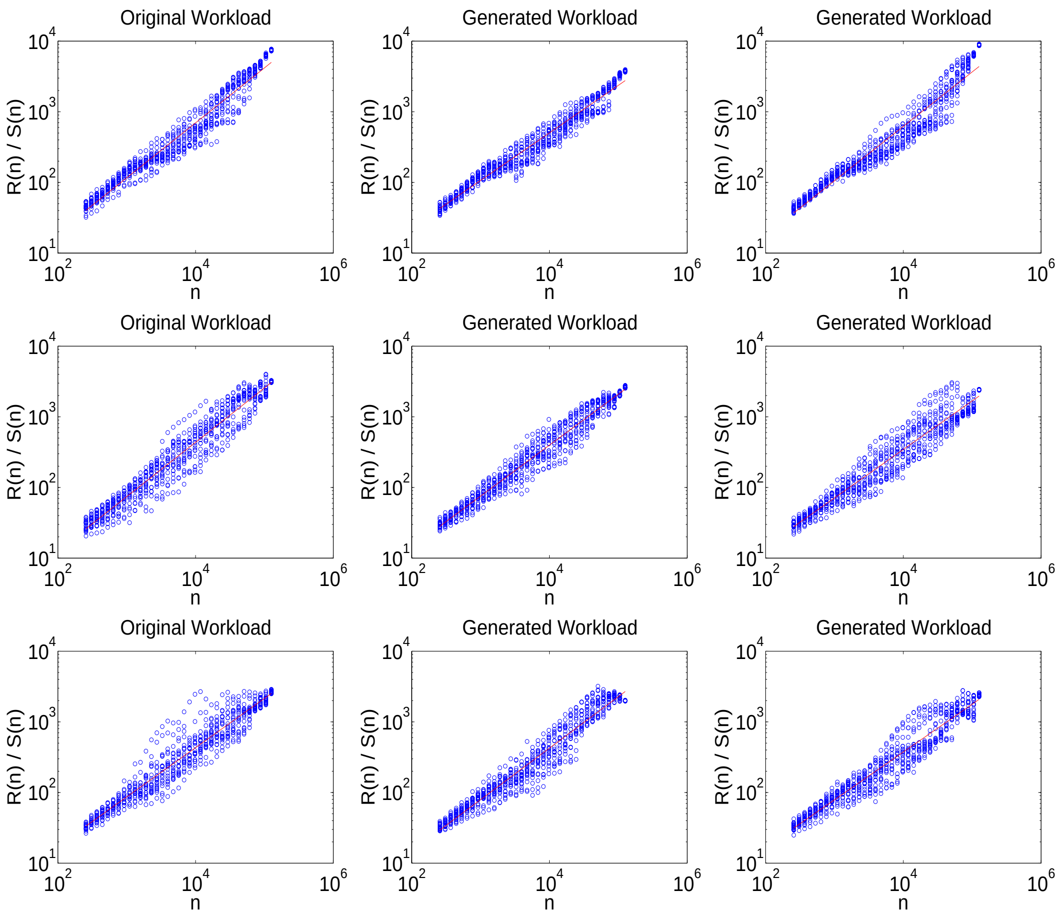

Figure 6. Comparison of Pox-plots for the original log and two resampled ones. From top: the LANL-CM5 log, the HPC2N log, and the ANL-Intrepid log.

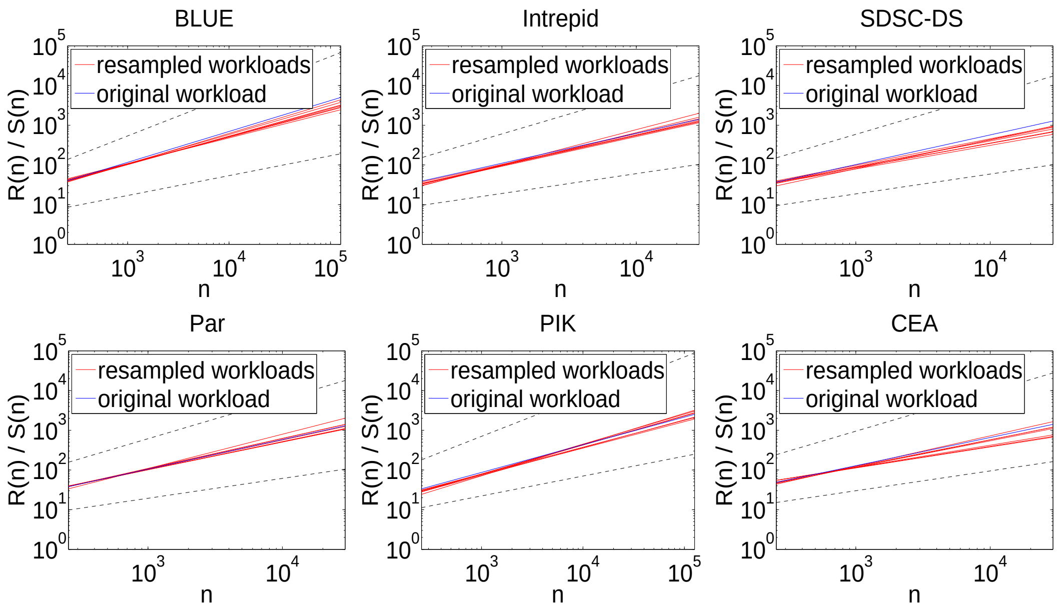

Figure 7. Comparison of the pox-plot regression lines of the original workload and the generated workloads. Dashed lines showing the slopes corresponding to $H = \frac{1}{2}$ and $H = 1$ are given for reference.

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

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

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

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

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

6. APPLICATIONS OF RESAMPLING

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

6.1. *Verification of Performance Results*

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

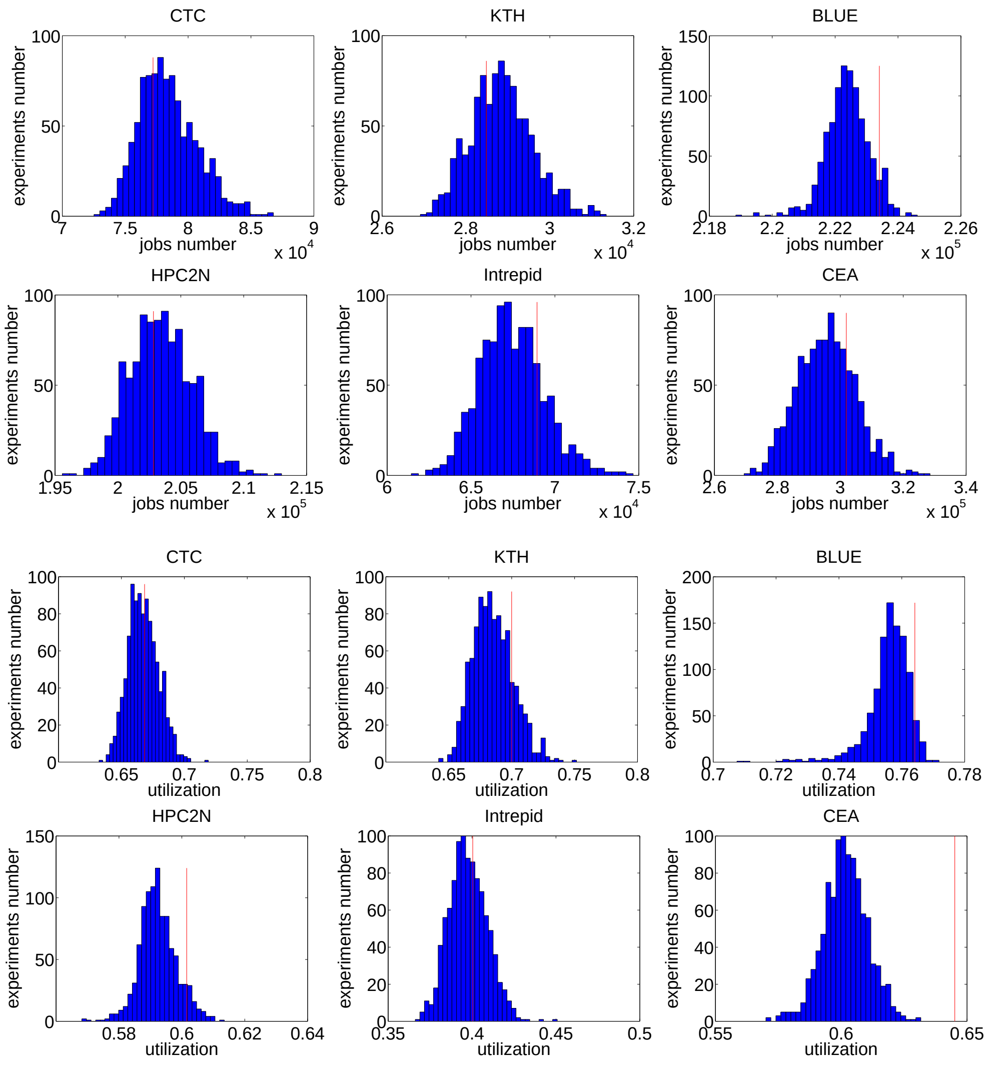

Figure 8. Histograms of the throughput and utilization in a thousand simulations with resampled workloads compared to using the original workload (vertical red line).

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

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

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

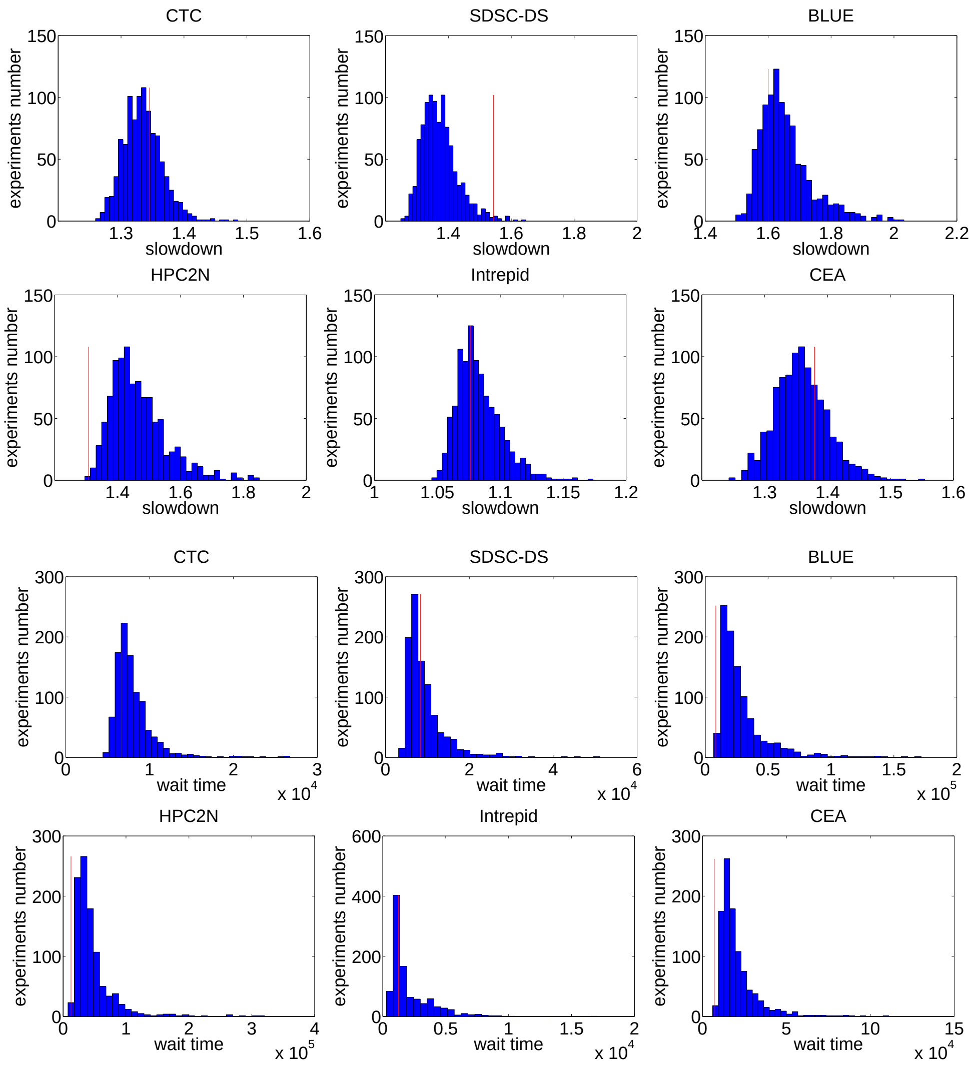

Figure 9. Histograms of the average slowdown and waiting time in a thousand simulations of EASY on resampled workloads, compared to a simulation using the original logs (vertical red line).

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

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

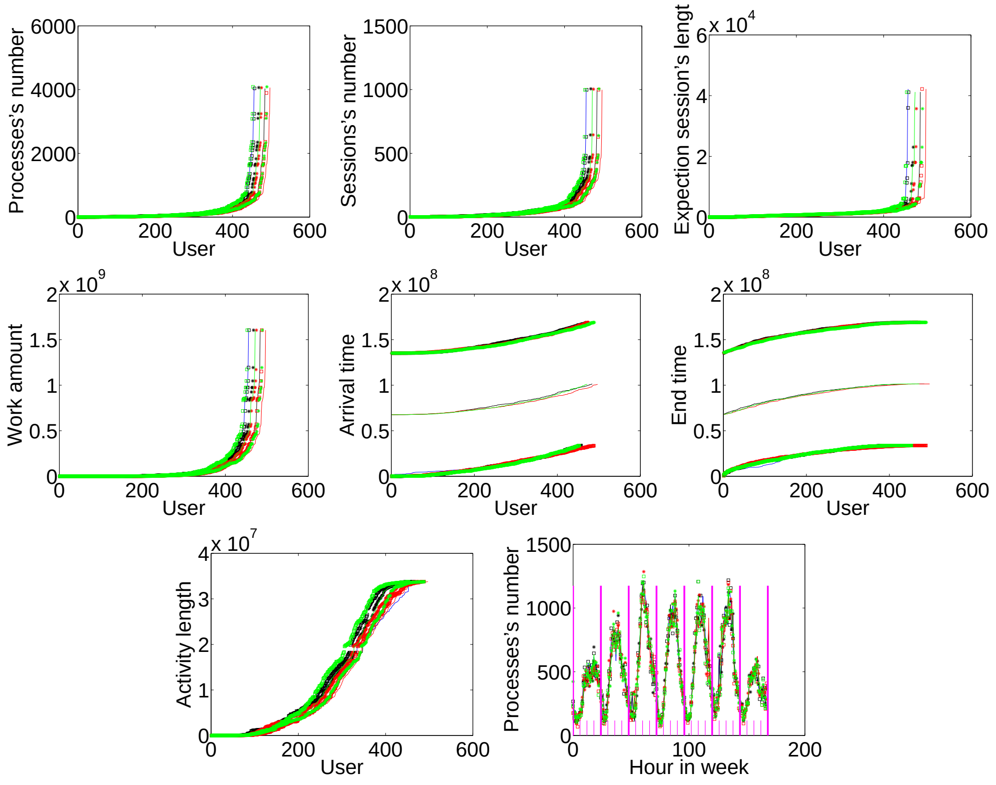

Figure 10. Comparison between the original SDSC-DS log to the first, third, and fifth parts of an extended resampled log that is 5 times longer.

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

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

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

6.2. Extending a Trace

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

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

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

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

6.3. Changing the Load

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

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

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

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

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

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

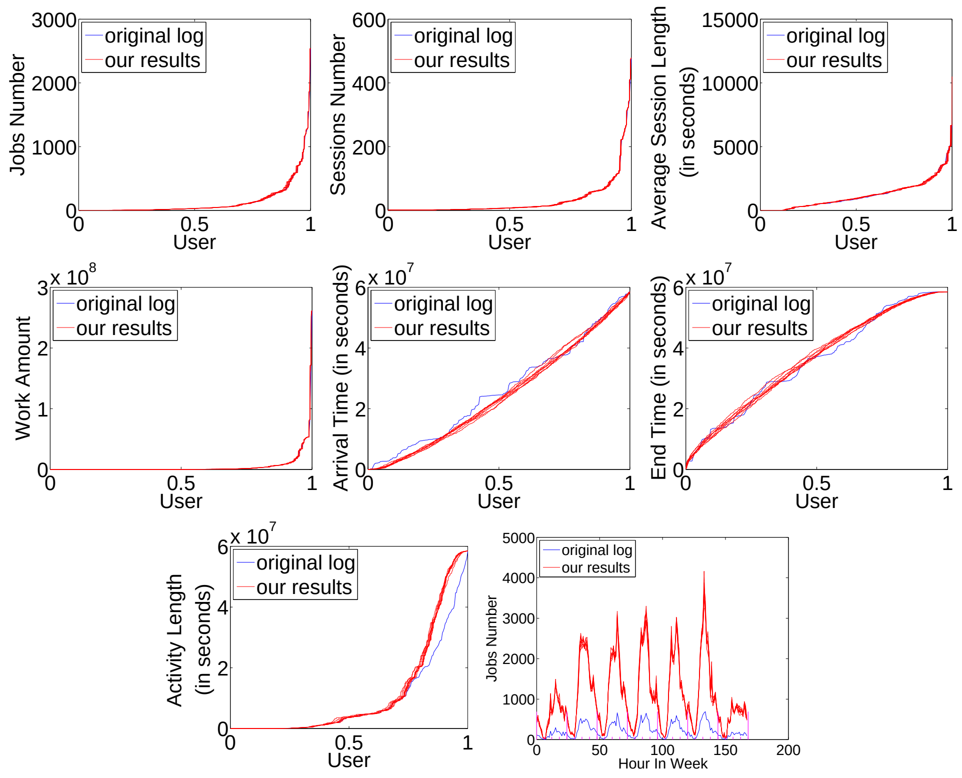

Figure 11. Comparison between the original OSC-cluster log to resampled workloads where the load is increased by a factor of 6.

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

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

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

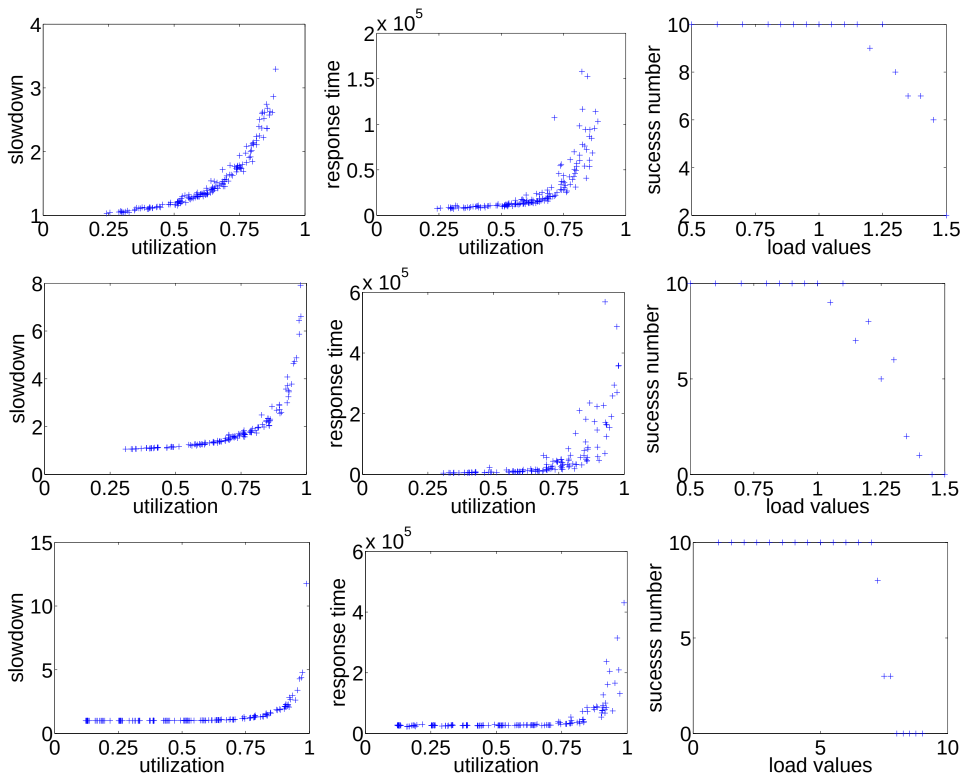

Figure 12. The performance of EASY under different load conditions for different logs: SDSC-DS, BLUE, and OSC-cluster respectively from top to bottom. Recall that the original OSC cluster log has very low load, so the load had to multiplied by higher factors to reach the range of interest.

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

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

6.4. Over-Sampling Rare Behaviors

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

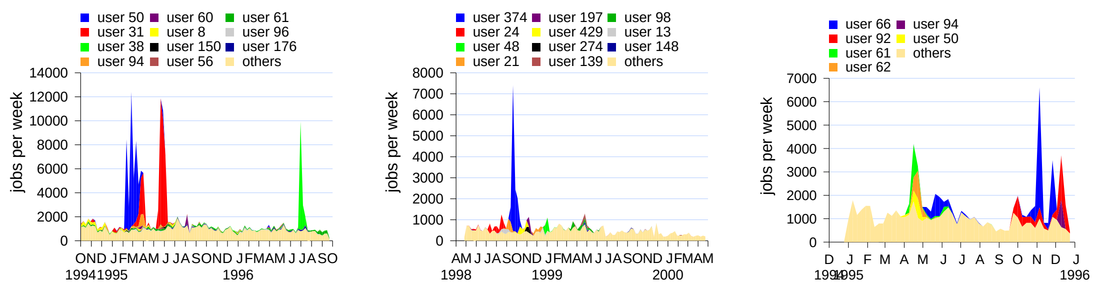

Figure 13. Flurries (flash crowds) in the original version of three logs: LANL-CM5, SDSC-SP2, and SDSC-Par95.

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

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

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

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

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

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

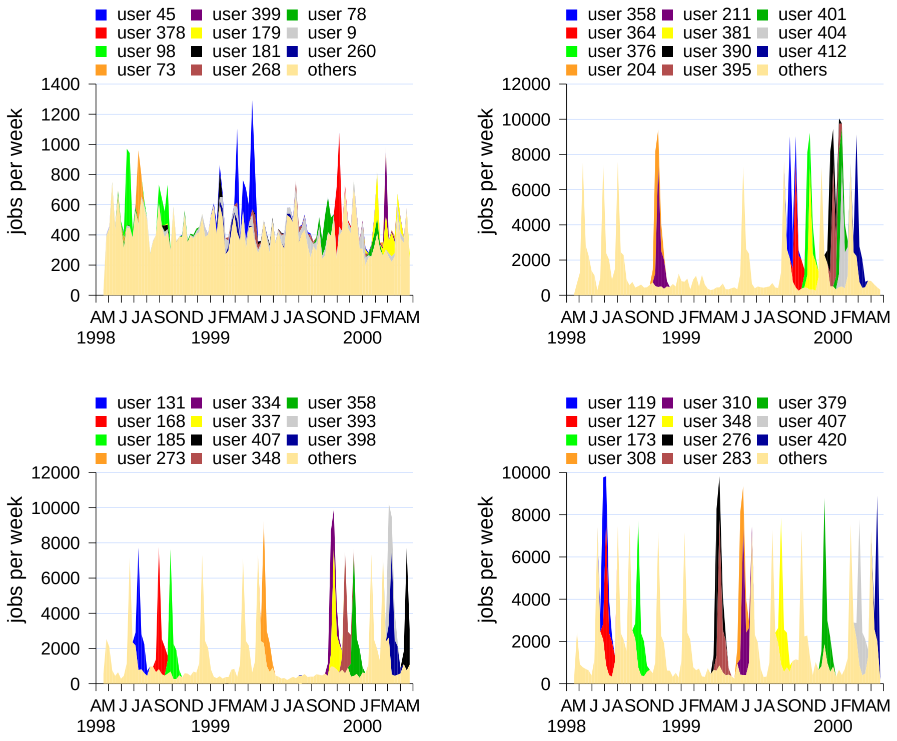

Figure 14. Adding flurries to the base workload of the SDSC-SP2 log. Note the much smaller scale in the first panel, where there are no flurries. In the others the values of RB_NPW are $\frac{1}{8}$, $\frac{1}{6}$, and $\frac{1}{4}$. In each case the 11 most active users are marked.

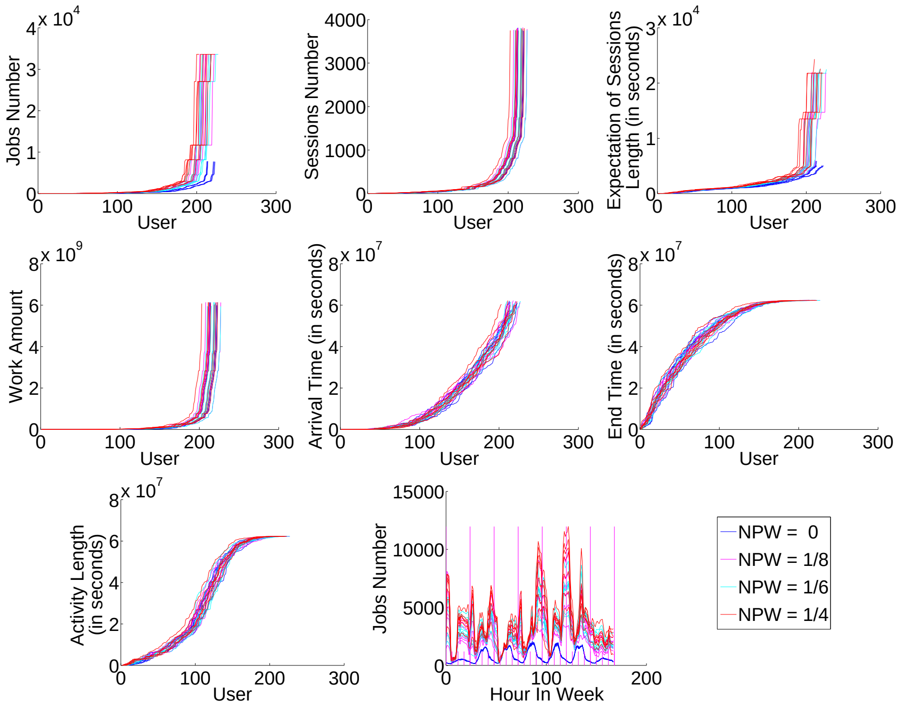

Figure 15. Comparison between the workloads created by running the simulation with different values of RB_NPW on the LANL-CM5 log.

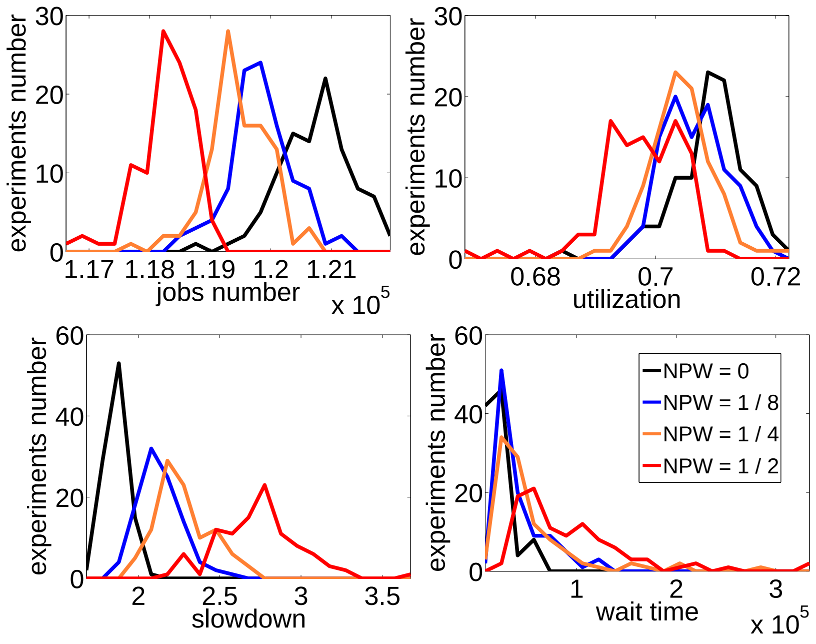

Figure 16. Distributions of number of non-flurry jobs, utilization, slowdown, and wait-time for 100 repetitions of simulating the performance of EASY on the LANL-CM5 workload with different frequencies of flurries.

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

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

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

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

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

6.5. Additional Applications

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

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

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

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

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

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

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

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

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

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

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

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

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

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

7. CONCLUSIONS

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

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

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

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

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

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

Acknowledgments

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

REFERENCES

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

20. D. G. Feitelson, D. Tsafrir, and D. Krakov, *Experience with the Parallel Workloads Archive*. Tech. Rep. 2012-6, Hebrew University, Apr 2012.
21. S. D. Gribble, G. S. Manku, D. Roselli, E. A. Brewer, T. J. Gibson, and E. L. Miller, "*Self-similarity in file systems*". In *SIGMETRICS Conf. Measurement & Modeling of Comput. Syst.*, pp. 141–150, Jun 1998, DOI: 10.1145/277851.277894.
22. F. Hernández-Campos, K. Jeffay, and F. D. Smith, "*Modeling and generating TCP application workloads*". In 4th *Broadband Comm., Netw. & Syst.*, pp. 280–289, Sep 2007, DOI:10.1109/BROADNETS.2007.4550436.
23. J. Jann, P. Pattnaik, H. Franke, F. Wang, J. Skovira, and J. Riodan, "*Modeling of workload in MPPs*". In *Job Scheduling Strategies for Parallel Processing*, pp. 95–116, Springer-Verlag, 1997, DOI:10.1007/3-540-63574-2_18. Lect. Notes Comput. Sci. vol. 1291.
24. P. Kamath, K.-c. Lan, J. Heidemann, J. Bannister, and J. Touch, "*Generation of high bandwidth network traffic traces*". In 10th *Modeling, Anal. & Simulation of Comput. & Telecomm. Syst.*, pp. 401–412, Oct 2002, DOI: 10.1109/MASCOT.2002.1167101.
25. L. Kovar, M. Gleicher, and F. Pighin, "*Motion graphs*". *ACM Trans. Graph.* **21(3)**, pp. 473–482, Jul 2002, DOI: 10.1145/566570.566605.
26. D. Krishnamurthy, J. A. Rolia, and S. Majumdar, "*A synthetic workload generation technique for stress testing session-based systems*". *IEEE Trans. Softw. Eng.* **32(11)**, pp. 868–882, Nov 2006, DOI: 10.1109/TSE.2006.106.
27. V. Kwatra, A. Schödl, I. Essa, G. Turk, and A. Bobick, "*Graphcut textures: Image and video synthesis using graph cuts*". *ACM Trans. Graph.* **22(3)**, pp. 277–286, Jul 2003, DOI:10.1145/882262.882264.
28. W. E. Leland, M. S. Taqqu, W. Willinger, and D. V. Wilson, "*On the self-similar nature of Ethernet traffic*". *IEEE/ACM Trans. Networking* **2(1)**, pp. 1–15, Feb 1994, DOI:10.1109/90.282603.
29. D. Lifka, "*The ANL/IBM SP scheduling system*". In *Job Scheduling Strategies for Parallel Processing*, pp. 295–303, Springer-Verlag, 1995, DOI:10.1007/3-540-60153-8_35. Lect. Notes Comput. Sci. vol. 949.
30. V. Lo, J. Mache, and K. Windisch, "*A comparative study of real workload traces and synthetic workload models for parallel job scheduling*". In *Job Scheduling Strategies for Parallel Processing*, pp. 25–46, Springer-Verlag, 1998, DOI:10.1007/BFb0053979. Lect. Notes Comput. Sci. vol. 1459.
31. U. Lublin and D. G. Feitelson, "*The workload on parallel supercomputers: Modeling the characteristics of rigid jobs*". *J. Parallel & Distributed Comput.* **63(11)**, pp. 1105–1122, Nov 2003, DOI:10.1016/S0743-7315(03)00108-4.
32. B. B. Mandelbrot, *The Fractal Geometry of Nature*. W. H. Freeman and Co., 1982.
33. N. Mi, G. Casale, L. Cherkasova, and E. Smirni, "*Injecting realistic burstiness to a traditional client-server benchmark*". In 6th *Intl. Conf. Autonomic Comput.*, pp. 149–158, Jun 2009, DOI: 10.1145/1555228.1555267.
34. "*Parallel workloads archive*". URL http://www.cs.huji.ac.il/labs/parallel/workload/.
35. K. Pawlikowski, "*Steady-state simulation of queueing processes: A survey of problems and solutions*". *ACM Comput. Surv.* **22(2)**, pp. 123–170, Jun 1990, DOI:10.1145/78919.78921.
36. V. Paxson and S. Floyd, "*Wide-area traffic: The failure of Poisson modeling*". *IEEE/ACM Trans. Networking* **3(3)**, pp. 226–244, Jun 1995, DOI:10.1109/90.392383.
37. M. Schroeder, *Fractals, chaos, Power Laws*. W. H. Freeman and Co., 1991.
38. E. Shmueli and D. G. Feitelson, "*Using site-level modeling to evaluate the performance of parallel system schedulers*". In 14th *Modeling, Anal. & Simulation of Comput. & Telecomm. Syst.*, pp. 167–176, Sep 2006, DOI: 10.1109/MASCOTS.2006.50.
39. J. Sommers and P. Barford, "*Self-configuring network traffic generation*". In 4th *Internet Measurement Conf.*, pp. 68–81, Oct 2004, DOI:10.1145/1028788.1028798.
40. D. Talby and D. G. Feitelson, "*Improving and stabilizing parallel computer performance using adaptive backfilling*". In 19th *Intl. Parallel & Distributed Processing Symp.*, Apr 2005, DOI:10.1109/IPDPS.2005.252.
41. D. Tsafrir and D. G. Feitelson, "*Instability in parallel job scheduling simulation: The role of workload flurries*". In 20th *Intl. Parallel & Distributed Processing Symp.*, Apr 2006, DOI:10.1109/IPDPS.2006.1639311.
42. D. Tsafrir, K. Ouaknine, and D. G. Feitelson, "*Reducing performance evaluation sensitivity and variability by input shaking*". In 15th *Modeling, Anal. & Simulation of Comput. & Telecomm. Syst.*, pp. 231–237, Oct 2007, DOI: 10.1109/MASCOTS.2007.58.
43. K. V. Vishwanath and A. Vahdat, "*Swing: Realistic and responsive network traffic generation*". *IEEE/ACM Trans. Networking* **17(3)**, pp. 712–725, Jun 2009, DOI:10.1109/TNET.2009.2020830.
44. M. C. Weigle, P. Adurthi, F. Hernández-Campos, K. Jeffay, and F. D. Smith, "*Tmix: A tool for generating realistic TCP application workloads in ns-2*". *Comput. Commun. Rev.* **36(3)**, pp. 67–76, Jul 2006, DOI: 10.1145/1140086.1140094.
45. Y. Wexler, E. Schechtman, and M. Irani, "*Space-time video completion*". In *Conf. Comput. Vision & Pattern Recog.*, vol. 1, pp. 120–127, Jun 2004, DOI:10.1109/CVPR.2004.1315022.
46. N. Zakay and D. G. Feitelson, "*On identifying user session boundaries in parallel workload logs*". In *Job Scheduling Strategies for Parallel Processing*, W. Cirne et al. (eds.), pp. 216–234, Springer-Verlag, 2012, DOI:10.1007/978-3-642-35867-8_12. Lect. Notes Comput. Sci. vol. 7698.
47. N. Zakay and D. G. Feitelson, "*Workload resampling for performance evaluation of parallel job schedulers*". In 4th *Intl. Conf. Performance Engineering*, pp. 149–159, Apr 2013, DOI:10.1145/2479871.2479893.
48. J. Zilber, O. Amit, and D. Talby, "*What is worth learning from parallel workloads? a user and session based analysis*". In 19th *Intl. Conf. Supercomputing*, pp. 377–386, Jun 2005, DOI:10.1145/1088149.1088200.