Reducing Performance Evaluation Sensitivity and Variability by Input Shaking

Dan Tsafrir
Dept. Computer Science
The Hebrew University, Jerusalem, Israel
and IBM T. J. Watson Research Center
Yorktown Heights, NY 10598
Email: dants@cs.huji.ac.il

Keren Ouaknine
Dept. Computer Science
The Hebrew University
Jerusalem 91904, Israel
Email: ouaknine@cs.huji.ac.il

Dror G. Feitelson
Dept. Computer Science
The Hebrew University
Jerusalem 91904, Israel
Email: feit@cs.huji.ac.il

Abstract—Simulations sometimes lead to observed sensitivity to configuration parameters as well as inconsistent performance results. The question is then what is the true effect and what is a coincidental artifact of the evaluation. The shaking methodology answers this by executing multiple simulations under small perturbations to the input workload, and calculating the average performance result; if the effect persists we can be more confident that it is real, whereas if it disappears it was an artifact. We present several examples where the sensitivity that appears in results based on a single evaluation is eliminated or considerably reduced by the shaking methodology. While our examples come from evaluations of scheduling algorithms for supercomputers, we believe the method has wider applicability.

Index Terms—Performance evaluation, Simulation, Workload trace, Workload perturbations, Variability, Instability, Sensitivity

I. INTRODUCTION

Performance evaluations are routinely done by simulating how a system would work with a given workload. In the interest of obtaining reliable and representative results, the workload is often taken from a trace of events that were recorded on a real system in production use. The evaluation results may then be influenced by unique interactions between the system and the specific trace used.

As an example, consider the following simulation of a backfilling parallel job scheduler. Backfilling amounts to using small jobs from the queue of waiting jobs to fill in holes in the schedule. This requires estimates of job runtimes to be available. The scheduler in question obtained these estimates by averaging the runtime of the last k jobs submitted by the same user. Simulations using a specific workload trace then showed that changing the number of jobs k from 12 to 13 led to a major reduction of 29% in the average bounded slowdown measured for all jobs in the trace. If this is a real effect, and 13 is indeed the magic number to use, this would be a major breakthrough. But if it is an artifact of unique conditions that occurred in this specific simulation, it is a distraction that should be ignored. In fact it turns out to be an artifact; this and other examples are detailed in Section VI.

Our solution is to "shake" the input workload: perform multiple runs with small random variations in the workload, and calculate the average of the results. For example, we can cause jobs to arrive a few minutes earlier or later than they do in the original workload trace. If an effect is real, it should be robust to such small variations. But if the effect is the result of a unique coincidence, there is a good chance that the shaking will change the conditions enough to eliminate the spurious effect. The shaking is the mechanism that creates multiple workloads when initially we have only one, thus enabling us to perform multiple measurements to characterize the distribution of results, and to calculate an average and confidence interval. Details of the methodology are described in Sections IV and V.

Section II elaborates on our motivation, and III discusses related work.

II. SENSITIVITY OF PERFORMANCE EVALUATION

All our examples are from simulations of the EASY scheduler [1], which is currently the most common method for parallel job scheduling. We found several cases of noisy or inconsistent performance results, where very small modifications to the workload or to a system parameter — that were expected to have little or no effect on the evaluation results — actually caused a large effect.

For example, in the SDSC workload (see Table I), job 64,241 was estimated to run for 18 hours and ran for 18 hours and 30 seconds. Running another simulation in which the extra 30 seconds were truncated, which represents a modification of 0.046%, resulted in a change of 8% in the average bounded slowdown of all the jobs in the trace [2]. Moreover, other minor modifications caused different changes (e.g. adding 10 seconds resulted in a change of 3.5%). This is obviously an undesirable sensitivity.

Another example appears in Fig. 1, which shows the average bounded slowdown of jobs from the CTC workload as a function of the load on the system (higher loads are achieved by consistently reducing the interarrival times between jobs). As in queueing theory, we would expect the curve to be smooth and continuous, growing asymptotically as we approach saturation. But the actual results show strong irregularities, where small changes to the load lead to jumps in the average bounded slowdown. Moreover, applying small random perturbations to the workload (the arrival times of some jobs are modified by up
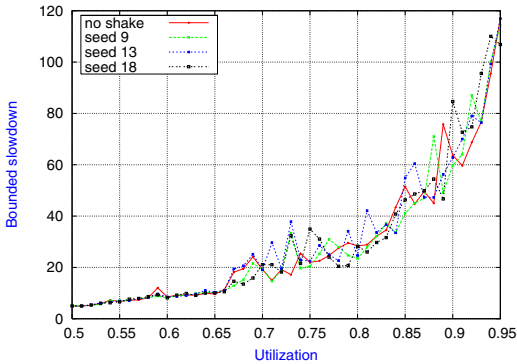
Fig. 1.   Bounded slowdown as a function of utilization, CTC log.

to 5 minutes) changes these irregularities, which now appear at other loads. Thus the irregularities do not have a real significance, and are just artifacts of unique interactions between the workload and the system. If we were to compare schedulers based on these evaluations, we could have concluded wrongly that one scheduler is better than the other, while in actuality, each single evaluation is simply not representative.

A well-known problem with using workload traces is that a trace provides only a single data point. It is therefore impossible to say with certainty whether a result is real or bogus. With workload models, in contradistinction, one can generate multiple workloads that are statistically identical and observe the effect of variability on the evaluation results. Shaking attempts to achieve the same utility with a given workload trace, by generating multiple closely related workload variants.

It should be emphasized that the purpose of shaking is not to solve irregularities but to circumvent them. Each case has its own reasons for why irregularities arise. These can be very hard to track down [2], and are usually a combination of unfavorable circumstances. Our purpose is not to resolve them, but to prevent their influence on the performance evaluation.

III. Related Work

Instability in performance evaluations has not been studied in depth. Our work is a followup to that of Tsafrir and Feitelson [2], who traced an instability to workload flurries.

In another study on job scheduling, England et al. [3] explain that performance evaluations are affected by the presence of large deviations and that robust systems should withstand these disturbances and maintain stable performance results. They present a methodology to measure the robustness of a system by determining the degradation in performance with the Kolmogorov-Smirnov test [4] to quantify the maximal difference between the CDFs with and without perturbations added to the system.

In a paper by Lawson and Smirni [5], the system adapted its backfilling parameters to the workload fluctuations. Some of the presented results seem to exhibit large localized fluctuations, e.g. the measured slowdown for successive weeks on four workloads (Section 3, Figure 4 of their paper). Thus it

seems that multiple-queue backfilling may also be sensitive to unique circumstances in the simulation.

Alameldeen and Wood presented the variability of results of architectural simulations of multi-threaded workloads in [6], and presented a methodology for reducing the probability of reaching incorrect conclusions. The methodology is based on a technique of injecting random perturbations to create a space of runs and using the mean as the performance result. This is very similar to our shaking methodology. They also presented the WCR (Wrong Conclusion Ratio) metric to quantify the risk of reaching an incorrect conclusion in the comparison of two different system configurations.

IV. The Shaking Methodology

Shaking consists of performing small perturbations on a specific parameter of the workload, executing simulations under these perturbations, and finally calculating the average of the performance results. In other words, the same simulation is run repeatedly, but before each run, one characteristic is modified by a small amount. The results of the multiple repetitions are then summarized to produce the final outcome. Fig. 2 shows a flow diagram of the shaking steps.
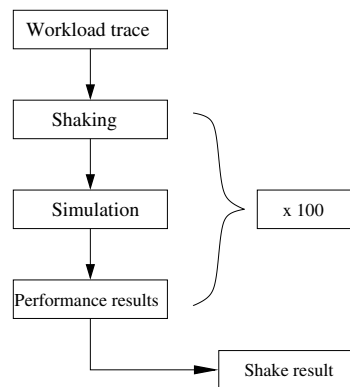

Fig. 2.   Flow diagram of the shaking methodology.

The cost of applying shaking is a multiplication of the runtime: if simulations are repeated 100 times, this will take approximately 100 times more than a single simulation. While this is a significant increase, it can be done as witnessed by multiple examples shown in Section VI. Of course if fewer repetitions are used the cost is reduced.

The main methodological questions are what parameters to shake, and to what degree. The main consideration is how much one can modify a workload without changing its nature. The *level of shaking* is defined by the amount of perturbation, the attribute chosen, and the percentage of jobs shaken.

In the context of parallel job scheduling, workloads include many thousands of jobs, each having (1) an arrival time, (2) a size (number of processors used), (3) a runtime, and (4) an estimate of the runtime provided by the user. These attributes have different properties in terms of their distributions, e.g. whether they are continuous or discrete. They also have different effects on how the job may interact with the scheduler. All these are considerations regarding how to shake them.

For example, the arrival time is a function of when users arrive at work, perform their tasks, take breaks, etc. Therefore, modifying the arrival time by a few minutes shouldn't affect neither the scheduler's strategies nor the overall results of the performance evaluation. But user estimates are often short and come from a restricted repertoire of different values [7], so changing them by a few minutes may completely change the behavior of the scheduler.

Once we select the workload attribute to shake, the degree of perturbation is applied using the following formula:

$$new\_val = orig\_val \pm pert \times rand \qquad (1)$$

This increases or decreases the original value with the same probability. The maximal modification is given by the perturbation value. The random factor assures a uniform distribution.

The *degree* of shaking denotes the maximal magnitude of the perturbation performed on the jobs. The shaking becomes stronger as the degree increases. This can be absolute or relative. *Absolute shaking* is performed as is using the degree that was given as a parameter to the shaking procedure. *Relative shaking*, on the other hand, also takes into account the relationship with the original value:

$$pert = \min\{pert,\ rel\_pct \times orig\_val\} \qquad (2)$$

Thus relative shaking performs smaller changes than absolute shaking: we use the minimum between the degree of shaking given and the relative percentage of the original value to shake. For example, if shaking the size with a perturbation of 5 processors and a relative percentage of 10% is applied to a job using 21 processors, then the applied perturbation is $pert = \min\{5, 10\% \times 21\} = 2.1$, which is rounded to 2 (the size is an integral number of processors). Here, the relative factor limited the perturbation.

Turning next to the workload attributes, we note that shaking the *arrival time* directly may lead to considerable modifications to the structure of the workload in terms of burstiness and the sequence of jobs that are submitted. We therefore prefer to shake the *interarrival time*, i.e. the time between two consecutive jobs. Thus, a job's arrival time is shaken relative to the time elapsed since the previous job's arrival time. Given an arrival time t_i, the interarrival time is $a_i = t_i - t_{i-1}$. This is modified to $a'_i = a_i \pm pert$, and the new arrival time is set to $t'_i = t_{i-1} + a'_i$.

Note that the modified interarrival is applied the original previous arrival, so as to avoid accumulation of perturbations. As a result some localized jobs can be interchanged. For example, in Fig. 3 the modification to the relatively long interarrival between job 1 and job 2 is enough to change the order of job 2 and job 3.

Shaking the *runtime* changes the job duration, thus is considered a stronger perturbation than interarrival. It may also change the predictability of the workload, as real workloads often include repeated executions of the same job [8]. Shaking the user estimate or job size are considered even stronger, and therefore less desirable. Modifying the *estimated runtime* does not change the actual runtime of the job. However, given that
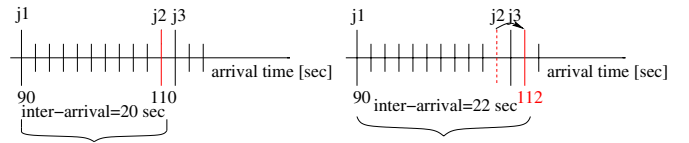

Fig. 3.   Example of job interchange as a result of shaking.

estimates tend to be modal (e.g. 5 minutes, 15 minutes, or 2 hours [7]), shaking may actually give the scheduler more information by making the jobs distinct from each other. This risks affecting the behavior of the scheduler.

The size attribute is obviously discrete, and shaking it may affect the workload considerably. For example, given a job of one processor shaking by ± 1 will either cancel or double the job. In most workloads size requests are predominantly for powers of two, and shaking them will create new values which may influence the fragmentation. The same applies in workloads where allocations are node-based, e.g. multiples of 8 processors. For these reasons, shaking size is considered a strong perturbation, leaving the interarrival time as the preferred attribute to shake.

The *percentage* of jobs denotes the fraction of jobs that are shaken. For example, setting the percentage to 50% means that half of the jobs in the workload will be shaken. Shaking a lower percentage of jobs stays closer to the original workload, but we need to shake enough to get an effect.

V. Variation of Parameters

Shaking has two important numerical parameters: the degree of shaking, and the percentage of jobs to shake. Setting these parameters to different values has an effect on the performance result. The experiments in this section contribute to our understanding of the impact of shaking, and to our ability to set ranges of reasonable shaking values.

Fig. 4 shows the pdf of the bounded slowdown metric when shaking the interarrival attribute by different degrees for a percentage of 1% and 10%. The X axis units are bounded slowdown relative to the original result (with no shaking at all), so the original result always appears at 100. Each curve represents 100 runs of the same configuration but with different random seeds. The figure shows that as the perturbation degree increases, the distributions of results depart from the original. In general, simulations configured with small degrees of perturbation did not spread much, and thus were not effective for avoiding irregularities.

The percentage has a similar effect: when only 1% of the jobs are shaken, the results remained relatively close to the original, and the span of results was in the range of 98–105. When 10% of the jobs were shaken, the range grew to 96–112: the percentage of jobs shaken increased, and so, more results spread away from the original evaluation. With 100% of jobs shaken, the span did not grow, but results spread away from the original evaluation.

Similar results were obtained for other workloads and attributes; overall, millions of simulations were performed to
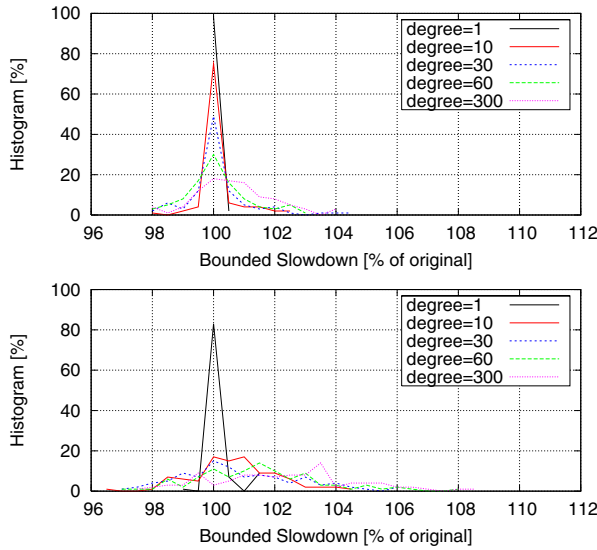
Fig. 4. distribution of bounded slowdown when shaking interarrivals of 1% (top) or 10% (bottom) of the SDSC jobs (original=93.37).

collect and compare the results. To obtain a better view of how shaking the parameters impacts the evaluation, we turn to 3D plots in which we modify both the degree and the percentage. The degree of shaking for the interarrival time attribute was set to 1, 10, 30, 60, and 300 seconds. The percentage of jobs was varied by 1, 3, 10, 30, and 100 percent. The measures adopted for evaluating the effect of shaking are presented in Fig. 5. We used three main measures: (1) the span of results, (2) the distance between the original evaluation and the shaking result, and (3) the concentration around the original evaluation.

1) The Span: The span is the interval which includes all the results. However, we use a more restrictive definition of the interval from the 5th percentile to the 95th percentile, in order to reduce the sensitivity to outliers. This shows the impact of shaking on the dispersal of the results. The main importance of the span is that it characterizes the degree of confidence we
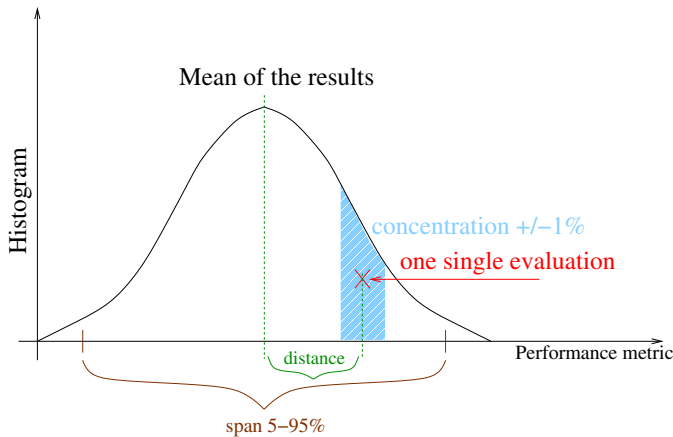

Fig. 5. Metrics used to quantify the effect of shaking.

can have in the results. If the span is relatively small despite aggressive shaking, we know that the results are robust. In most cases, stronger shaking increases the span. To avoid the issue of units, we express the span as a deviation from the original value in percents, as in Fig. 4.
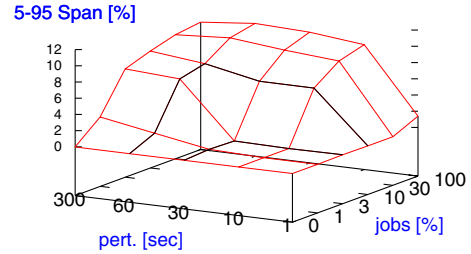

Fig. 6. Span of bounded slowdown when shaking interarrivals, BLUE log.

Fig. 6 shows the effect of shaking on the span. We see that a minimal percentage of 10% and a degree of 10–30 seconds are required to get a real impact. Less then that is not effective, and more perturbation doesn't make a difference. Similar measurements for the runtime and estimate attributes required a minimal percentage of 30% and a degree of 60 seconds to have an impact on the performance results. By comparing the results on four different workloads, with different degrees and percentages, we see that as we perform stronger shaking, the span of results increases. Importantly, one single result can be any of the points in the span, and so we need to use the average of all these points to get a representative result.
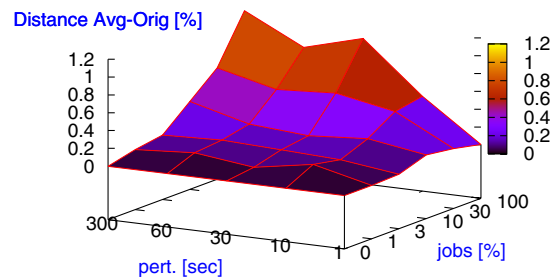

Fig. 7. Distance of average wait time from original, shaking interarrivals on the BLUE log.

2) The Distance: Another way to quantify the influence of shaking is to measure the distance between the original evaluation and the average shaken results (again, the units are percents of the original value). A small distance shows that the single evaluation was close to the shaken result. Fig. 7 shows that as we perform stronger shaking, the distance tends to increase. Thus strong perturbations may change the workload significantly and remove similarities with the

original evaluation. To reach reliable conclusions, one can use the results based on a combination of the span and the distance.

3) The Concentration: The concentration measures the deviation of the shaken results from the original evaluation. The purpose is to observe whether the shaken results were within $\pm 1\%$ of the original evaluation. If the results are concentrated, it means that the impact of the shaking results was minor. However, shaking might still be effective since one can circumvent the instability with very minor changes, as will be illustrated in the next section.
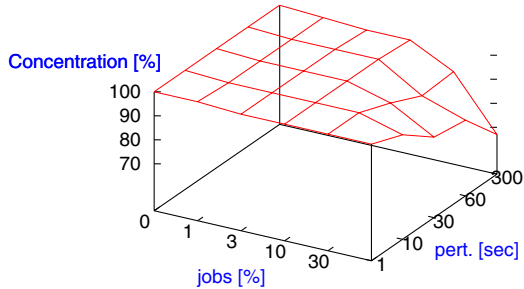

Fig. 8. Concentration of the wait time metric when shaking interarrivals, SDSC log.
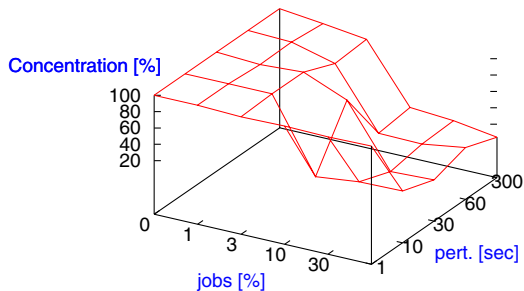

Fig. 9. Concentration of bounded slowdown when shaking interarrivals, BLUE log.

Fig. 8 shows the concentration of results when shaking the interarrival attribute, based on the SDSC log. The figure shows that, even with the most aggressive shaking, 65% of the results were in the interval of 99–101%. Thus, most results remain close to the original evaluation. In Fig. 9, the concentration decreases to 40%. To resume, strong shaking creates new evaluation results which are included in the shaking average, and these are responsible for the change of the outcome. If the perturbations are too small the overall effect is reduced and the original result prevails.

The purpose of shaking is to yield reliable performance evaluations without applying a major change on the workload. Considering that and the above results, we can suggest a default minimal configuration for shaking in the context of parallel job scheduling: a minimal percentage of 10% and

a degree of at least 60 seconds on the interarrival or the runtime attribute. Specific studies may however require further adjustments depending on the case parameters to which it is applied, e.g., the log, the performance metric, etc.

Note that these suggested values are very modest: 90% of the trace data remains without change, and for those jobs that do change, the modification is small — up to one minute, when many interarrival times and runtimes are originally much longer. We claim that even more aggressive shaking, e.g. shaking all the jobs by up to 5 or even 15 minutes, is also still reasonable. However, this is not a formally proven fact, as we cannot know what is the "correct" result of an evaluation, and when the workload is changed too much. Rather, the claim is based on an intuitive understanding of the domain: a parallel supercomputer typically serves several hundred jobs a day, these jobs are typically submitted by human users, and it is therefore reasonable to claim that we want the performance to be robust to small changes like a specific user submitting a specific job a couple of minutes earlier or later. We would not make such a claim for much larger values: if many users submit their jobs several hours earlier or later, this would be a change that we would feel uncomfortable with. Indeed, we tried simulations with such large values, but this led to questionable results due to bursts of activity that existed in the original trace but were spread out due to the aggressive shaking. In other domains, the values would be different. For example in networking, shaking the arrival times of packets may be expected to be on the scale of milliseconds rather than minutes.

VI. Shaking Applied to Sensitive Test Cases

In this section, we discuss several examples of performance evaluations, where similar simulations lead to very different results. All the examples relate to parallel job scheduling, using logs from the Parallel Workloads Archive (Table I), which represent real parallel production environments. Specifically, we present five such evaluations, and study the impact of shaking on each.

TABLE I
LOGS USED IN OUR EXAMPLES, AVAILABLE FROM [9].

source	duration	jobs	file
CTC SP2	1996-7	79,302	CTC-SP2-1996-2
SDSC SP2	1998-0	73,496	SDSC-SP2-1998-2.1-cln
Blue Horizon	2000-3	250,440	SDSC-BLUE-2000-2.1-cln

A. The Butterfly Effect

The *butterfly effect* refers to the sensitivity of the outcome on the initial conditions, especially where tiny variations in a specific attribute produce large variations in the performance results. An example is described in [2], where job 64,241 of the SDSC workload was estimated to last 18 hours and turned out to run for 18 hours and 30 seconds. To correct this, the job was shortened by 30 seconds in the simulation. This modification truncated a single job by 0.046%. Surprisingly,

	Original	shake 1mn	shake 5mn	shake 15mn
w/o trunc.	88.15	86.82	86.99	87.04
with trunc.	81.38	86.62	87.01	86.91
abs diff	8.31%	0.23%	0.02%	0.14%

it resulted in a change of 8.3% in the average bounded slowdown of *all* the jobs in the trace (79,302 jobs), which is an undesirable sensitivity to the initial conditions.

Table II compares the original evaluation with the shaking result of the performance evaluation with and without the 30 second truncation. The input to these two simulations is nearly identical: all job arrivals and attributes are the same, except for a 0.046% difference in the runtime of one job out of 79,302. The first column of the table shows the 8.3% difference of the original evaluation (no shaking). The other columns show results for various relative shaking degrees on the interarrival time attribute. Even a small degree of one minute on all jobs removes the instability yielding a 0.2% difference with and without the truncation.

B. Simultaneous Job Arrivals

When several jobs in the trace have the same arrival time, it is natural to schedule them in the order that they appear. But a simulator that inserts arrival events into the first appropriate location in the event queue will end up reversing this order. This is an implementation detail that should not have a large impact on results. But an evaluation on the SDSC log with and without the reversal of simultaneous jobs led to a surprisingly high 6.7% performance difference, which casts a shadow on the whole simulation methodology. Running the same simulation using the shaking methodology with a degree of 5 minutes (thereby creating different sets of jobs that happen to have the same arrival times) reduced the difference to 0.47%.

C. Load Variations

The load on the system can be varied by modifying the interarrival times. Effectively, we linearly increase the arrival rate of jobs in the system to increase the load. Note that with this modification we preserve the statistical characteristics of the arrival pattern in the original trace, except that the same jobs now arrive faster. As shown in Fig. 1, this leads to surprising irregularities in the results.

To investigate this more closely, we performed simulations of all loads in the range [0.5..0.95] with a resolution of 0.001, and compared the original evaluation with the shaken results. Fig. 10 shows the bounded slowdown as a function of the load on the CTC workload. The curve of the single evaluation is very noisy, and even minute differences of 0.001 in the load can lead to large changes in the results. The curves of the shaking results are much smoother and show the expected trend. Thus all the variations in the original evaluations are artifacts of using this specific workload trace.
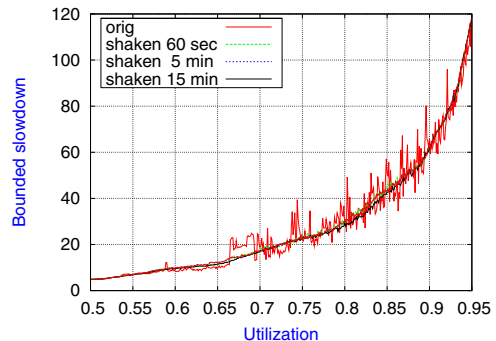

Fig. 10. Bounded slowdown as a function of load variation, with and without shaking, using the absolute version.

D. The Prediction Window

The first sensitivity example in Section I concerned the prediction of job runtimes by averaging the runtimes of the last k jobs by the same user [10]. k is called the *predication window*. Changing its value from 12 to 13 when simulating the SDSC workload led to a high peak at size 12, and an abnormal low at size 13. These variations created a major difference of 29% in the bounded slowdown.
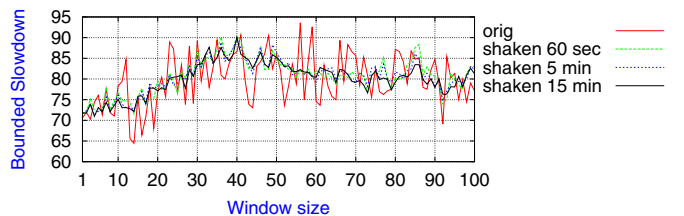

Fig. 11. Reducing instability by relative shaking for the prediction window size example.

Fig. 11 extends these results by showing the bounded slowdown for all values of k from 1 to 100, and how it is effected by relative shaking. The shaking curves are much smoother then the single evaluation curve, and in particular, there is no real threshold between 12 and 13. The reduced variability also clearly exposes the trend of degraded performance as the window size grows to 40.

E. The Estimation Factor

A simple way to study the effect of inaccurate runtime estimates is to assume that a job's estimate is uniformly distributed within $[R, (f+1)R]$, where R is the job's real runtime, and f is a "badness" factor (so called because estimates become increasingly inaccurate as f grows, while $f = 0$ implies that the estimates are identical to the runtimes) [11]. A very surprising result was that, in terms of performance, inaccurate estimates seemed preferable to accurate ones. However, a more thorough investigation shows that the result is very sensitive to the precise value of f used [12]. In Fig. 12 we see a comparison between the shaken result and the original evaluation. Without shaking, the performance is noisy and
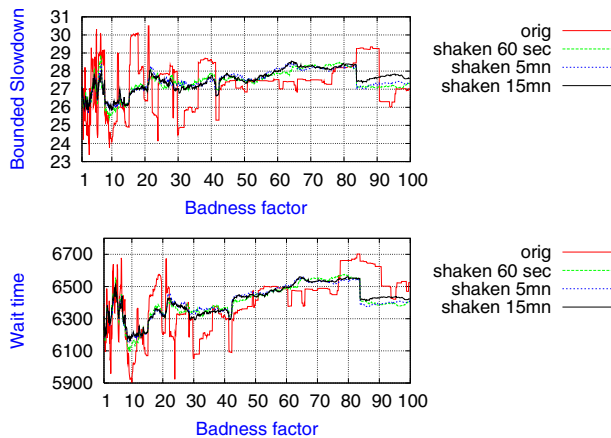
Fig. 12. Bounded slowdown (top) and wait time (bottom) as a function of the badness factor. SDSC log.

unstable. Relative shaking has a smoothing effect for both the bounded slowdown and the wait time metrics.

VII. CONCLUSIONS

Our first contribution in this paper is to draw attention to the possible sensitivity of simulations based on a single workload trace, where circumstances unique to the trace cause large apparent performance fluctuations as some input parameter or system configuration parameter is changed.

The second contribution is to propose a methodology that enables us to verify whether this sensitivity is a really meaningful effect, or an artifact. The methodology, called "shaking", is based on executing multiple simulation runs each with a small modification to one selected attribute of the workload. This generates multiple closely related workloads that serve as multiple equally-valid samples from the workload space. Averaging the performance results obtained with all the shaken workloads was found to be much more stable than the original simulation result.

In order to assess the effect of shaking, we applied the methodology on four workloads, four job attributes, several perturbation degrees, and also several different job percentages. It appears that the best attribute to shake is the interarrival time of a job, for at least 10% of the jobs in the workload and a minimal perturbation of 1 minute. We applied this and slightly more extreme configurations to several examples of unstable evaluations, and found that it significantly improved the quality of the performance results and allowed the true system behavior to be identified.

Our experience with multiple examples indicates that relative shaking is more robust than absolute shaking: it retains the characteristics of the original workload more closely, and therefore the results do not depend so much on the specific degree of shaking used. As a result, it is sufficient to use a relatively low degree of shaking, e.g. 1 to 5 minutes. Note that even with shaking, the simulation results are never perfectly smooth. However, it would be wrong to use more aggressive shaking to completely smooth out the results, as this risks the elimination of true effects.

The shaking methodology is easy to understand and easy to apply, and we hope it will be further used for performance evaluations of scheduling algorithms, and perhaps in other domains.

REFERENCES

[1] L. Malinowsky and P. Öster. Scheduling of a parallel workload: Implementation and use of the Argonne EASY scheduler at PDC. In *Applied Parallel Computing*, pp. 309–314. LNCS vol. 1541, Springer-Verlag, 1998.

[2] D. Tsafrir and D. G. Feitelson. Instability in parallel job scheduling simulation: The role of workload flurries. In *Intl. Parallel & Distributed Proc. Symp.*, Apr 2006.

[3] D. England, J. Weissman, and J. Sadagopan. A new metric for robustness with application to job scheduling. In 14th *High-Performance Distrib. Comput.*, July 2005.

[4] T. Gonzalez, S. Sahni, and W. R. Franta. An efficient algorithm for the Kolmogorov-Smirnov and Lilliefors Tests. *ACM Trans. Math. Softw.* 3(1):60–64, 1977.

[5] B. G. Lawson and E. Smirni. Multiple-queue backfilling scheduling with priorities and reservations for parallel systems. In *Job Scheduling Strategies for Parallel Processing*, pp. 72–87. LNCS vol. 2537, Springer Verlag, 2002.

[6] A. R. Alameldeen and D. Wood. Variability in architectural simulations of multi-threaded workloads. In 9th *High-Performance Comput. Arch.*, Feb 2003.

[7] D. Tsafrir, Y. Etsion, and D. G. Feitelson. Modeling user runtime estimates. In *Job Scheduling Strategies for Parallel Processing*, pp. 1–35. LNCS vol. 3834, Springer-Verlag, 2005.

[8] D. G. Feitelson. Locality of sampling and diversity in parallel system workloads. In 21st *Intl. Conf. Supercomputing*, Jun 2007.

[9] Parallel workloads archive. www.cs.huji.ac.il/labs/parallel/workload.

[10] D. Tsafrir, Y. Etsion, and D. G. Feitelson. Backfilling using system-generated predictions rather than user runtime estimates. *IEEE Trans. Par. Dist. Syst.* 18(6), Jun 2007.

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

[12] D. Tsafrir and D. G. Feitelson. The dynamics of backfilling: solving the mystery of why increased inaccuracy may help. In *IEEE Intl. Symp. Workload Characterization*, Oct 2006.

[13] A. Barak, A. Shiloh, and L. Amar. An organizational grid of federated MOSIX clusters. In 5th *IEEE Intl. Symp. Cluster Computing & Grid*, pp. 350–357, May 2005.