

Reducing Performance Evaluation Sensitivity and Variability by Input Shaking

A thesis submitted in partial fulfillment
of the requirements for the degree of
Masters of Science

by
Keren Ouaknine

Supervised by
Prof. Dror Feitelson

School of Engineering and Computer Science
The Hebrew University of Jerusalem
Israel

June 10, 2007

Abstract

This thesis describes a methodology for performing evaluations of scheduling algorithms for super-computers based on simulation results. Simulations sometimes lead to observed variability and sensitivity to configuration parameters. 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 the examples come from evaluations of scheduling algorithms for supercomputers, we believe the method has wider applicability.

Acknowledgments

I wish to express my gratitude to Prof. Dror Feitelson for his precious guidance and instruction. I would like to thank Dr. Dan Tsafrir for his supporting guidance and investment. It has been a great privilege to work with both of them.

Special thanks go to Bracha Hod and Lior Amar for their support and advice, and to the Distributed Computing lab for the use of MOSIX and clusters.

Finally, I would like to thank my family for their unconditional support and encouragement all along my studies.

Table of Contents

Abstract	2
Acknowledgments	3
1 Introduction	8
2 Related Work	9
3 Problem Statement	10
3.1 Supercomputers and Workloads	10
3.2 Sensitivity of Performance Evaluation	11
4 The Shaking Methodology	14
4.1 Parameters	14
4.2 User options	15
4.2.1 Attributes	16
4.2.2 Percentage of jobs	17
4.2.3 Degree of Shaking	18
4.2.4 The Shaking Formula	18
5 The Effect of the Shaking Parameters	19
5.1 Variation of parameters	19
5.2 The Effect of the Parameters on the Performance Results	21
5.2.1 The Measures	21
5.3 A Default configuration	26
6 Shaking Applied to Sensitive Test Cases	27
6.1 The Butterfly Effect	27
6.2 Simultaneous Job Arrivals	28
6.3 Load Variations	29
6.4 The Prediction Window	30

Table of Contents **5**

6.5	The Estimation Factor	33
6.6	Summary	34
7	Conclusions	36
	Bibliography	38
A	CPU usage	40

List of Figures

3.1	Bounded slowdown as a function of utilization, CTC log.	12
4.1	Flow diagram of the shaking methodology.	15
4.2	The user's options for shaking configuration	16
4.3	Diagram of job interchange	17
5.1	Distribution of bsld, shaking 1% of interarrivals, SDSC	20
5.2	Distribution of bsld, shaking 10% and 100% of interarrivals, SDSC	20
5.3	The histograms of bsld on Estimate with 10% and 100% shaking	21
5.4	Metrics used to quantify the effect of shaking.	22
5.5	Span of bounded slowdown when shaking interarrivals	23
5.6	Span of bounded slowdown when shaking runtimes	23
5.7	Results of the bounded slowdown on Estimate and Size attribute	24
5.8	Results of the Distance metric	24
5.9	Concentration of the wait time metric	25
5.10	Concentration of the bounded slowdown metric	25
6.1	Bsld as a function of load variation, with and w/o shaking, ab. shaking	29
6.2	Running mean of one run at load 0.85	30
6.3	Difference of perturbations between the original run and 1hr shaking	31
6.4	Difference of perturbations between the original run and 10hr shaking	31
6.5	Reducing instability by relative shaking, on Bounded slowdown	32
6.6	Reducing instability by absolute shaking, on Bounded slowdown	32
6.7	Reducing instability by relative shaking, on Wait time	33
6.8	Reducing instability by absolute shaking, on Wait time	33
6.9	Bounded slowdown as a function of the badness factor, rel. shaking	34
6.10	Bounded Slowdown as a func. of the badness factor on abs. shaking	34
6.11	Wait time as a function of the badness factor, rel. shaking	35
6.12	Wait time as a function of the badness factor, abs. shaking	35

List of Tables

3.1	Logs	11
6.1	Bounded slowdown with and w/o the 30-second truncation abs. shaking	28
6.2	Bounded slowdown with and w/o the 30-second truncation rel. shaking	28
6.3	Wait time with and w/o the 30-second truncation abs. shaking	28
6.4	Wait time with and w/o the 30-second truncation rel. shaking	28
A.1	CPU Hours for simulations in 5 months	40

Chapter 1

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 Chapter 6.

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. Shaking is similar to using multiple measurements to characterize the distribution of results, and calculating an average and confidence interval. 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. Chapter 2 discusses related work, and 3 elaborates on our motivation. Details of the methodology are described in Chapters 4 and 5.

Chapter 2

Related Work

Instability in performance evaluations has not been studied in depth. Our work is a followup to that of Tsafirir and Feitelson [3], who traced an observed instability to workload flurries.

In another study on job scheduling, England et al. [4] 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 [5] to quantify the maximal difference between the CDFs with and without perturbations added to the system.

In a paper by Lawson and Smirni [6], 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). Thus it seems that multiple-queue backfilling may also be sensitive to unique circumstances in the simulation.

Alameldeen and Wood discussed the variability of results of architectural simulations of multi-threaded workloads in [7], 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.

Chapter 3

Problem Statement

3.1 Supercomputers and Workloads

Supercomputers are powerful machines supporting the execution of parallel jobs. They are designed with many processors, each with its own memory. When a job arrives at the supercomputer, it starts running or waits until enough processors become available. More precisely, a job that cannot run immediately is placed in the waiting queue, which is controlled by the computer's scheduler. The scheduler chooses which job in the waiting queue is the next to run. In order to make this decision, the scheduler uses the job's parameters, such as its size (the number of processor needed) and the runtime estimation made by the user.

The scheduler that we used for our simulations is called EASY backfilling [2], which is currently the most common method for parallel-job scheduling. The EASY scheduler uses aggressive backfilling, i.e., small jobs are moved ahead to fill in holes in the schedule, as long as they do not delay the first job in the queue. To compute the reservation of the first job, users are required to provide a runtime estimate for all submitted jobs. Note that if the running time of a job exceeds its estimation time, the job is killed meaning that the scheduler enforces the runtime estimate.

Below are three main metrics to measure performance:

- **Wait time:** The wait time (w) metric is the time elapsed from the job's submit time until its starting time.
Note that the response time is defined as the waiting time + running time.
- **Slowdown:** The slowdown metric is the response time normalized by the running time (r): $sd = \frac{w+r}{r}$; the slowdown reflects the relative delay factor of the

job. If a job’s runtime is one hour and it waits two hours to be scheduled, then it endured a slowdown of three.

- **Bounded slowdown:** The bounded slowdown limits the influence of very short jobs on the metric. The running time is used to normalize the response time only if it is higher than a certain threshold value. This prevents very short jobs from creating exceeding high values. A commonly used threshold of 10 seconds was set, yielding the formula $\textit{bounded slowdown} = \max\left(1, \frac{w+r}{\max(10,r)}\right)$.

The *workload* on a supercomputer is composed of a stream of jobs that run on it. A workload trace is a record of these jobs in a log file. Briefly, logs are given in plain ASCII text. The top of each such log describes the general aspects of the respective workload: which machine generated it, how many processors it has, what are its queues and partitions, etc. The body of the log is a sequence of lines, such that each line represents a job. A line is composed of eighteen fields (as defined by the SWF) [1] separated by whitespace. Each field specifies a job attribute: the jobs arrival time, runtime, estimate, processors number, user, group, memory size, completion status, executable, queue, partition and more. A job is essentially characterized by the following fields: (1) arrival time, also called submit time, (2) runtime or execution time, (3) size, i.e., the number of processor needed, and (4) runtime estimate, provided by the user.

The simulations in this paper were based on four workloads from supercomputers.

Table 3.1: Logs used in our examples, available from [11].

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
KTH SP2	1996-97	28,489	KTH-SP2-1996-1

The workloads listed above executed thousands of jobs submitted by hundreds of users over 1-3 years. They are used as inputs to the simulations in order to analyze scheduling algorithms and evaluate their performance.

3.2 Sensitivity of Performance Evaluation

We found several examples 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 3.1), 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 8.3% in the average bounded slowdown of all the jobs in the trace [3]. 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.

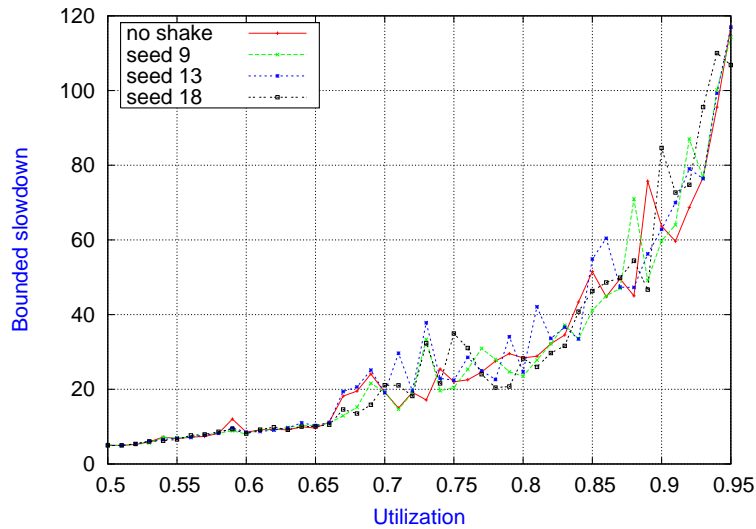


Figure 3.1: Bounded slowdown as a function of utilization, CTC log.

Another example appears in Figure 3.1, 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 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 [3], and are usually a combination of unfavorable circumstances. Our purpose is not to resolve them, but to prevent their influence on the performance evaluation.

Chapter 4

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. Figure 4.1 shows a flow diagram of the shaking steps.

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 Chapter 6. Of course if fewer repetitions are used the cost is reduced.

The main methodological questions are what parameters to shake, and to what degree.

4.1 Parameters

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

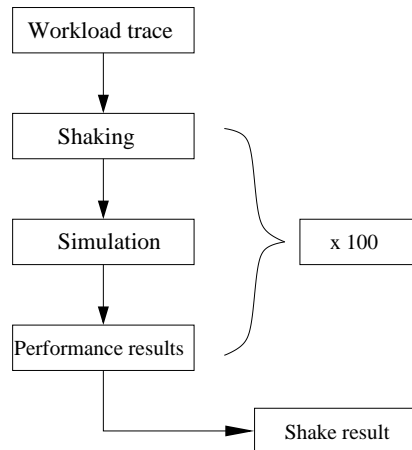


Figure 4.1: Flow diagram of the shaking methodology.

shouldn't affect neither the scheduler's strategies nor the overall results of the performance evaluation.

It is with this assumption that we performed small perturbations, collected a set of performance evaluations, and finally calculated an average to establish a reliable evaluation.

Below are four parameters that define the character of shaking. Each will be examined in turn in this chapter.

1. **Attribute** to shake: Arrival time, size, runtime, or estimated runtime.
2. **Degree** of shaking: Time attributes (arrival, runtime, and estimation) are measured in seconds; size is measured in number of processors.
3. **Absolute or Relative**: Absolute change uses only the degree, while relative change uses the degree and a percentage of the original value to be shaken.
4. **Percentage**: The percentage of jobs to shake out of all the jobs in the workload.

4.2 User options

In Figure 4.2, we see how to perform shaking from a user's point of view. The user chooses a workload, an attribute, a perturbation level and a percentage of jobs to be shaken. The outcome, i.e., the performance evaluation is measured by the bounded slowdown or the wait time.

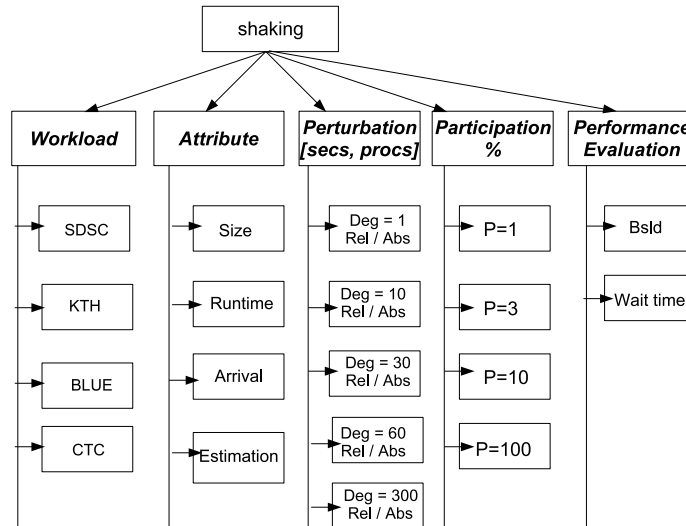


Figure 4.2: The user's options for shaking configuration

4.2.1 Attributes

As mentioned, a job in a workload is defined by many attributes, but its essence is characterized by the arrival time, the runtime, the estimated runtime and the size. The first three attributes are called *time attributes*, and are expressed in seconds; the size represents the number of processors required by the job.

Below is a description of each attribute and its characteristics:

- 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 to the original previous arrival, so as to avoid accumulation of perturbations. As a result some localized jobs can be interchanged. For example, in Figure 4.3 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.

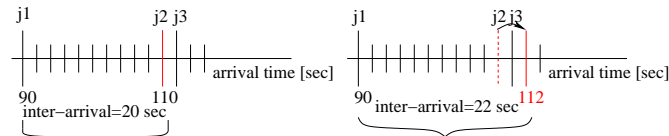


Figure 4.3: Example of job interchange as a result of shaking.

- 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 [10]. Shaking the user estimate or job size are considered even stronger, and therefore less desirable.
- 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.
- The *estimated runtime* attribute changes the estimation time of a job provided by the user. Note that estimates are generally inflated because users do not want the system to kill their jobs in case the actual runtime exceeds the estimate. Modifying the estimated runtime does not change the actual runtime of the job. However, given that estimates tend to be modal (e.g. 5 minutes, 15 minutes, or 2 hours [8], [9]), shaking may actually give the scheduler more information by making the jobs distinct from each other and add some level of noise into the scheduler, since backfilling [2] relies on the estimations. This risks affecting the behavior of the scheduler.

Considering the level of shaking, and the effect on the scheduler, the preference order for the attributes is defined as follows: (1) arrival, (2) runtime, (3) estimated runtime, and (4) size.

4.2.2 Percentage of jobs

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.

4.2.3 Degree of Shaking

The *degree* of shaking denotes the maximal magnitude of the perturbation performed on the jobs. The shaking becomes stronger as the degree increases. Degree depends on two factors: first, the user expresses how strong the shakes ought to be; second, he indicates whether shaking is 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 consideration the percentage of the original value to be shaken.

$$pert = \min\{deg, rel_pct \times orig_val\} \quad (4.1)$$

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 on size of 5 processors with a relative percentage of 10% is applied to a job using 21 processors, then the 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.

4.2.4 The Shaking Formula

The complete shaking formula is:

$$pert = \min\{deg, rel_pct \times orig_val\} \quad (4.2)$$

$$new_val = orig_val \pm pert \times rand \quad (4.3)$$

The formula 4.3 modifies the original value to the left or right by the perturbation value. The \pm means determines if the change is left or right, with the same probability.

Chapter 5

The Effect of the Shaking Parameters

5.1 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 chapter contribute to our understanding of the impact of shaking, and to our ability to set ranges of reasonable shaking values.

In each experiment, shaking is repeated 100 times with different random seeds, producing a distribution of results. This distribution is displayed in two ways.

- Cumulative Distribution Function (CDF) - For every real number x , the probability that a result takes a value less than or equal to x . The CDF is given by $F(x) = P(X \leq x)$.
- Probability Density Function (PDF) - the smoothed-out version of a histogram.

Figures 5.1 and 5.2 shows the bounded slowdown metric when shaking the inter-arrival 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. 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. Many variations were checked, but just a few are presented here to avoid a surplus of graphs.

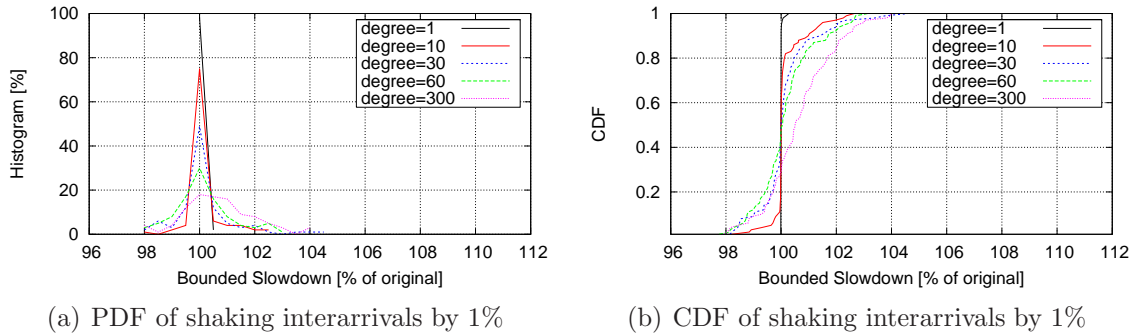


Figure 5.1: Distribution of bsld, shaking 1% of interarrivals, SDSC

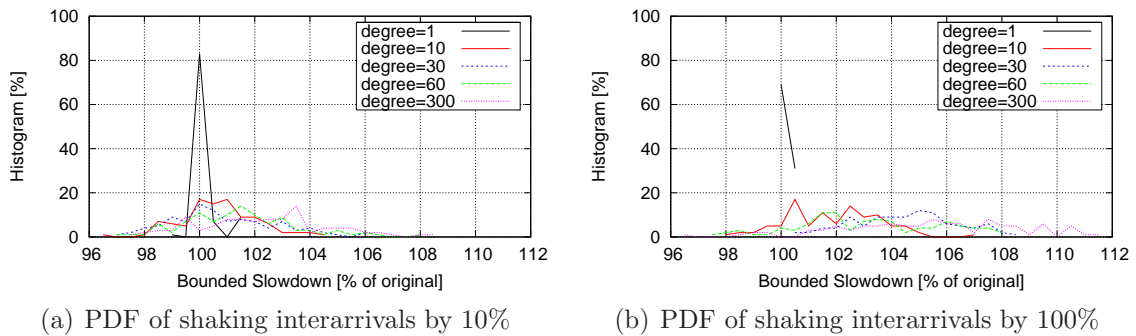


Figure 5.2: Distribution of bsld, shaking 10% and 100% of interarrivals, SDSC

Figures 5.3 show the results on the runtime estimation. We see a smaller span of results compared to figure 5.2 since the effect of the estimation on the scheduler is reduced relative to the effect of the arrival attribute (as mentioned in Section 4.2.1).

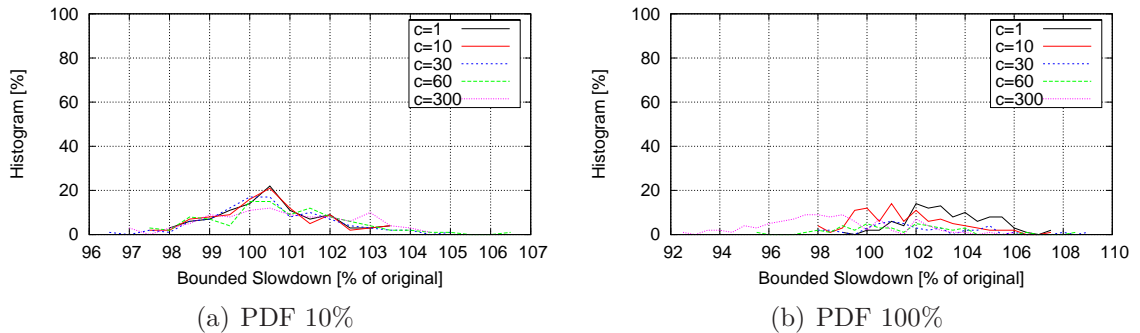


Figure 5.3: The histograms of bsld on Estimate with 10% and 100% shaking

5.2 The Effect of the Parameters on the Performance Results

Similar results were obtained for other workloads and attributes; overall, millions of simulations were performed to 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.

5.2.1 The Measures

The measures adopted for evaluating the effect of shaking are presented in Figure 5.4. 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.

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 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 Figure 5.5 and 5.6.

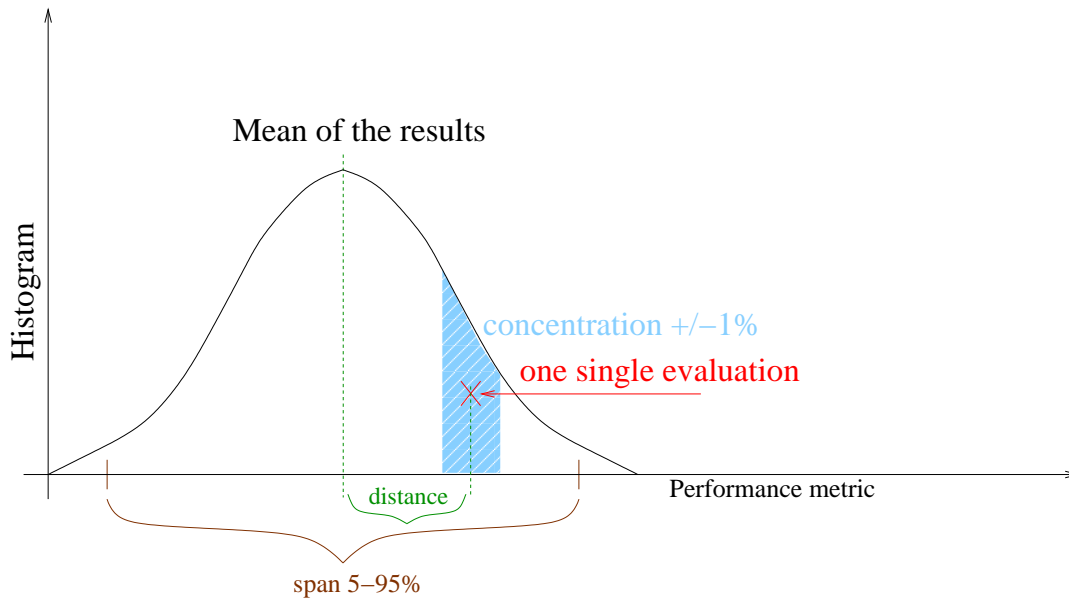


Figure 5.4: Metrics used to quantify the effect of shaking.

Figure 5.5 shows the effect of shaking on the span. We see in figure 5.5(a) that a minimal percentage of 10% and a degree of 10–30 seconds are required to get a real impact. Less than that is not effective, and more perturbation doesn't make a difference. In figures 5.6 and 5.7, we see similar measurements for the runtime and estimate attributes. They 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.

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. Figures 5.8 show 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.

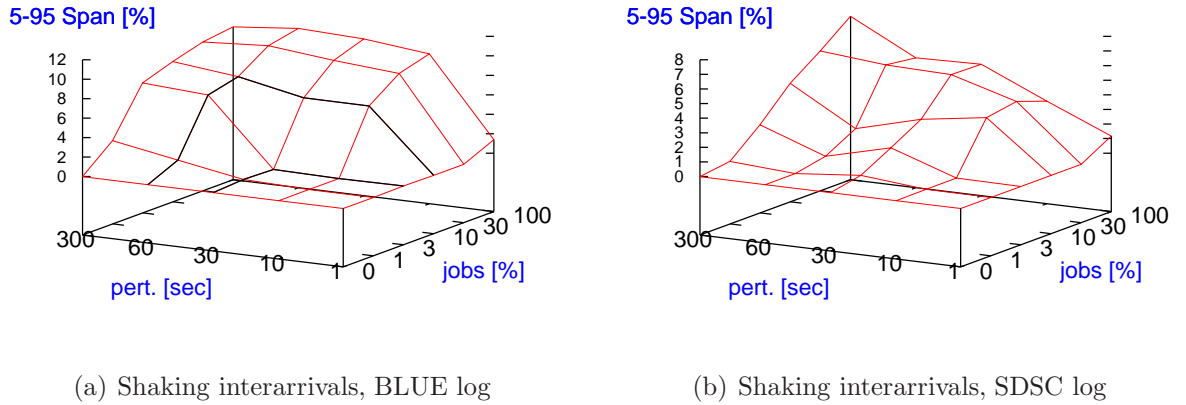


Figure 5.5: Span of bounded slowdown when shaking interarrivals

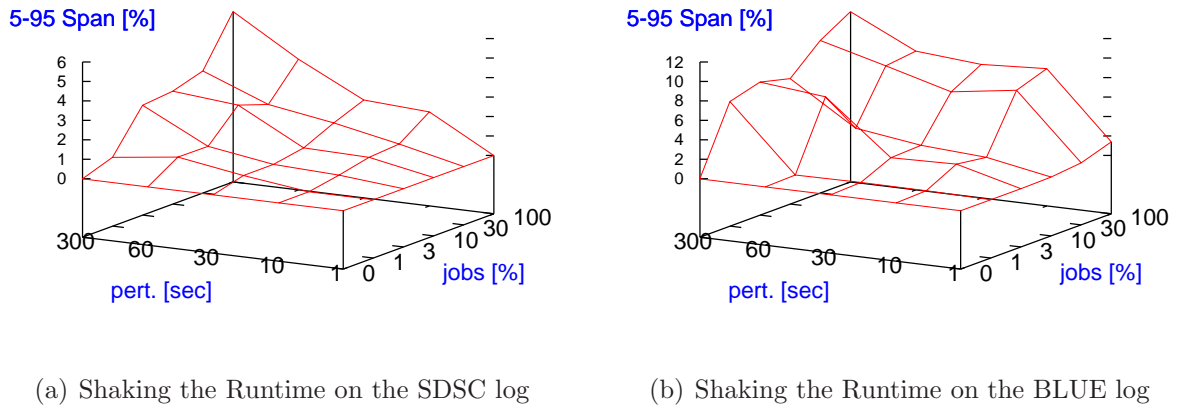
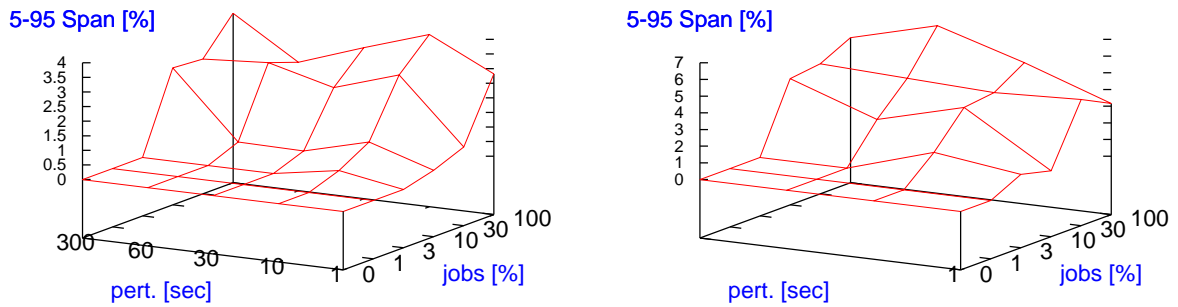


Figure 5.6: Span of bounded slowdown when shaking runtimes

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 chapter.

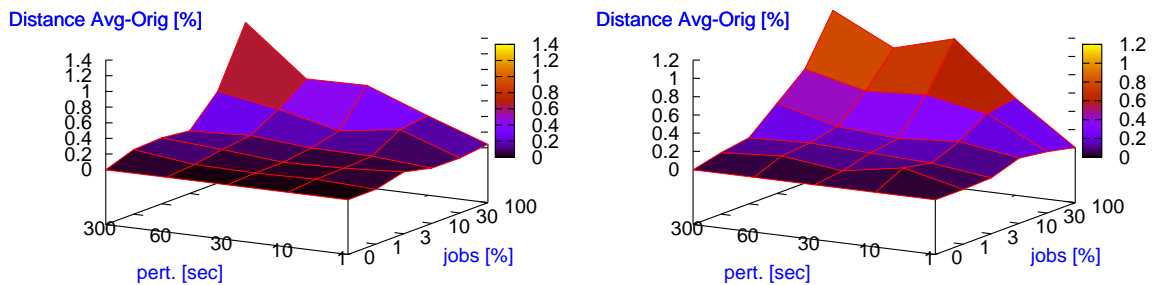
Figure 5.9(a) shows the concentration of results when shaking the interarrival attribute, based on the SDSC log. The figure shows that, even with the most aggressive



(a) Shaking the Estimate on the SDSC log

(b) Shaking the Size on the CTC log

Figure 5.7: Results of the bounded slowdown on Estimate and Size attribute

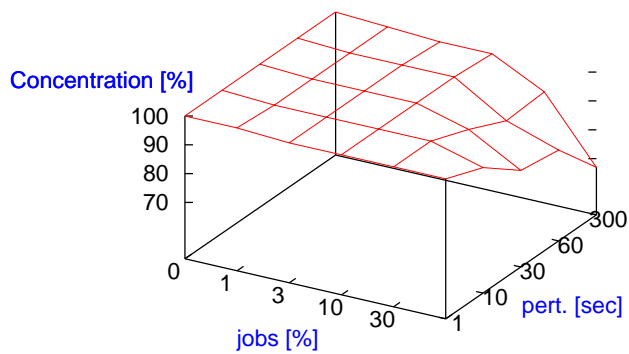


(a) Distance of average wait time from original, shaking runtimes on the SDSC log.

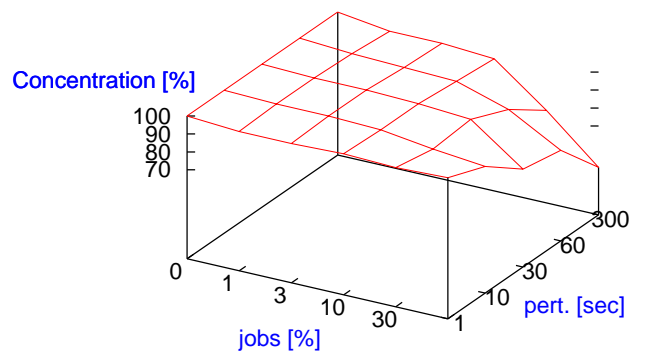
(b) Distance of average wait time from original, shaking interarrivals on the BLUE log.

Figure 5.8: Results of the Distance metric

shaking, 65% of the results were in the interval of 99–101%. Thus, most results remain close to the original evaluation. In Figure 5.10(a), 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.

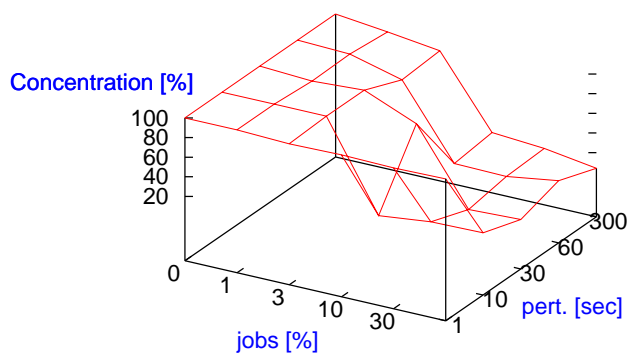


(a) Shaking interarrivals, SDSC log

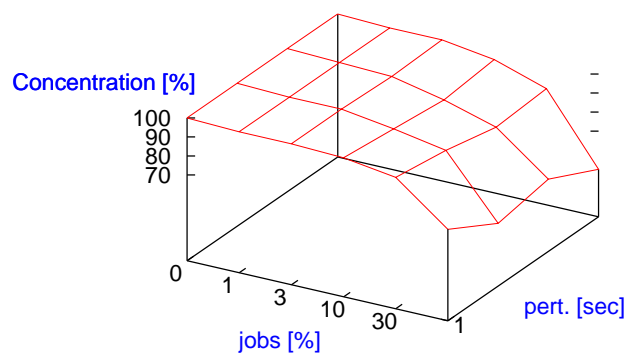


(b) Shaking runtimes, BLUE log

Figure 5.9: Concentration of the wait time metric



(a) Shaking interarrivals, BLUE log



(b) Shaking size, SDSC log

Figure 5.10: Concentration of the bounded slowdown metric

5.3 A Default configuration

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.

Chapter 6

Shaking Applied to Sensitive Test Cases

In this chapter, 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 3.1), which represent real parallel production environments. Specifically, we present five such evaluations, and study the impact of shaking on each.

6.1 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 [3], 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, 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 6.1 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 absolute shaking degrees on the interarrival time attribute. Even a small degree of one minute on all jobs removes the instability yielding a 0.3% difference with and without the truncation.

Table 6.2 makes the same comparison, with relative shaking. Here as well there is

a reduction of the difference (with and without the 30 second truncation) even with small degrees. Relative shaking performs a relative change on the interarrival time attribute which yields to more stable results.

Table 6.1: Bounded slowdown with and w/o the 30-second truncation abs. shaking

	Original	shake 1mn	shake 5mn	shake 15mn
w/o trunc.	88.15	88.57	90.48	91.66
with trunc.	81.38	88.28	90.57	91.33
abs diff	8.31%	0.3%	0.09%	0.3%

Table 6.2: Bounded slowdown with and w/o the 30-second truncation rel. shaking

	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%

Table 6.3 and 6.4 show the comparison on the wait time performance metric. Based on this metric, the original difference is much smaller, still shaking erases the difference.

Table 6.3: Wait time with and w/o the 30-second truncation abs. shaking

	Original	shake 1mn	shake 5mn	shake 15mn
w/o trunc.	1865	18895	19014	19247
with trunc.	1813	18873	19021	19220
abs diff	2.8%	0.1%	0.03%	0.01%

Table 6.4: Wait time with and w/o the 30-second truncation rel. shaking

	Original	shake 1mn	shake 5mn	shake 15mn
w/o trunc.	18653	18646	18696	18745
with trunc.	18130	18621	18695	18732
abs diff	8.31%	0.01%	0.005%	0.06%

6.2 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%.

6.3 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 Figure 3.1, 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. Figure 6.1 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.

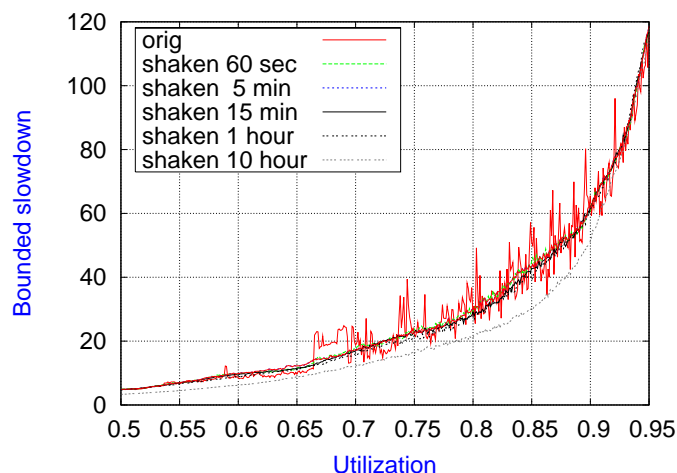


Figure 6.1: Bslsd as a function of load variation, with and w/o shaking, ab. shaking

The shaking curves are close to each other except for the 10h curve, which is below all the others. To understand the reason of this difference, we focused on one point where the difference was high ($load = 0.85$). As mentioned, each point is an average

of 100 runs. The general tendency in these runs shows a lower bounded slowdown on the 10hr curve relative to the 1hr curve. To account for this difference, we analyzed one run where the gap of the bounded slowdown was significant, and calculated the running mean of this run (see Figure 6.2). On the 1h curve, there are peaks that do not appear on the 10hr curve. These peaks explain the accumulated difference in the ends level of the bounded slowdown. The question remains how come peaks appear on small perturbations degree and not on stronger ones? Figures 6.3 and 6.4 show the difference between the original and the shaken arrival time of a job, in the peak area [228,000-230,000]. For example, a job arrived originally at 228,005 sec and was delayed to 228,305 sec due to shaking. The difference in arrival times is 300 sec (5 mn). In the figures, each point represents a job. With the 1h shaking, the majority of jobs are concentrated in two columns representing two bursts of arrival, whereas in the 10h shaking the arrival time moved by [-10h,10h] which caused a spreading of jobs. Thus, changes performed by strong perturbations remove bursts in the running mean. We do not wish to affect the simulation to such a degree; therefore we consider small perturbation to be optimal since it removes the instability from the original run; preserving the character of the workload.

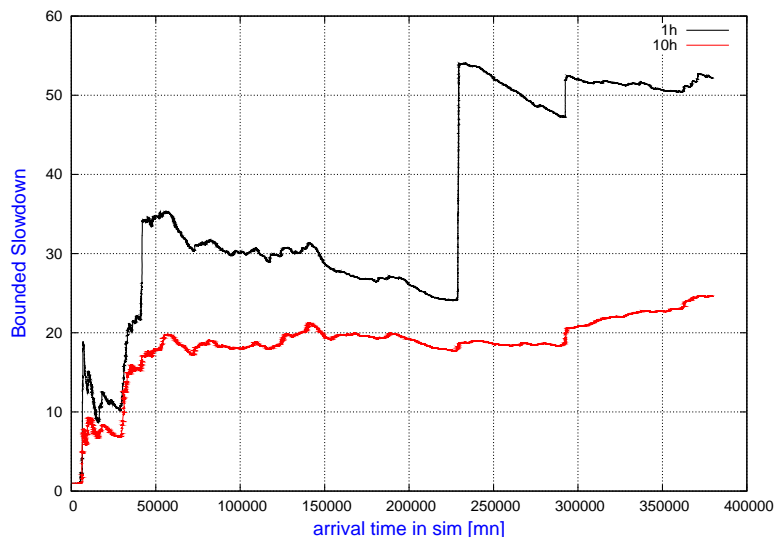


Figure 6.2: Running mean of one run at load 0.85

6.4 The Prediction Window

The first sensitivity example in Chapter 1 concerned the prediction of job runtimes by averaging the runtimes of the last k jobs by the same user [12]. k is called the

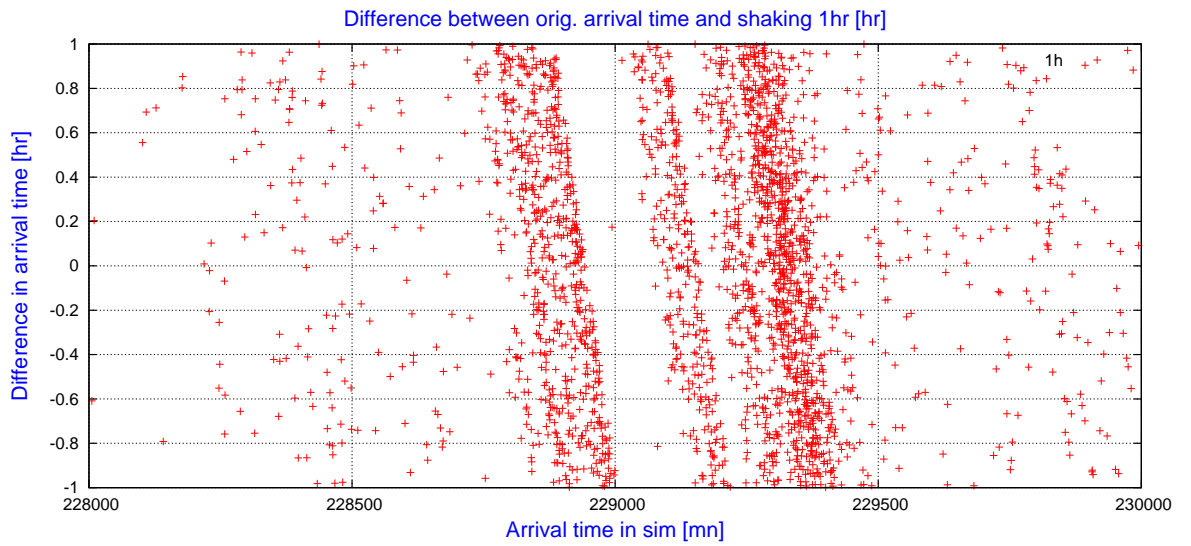


Figure 6.3: Difference of perturbations between the original run and 1hr shaking

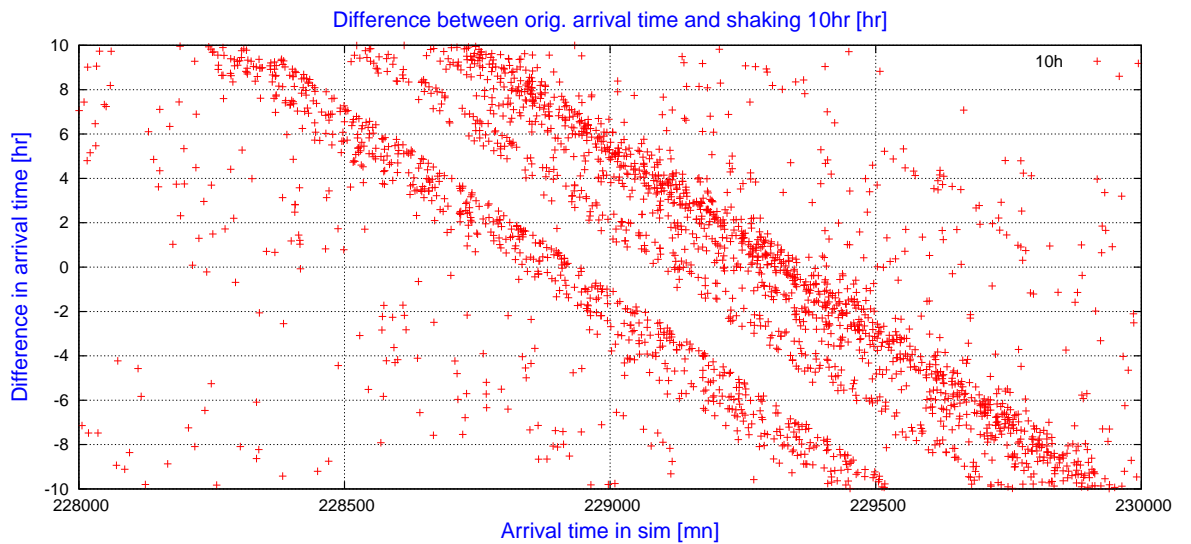


Figure 6.4: Difference of perturbations between the original run and 10hr shaking

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.

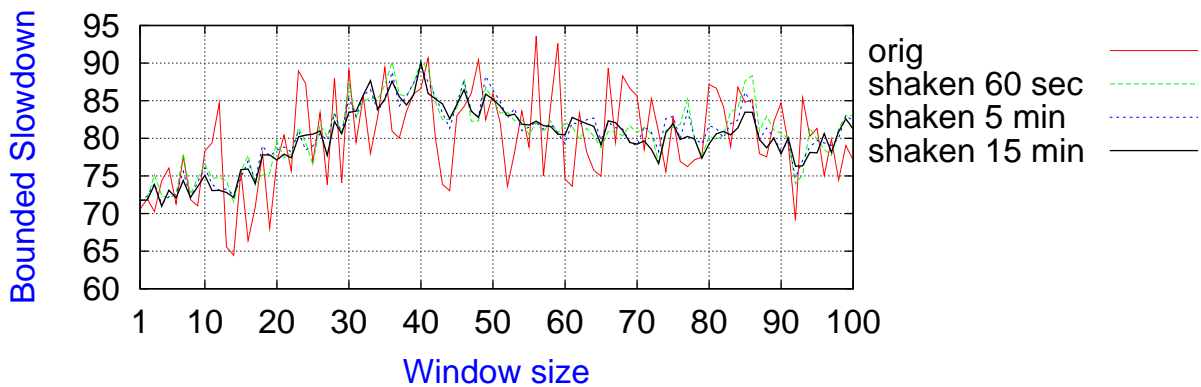


Figure 6.5: Reducing instability by relative shaking, on Bounded slowdown

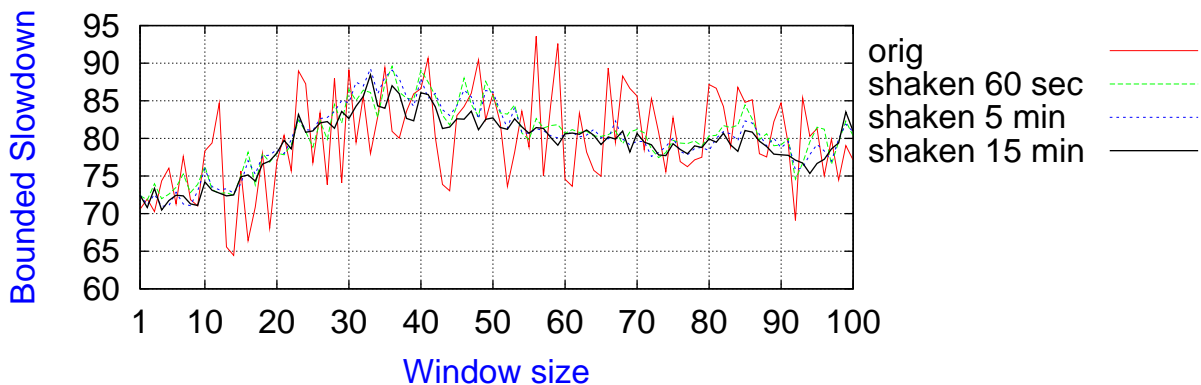


Figure 6.6: Reducing instability by absolute shaking, on Bounded slowdown

Figure 6.5 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 than 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. Figure 6.6 shows the bounded slowdown with absolute shaking. The results are smoother and show less sensitivity. Figure 6.7 and 6.8 show the shaking results on the wait time metric with relative and absolute shaking respectively.

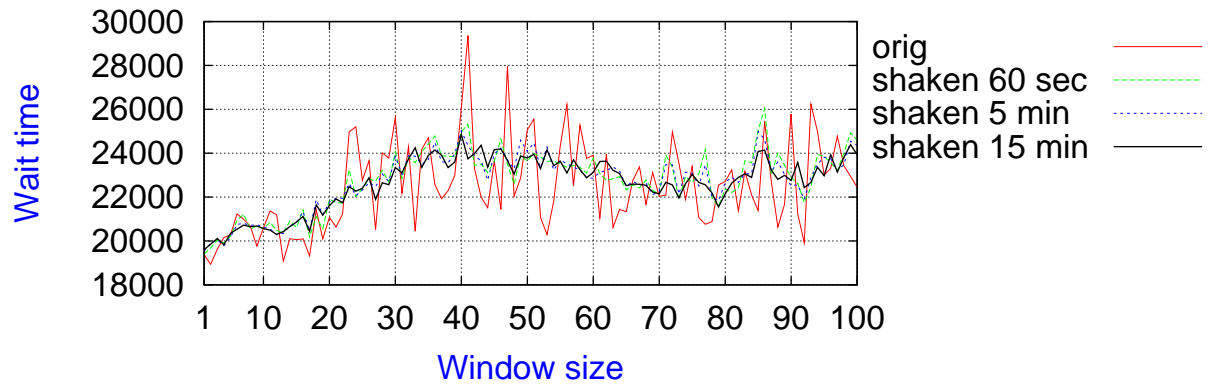


Figure 6.7: Reducing instability by relative shaking, on Wait time

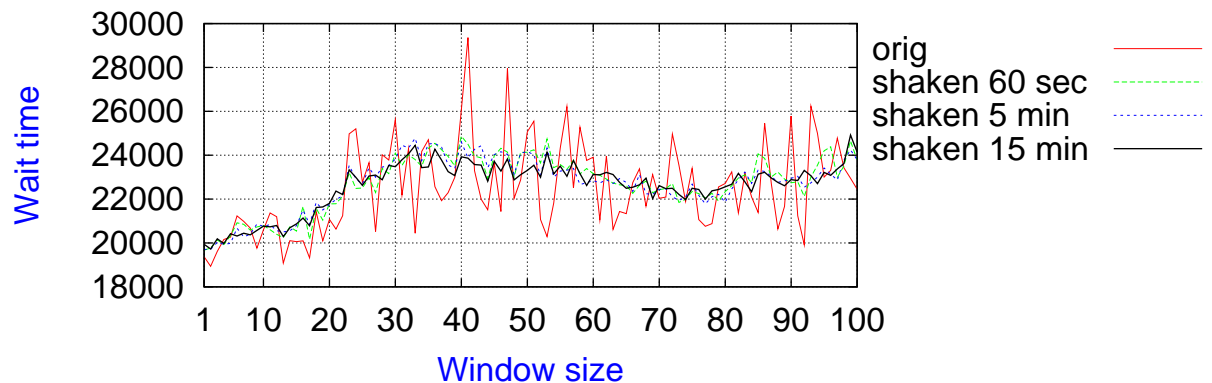


Figure 6.8: Reducing instability by absolute shaking, on Wait time

6.5 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). 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 [13]. In Figures 6.9, 6.10, 6.11 and 6.12 we see a comparison between the shaken result and the original evaluation. Without shaking, the performance is noisy and unstable. Both relative

and absolute shaking have a smoothing effect for both the bounded slowdown and the wait time metrics.

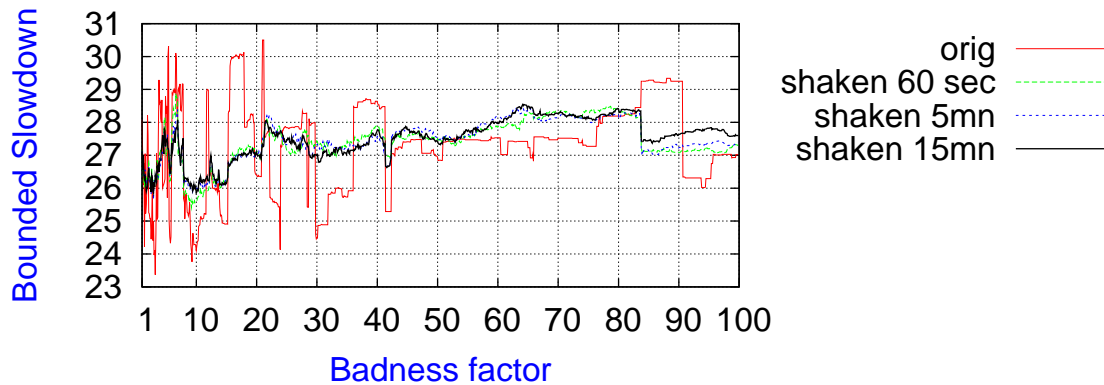


Figure 6.9: Bounded slowdown as a function of the badness factor, rel. shaking

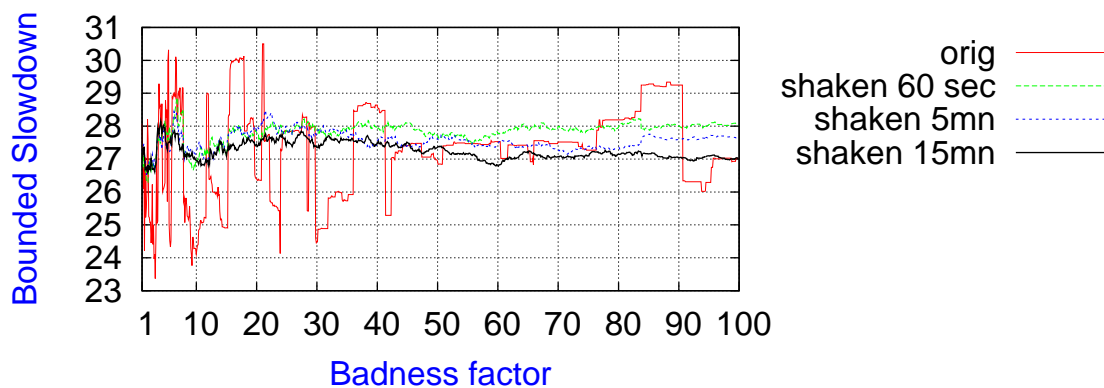


Figure 6.10: Bounded Slowdown as a func. of the badness factor on abs. shaking

6.6 Summary

Regarding the shaking parameters, multiple examples show that relative shaking is more robust: it retains the characteristics of the original workload, 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. 15 min. Note that even with shaking, the simulation results are never perfectly smooth. However, they exhibit

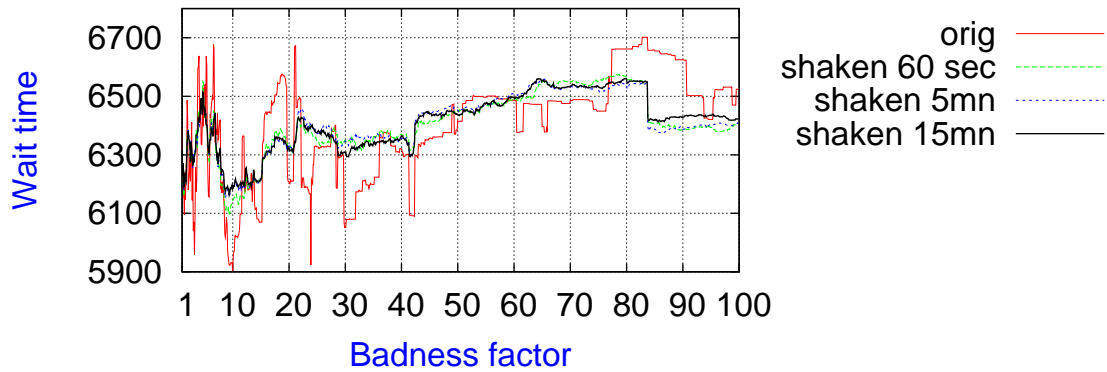


Figure 6.11: Wait time as a function of the badness factor, rel. shaking

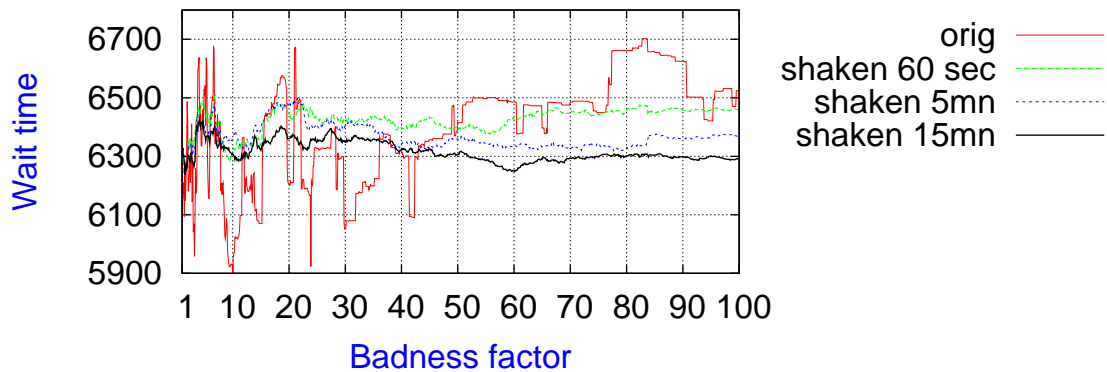


Figure 6.12: Wait time as a function of the badness factor, abs. shaking

much less sensitivity than the original results with no shaking. This also implies that the sensitivity of the original results is actually an artifact, and that the shaken results are more reliable.

The aforementioned cases showed the instability of single results, and the effect of various shaking configuration on each of them. The average of multiple runs, where each run is slightly different, yields a final outcome that is more representative.

Chapter 7

Conclusions

Our first contribution in this thesis 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.

The 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.

Bibliography

- [1] David Talby. The standard workload format. <http://www.cs.huji.ac.il/labs/parallel/workload/swf.html>.
- [2] 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.
- [3] D. Tsafir and D. G. Feitelson. Instability in parallel job scheduling simulation: The role of workload flurries. In *IPDPS*, Apr 2006.
- [4] D. England, J. Weissman, and J. Sadagopan. A new metric for robustness with application to job scheduling. In *HPDC-14*, July 2005.
- [5] 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.
- [6] B. G. Lawson and E. Smirni. Multiple-queue backfilling scheduling with priorities and reservations for parallel systems. In *JSSPP*, pp. 72–87. LNCS vol. 2537, Springer Verlag, 2002.
- [7] A. R. Alameldeen and D. Wood. Variability in architectural simulations of multi-threaded workloads. In *HPCA-9*, Feb 2003.
- [8] D. Tsafir, Y. Etsion, and D. G. Feitelson. Modeling user runtime estimates. In *JSSPP*, pp. 1–35. LNCS vol. 3834, Springer-Verlag, 2005.
- [9] C. Lee, Y. Schwartzman, J. Hardy, and A. Snavely. Are user runtime estimates inherently inaccurate?, *JSSPP* 2004. <http://citeseer.ist.psu.edu/lee04are.html>.
- [10] D. G. Feitelson. Locality of sampling and diversity in parallel system workloads. In *21st ICS*, Jun 2007.

-
- [11] Parallel workloads archive. www.cs.huji.ac.il/labs/parallel/workload.
- [12] D. Tsafir, 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.
- [13] D. Tsafir and D. G. Feitelson. The dynamics of backfilling: solving the mystery of why increased inaccuracy may help. In *IISWC*, Oct 2006.

Appendix A

CPU usage

The calculations on the HUGI clusters for our simulations took cpu time as presented below. The hours refer only to the period between the 30/10/06 to 10/03/07.

User	klee	amos-t	bmos	cmos	dlk	os	xil	cab	amos-v	Total
ouaknine	1295	40	3980	7740	5946	6712	2526	7277	1065	36581

Table A.1: CPU Hours for simulations in 5 months