Experimental Analysis of the Root Causes of Performance Evaluation Results: A Backfilling Case Study

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

Abstract

The complexity of modern computer systems may enable minor variations in performance evaluation procedures to actually determine the outcome. Our case study concerns the comparison of two parallel job schedulers, using different workloads and metrics. It shows that metrics may be sensitive to different job classes, and not measure the performance of the whole workload in an impartial manner. Workload models may implicitly assume that some workload attribute is unimportant and does not warrant modeling; this too can turn out to be wrong. As such effects are hard to predict, a careful experimental methodology is needed in order to find and verify them.

Keywords:
C.1.4.d Scheduling and task partitioning
C.4.g Measurement, evaluation, modeling, simulation of multiple-processor systems
D.4.8.f Simulation
K.6.2.d Performance and usage measurement

1 Introduction

The goal of performance evaluation is often to compare different system designs or implementations. The evaluation is expected to bring out performance differences that will allow for an educated decision regarding what design to employ or what system to buy. Thus it is implicitly assumed that observed performance differences indeed reflect important differences between the systems being studied.

However, performance differences may also be an artifact of the evaluation methodology. The performance of a system is not only a function of the system design and implementation. It may also be affected by the workload to which the system is subjected, and even by the metric being used to gauge the performance. Thus it is important to understand the effect of the workload and metrics on the evaluation. To complicate matters, in some cases the effect is not due to the metric or workload alone, but rather to an interaction between them and the system [3]. We present a case study of a detailed analysis of such a situation.

The domain we will use is the scheduling of parallel jobs for execution on a parallel supercomputer. Such scheduling is typically done by partitioning the machine's processors and running a job on each partition. This is similar to packing in two dimensions. Regard one dimension as representing processors, and the other as representing time. A parallel job is a rectangle, representing the use of a certain number of processors for a certain duration of time. The scheduler has to pack these rectangles as tightly as possible, within the space provided by the available resources. The sizes of the rectangles are known, as each submitted job comes with a specification of how many processors to use, and an estimate of how long it will run.

Given that jobs come in various sizes, they typically do not pack perfectly. Therefore holes are left in the schedule. Backfilling is the optimization of trying to fill in these holes using newly submitted small jobs. Two main versions of backfilling have been studied. EASY backfilling is aggressive in trying to maximize system utilization [10]. When the first queued job cannot be scheduled because enough processors are not available, it calculates when the required processors are expected to become free (based on the running jobs' runtime estimates), and makes a reservation to run this job at that time. It then continues to scan the queue of waiting jobs, and allocates processors to any job that is small enough and will not interfere with the commitment to run the first queued job. Conservative backfilling places a greater emphasis on predictability [13], and makes a reservation for every queued job.

The two versions of backfilling make different tradeoffs regarding the number of reservations and their effect on backfilling and predictability. The question is then which of the two approaches is better. Our case study is a set of simulations designed to answer this question.

2 Evaluation Methodology and Results

We perform the evaluation by discrete event simulation. The events of interest are the arrival and termination of jobs. Each job requires a certain number of processors, runs for a certain time, and also has a user estimate of its runtime. As each job terminates, the quality of the service it had received is tabulated. One of the most commonly used metrics for scheduling is the average response time (the time from when a job was submitted until it terminated). An alternative is the slowdown (the response time normalized by the job's actual running time). This measures how much slower the job ran due to scheduling conflicts with competing jobs, and seems to better capture users' expectations that a job's response time will be proportional to its runtime. We use a version known as "bounded slowdown" [5], where a threshold of 10 seconds is used rather than the actual runtime for very short jobs.

The simulation uses the batch means approach to evaluate confidence intervals for the average response time and slowdown [6]. The batch size is set to be 5000 jobs, as recommended by MacDougall for open systems under high load [12]. Depending on the length of the workload log, 10 to 17 batches were completed, and the first discarded to bring the system to a steady state. We also performed a more sophisticated analysis of these results using the common random numbers variance reduction technique [8]. In this analysis, we first compute the difference in response times or slowdowns between the two schedulers on a per-job basis, which is possible because we are using the same workload log. We then compute confidence intervals on these differences using the batch means approach. If the confidence intervals do not include zero, the difference is significant.

The study used workloads from the parallel workloads archive (www.cs.huji.ac.il/labs/parallel/workload). The main ones are a log of jobs run on the Cornell Theory Center (CTC) IBM SP2 machine from

July 1996 through May 1997[1], and a model based on the first $2\frac{1}{2}$ months of this workload which was created by Jann et al. [7]. Despite the close relationship between the two workloads, there are several differences:

- In the CTC workload, most jobs request a power-of-two number of nodes. In the Jann workload it was decided not to favor powers of two, based on the assumption that this is not a real attribute of the workload but rather a consequence of administrative policies. This notion was later borne out by a user survey conducted by Cirne and Berman [2].
- The CTC workload had a sharp administrative limit on runtime at 18 hours. In the Jann model there is no such limit, as a continuous distribution is used to represent the runtimes, so some extend beyond 18 hours. At the other extreme, the CTC log hardly has any jobs shorter than 30 seconds, probably because the measurement includes the time to start up all the processes and to report their termination. In the Jann model 10% of the jobs are shorter than 30 seconds, and many are sub-second.
- The Jann model uses a hyper-Erlang distribution to model runtimes and interarrival times. While this matches the first three moments of the original distribution, it also leads to a somewhat bimodal structure that does not exist in the original.
- The Jann model does not include the modeling of user estimates of runtime. We therefore used the actual runtime in the simulations. This is equivalent to assuming that the estimates are perfect.

In addition, we also use a log of jobs run on the San-Diego Supercomputer Center (SDSC) IBM SP2 from May 1998 through April 2000[2], and a model proposed by Feitelson [4]. These two workloads have a similar distribution of job sizes which has fewer small jobs than the CTC and Jann workloads. The Feitelson model is also unique in having many more short jobs. The SDSC log also has shorter jobs than CTC and Jann, but not as short as in the Feitelson model.

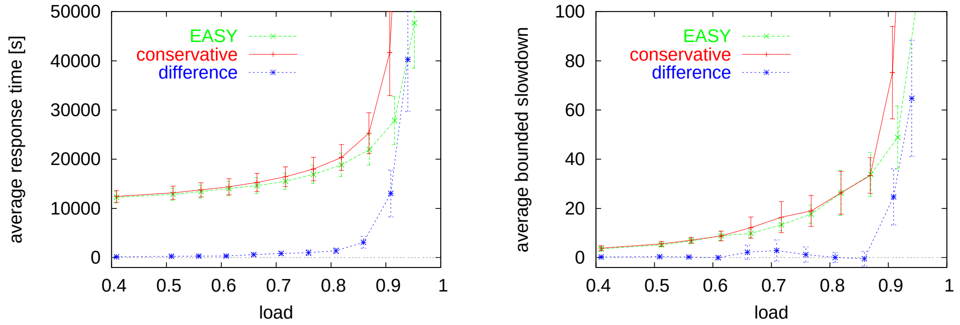
Each workload is represented by a single data file specifying the arrival time, runtime, and number of processors used by each job, and in the case of the CTC and SDSC workloads, also the estimated runtime. In order to create different load conditions, all the arrival times are multiplied by a suitable constant. For example, if the original workload file leads to a load of 0.7 of capacity, multiplying all interarrival times by a factor of $7/8 = 0.875$ will cause the jobs to arrive faster, and increase the load to 0.8 of capacity.

The typical methodology for evaluating systems (in our case, schedulers) calls for simulating their behavior, and tabulating the resulting performance. This is usually repeated numerous times under different conditions, including different load levels and different parameter settings for the system. This enables a study of how the system responds to load, and what parameter settings lead to optimal performance.

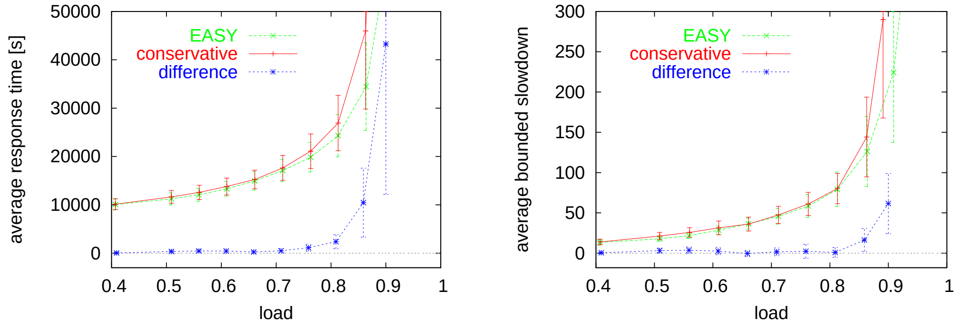
[1] For the simulations and analysis reported here we removed a flurry of 1999 one-node jobs lasting about 42–48 seconds each that were submitted by user 135 in one day; it is assumed that this was the result of a misbehaving script. This large burst of small jobs is very sensitive to details of the scheduling and tends to cause large fluctuations in simulation results. It represents 2.55% of the total workload of 78500 jobs. The justification for removing such flurries is elaborated in [15].

[2] In this log we also removed two flurries, one composed of 30-second one-node jobs by user 365, and the other composed of 1-minute 32-node jobs by user 319 [15]; they are known to affect simulation results, but in the specific simulations used here they seem to have only a marginal effect.

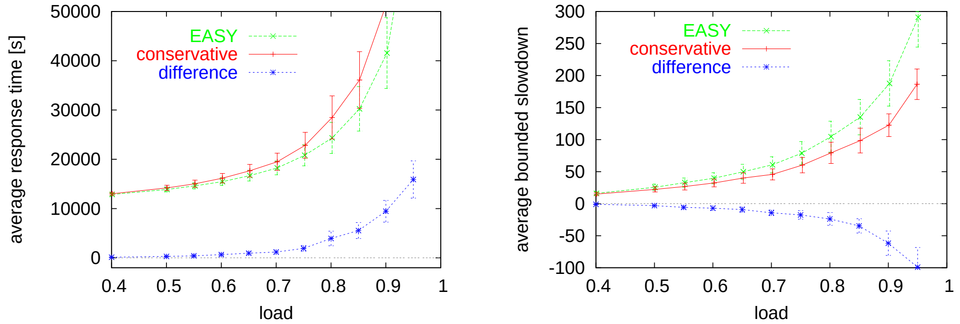
CTC workload



SDSC workload



Jann model



Feitelson model

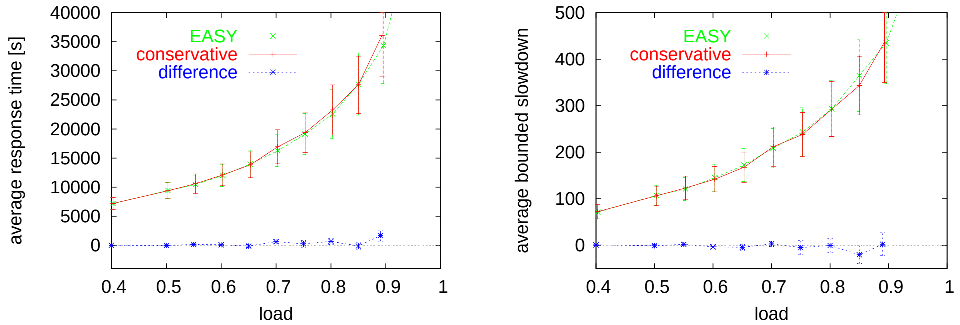


Figure 1: *Simulation results with 90% confidence intervals. "Difference" is conservative minus EASY.*

In this paper we emphasize two other dimensions instead: the workload used to drive the simulation, and the metric used to measure the results. Figure 1 shows results of comparing our two schedulers using the four different workloads and two different metrics. In all cases, simulations are run for different load levels, and the behavior of the system as a function of load is plotted. These results can be divided into four groups:

- Results which show that EASY is better than conservative. These include the results obtained with the CTC, SDSC, and Jann workloads, when using the response time metric.
- Results which show that conservative is better than EASY. Such results were obtained from the Jann workload when using the bounded slowdown metric.
- Results showing that both schedulers are essentially the same. The results from the Feitelson model fall into this class.
- A combination: results that show the schedulers to be equivalent under most loads, and show an advantage for EASY only under high load. Such results were observed for CTC and SDSC when using the bounded slowdown metric. It should be noted that such high loads are seldom achieved in practice on this class of machines.

In short, if only a single set of results is used, the conclusions would depend on which specific set was chosen. Our goal in the next sections is to uncover the root causes for these discrepant results.

3 Analysis of Jann vs. CTC Results

As noted above, the Jann model is specifically meant to mimic the CTC workload. Nevertheless, the simulation results indicate that the two produce discrepant predictions. For the Jann workload the response time metric favors EASY backfilling, whereas the slowdown metric favors conservative backfilling. The CTC workload, at the same time, favors EASY for both metrics (albeit for slowdown only under very high load). This is therefore actually a triple interaction of scheduler, workload, and metric. In this section we focus on a detailed analysis of how this triple interaction comes about.

Both the slowdown metric and the backfilling policy are sensitive to job duration. We therefore partitioned the jobs into five classes according to their duration, and tabulated the results for each class separately. The classes used were very short (< 30sec), short (< 5min), medium (< 1hr), long (< 10hr), and very long (> 10hr). In the results for the CTC workload, both metrics favor EASY backfilling for each class individually, and also for all of them together. But in the Jann workload we indeed see a difference that depends on job class. For jobs that are longer than 1 hour, both metrics favor EASY. But for shorter jobs, both metrics favor conservative backfilling. This behavior leads directly to the results quoted above for the whole workload. The average response time of all jobs is dominated by the high values representing long jobs, and is therefore similar to the response time for long jobs, and favors EASY. For the average slowdown, on the other hand, the high values come from the short jobs, because a very small denominator leads to a high value. Thus the average becomes similar to the slowdown of short jobs, which favors conservative.

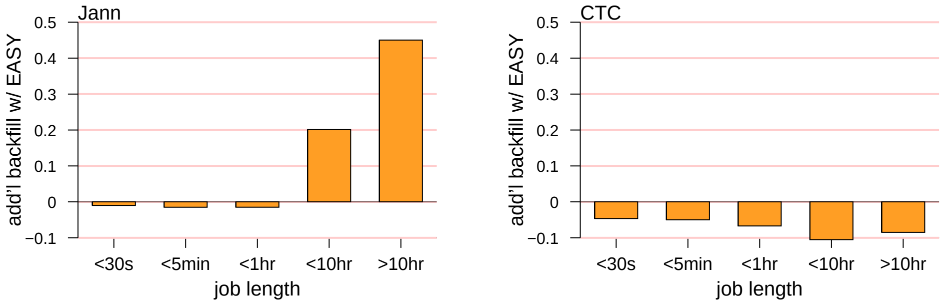


Figure 2: *Increased backfilling with EASY relative to conservative scheduler. Data for load of 0.85.*

The breakdown into classes indicates that the difference between the CTC and Jann results is due to the short jobs, which fare better under conservative in the Jann workload, but not in the CTC workload. To try and understand why this happens, we need to understand how the scheduler interacts with the workload. As the difference between our two schedulers is in their backfilling policy, a good place to start is investigating their backfilling patterns. Figure 2 shows how much more backfilling is achieved by EASY, as a fraction of the backfilling achieved by conservative; thus a value of 0.1 means that EASY did 10% more backfilling. Surprisingly, for CTC conservative backfilling actually achieves somewhat higher backfilling levels! But the main difference between the workloads is that under the Jann workload, EASY achieved much more backfilling of long jobs.

After checking many other possibilities (see below), the root cause for this was found to be the use of accurate runtime estimates with the Jann workload. Accurate runtime estimates provide full information for backfilling decisions. The conservative algorithm has to take multiple reservations into account, and this is especially problematic for long jobs, that have the potential to interact with many other jobs. EASY, on the other hand, only has to consider one reservation. Therefore conservative achieves much less backfilling. But when estimates are inaccurate, jobs tend to terminate before the time expected by the scheduler. This creates holes in the schedule that provide new backfilling opportunities, that can be exploited by both schedulers.

To verify this hypothesis, we re-ran the CTC simulations but using the actual runtimes rather than the original user estimates to control the backfilling. The results, shown in Figure 3, largely confirm the conjecture. When using accurate estimates, conservative performed much less backfilling of long jobs than before. And as expected, this led to an inversion of the results, with conservative achieving better average slowdown scores. A similar but weaker effect also occurred when running the SDSC workload with accurate runtime estimates. The results were inverted, but the difference was much closer to zero.

We are now left with one last question: how does the reduced backfilling under the conservative policy lead to better performance when measured using the slowdown metric? The reduced backfilling applies to *long* jobs. On the other hand, slowdown is sensitive mainly to *short* jobs. Recall that under the EASY policy backfilling is allowed provided it does not delay the first queued job. But it might delay subsequent queued jobs, including short ones, that will then suffer from a very large slowdown. The conservative policy prohibits such backfilling, with the specific goal of

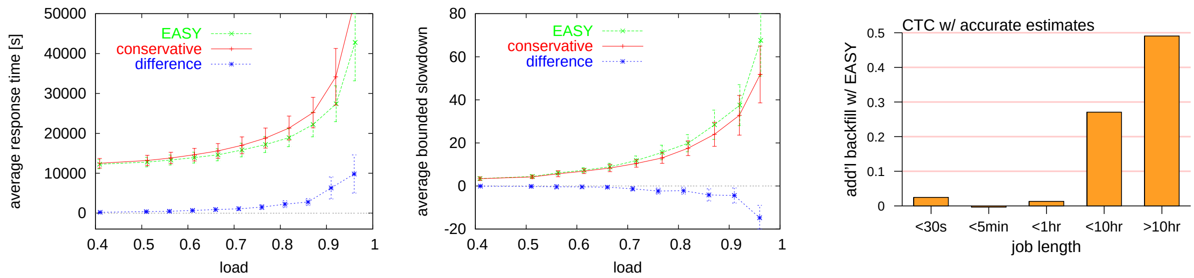


Figure 3: *Results for the CTC workload when using actual runtimes as estimates, to verify that this is the cause of the Jann results. Compare with Figures 1 and 2.*

preventing such delays for subsequent jobs. In the simulations with the Jann workload, this turned out to be the decisive factor. Similar observations have been made independently by Srinivasan et al. [14].

To summarize, our analysis exposed the following triple interaction:

- The Jann and CTC workloads differ (among other things) in that the CTC workload is a real trace including user estimates of runtime, whereas the Jann model does not include this detail.
- Due to using accurate estimates for the Jann model, the conservative scheduler achieved less backfilling of long jobs that use few processors. This is obviously detrimental to the performance of these long jobs, but turned out to be beneficial for short jobs that don't get delayed by these long jobs.
- As response time is dominated by long jobs, the response time metric showed that EASY is better than conservative for the Jann workload. The slowdown metric, on the other hand, is dominated by short jobs, so it showed conservative to be better.

It should be noted that other works have also investigated the effect of user runtime estimates on performance [13, 16, 1]. However, these works did not elucidate the mechanisms by which the inaccurate estimates cause their effect.

Finding that the difference between the CTC and Jann results hinges on the runtime estimates was surprising not only because this is routinely brushed aside as unimportant, but also because there is no lack of other candidates, which seem much more significant. The usual suspects regarding performance variations are the statistical differences between the workloads. The most striking difference is that the Jann workload has tails at both ends, which the CTC workload does not (Section 2).

The long jobs in the tail could affect the results by causing longer delays to other jobs that wait for their termination because they need their processors. To check this, we re-ran the simulations with a modified version of the Jann workload, in which all jobs longer than 18 hours were deleted. The results were essentially the same as for the original workload. The short jobs could affect the results by contributing very high values to the average slowdown metric. This was checked by removing all the jobs shorter than 30 seconds. Again, the results were not significantly different from those of the original workload.

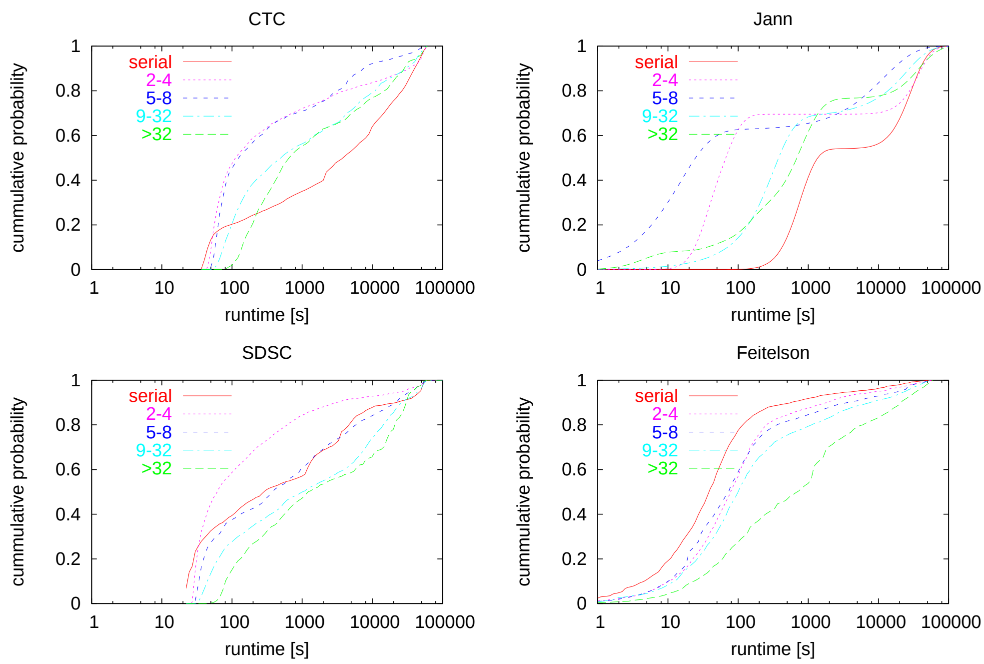


Figure 4: *Distributions of runtimes for jobs of different sizes in the various workloads.*

Another major difference between the workloads is that in the original CTC workload most jobs use power-of-two nodes, whereas in the Jann model jobs are spread evenly between each two consecutive powers of two. Previous work has shown that the fraction of jobs that are powers of two is important for performance, as it is easier to pack power-of-two jobs [11]. However, in our case this seemed not to make a qualitative difference. It was checked by running the simulations on a modified version of the Jann workload in which the sizes of 80% of the jobs (chosen at random) were rounded up to the next power of two.

Finally, the size of the system is also important. The Jann model specifies the use of 322 processors, which is the number used in the simulations. For CTC, we used 430 processors, which is the size of the batch partition on the CTC machine. Repeating the simulations using 512 nodes led to somewhat lower backfilling rates, because 512 is a power of two, and therefore jobs pack much better, leaving less holes in the schedule. However, the relationship between the two schedulers did not change significantly.

4 Comparison with Other Results

Having explained the results obtained using the CTC and Jann workloads, and the reasons for the differences between them, we now turn to the Feitelson workload (the behavior of the SDSC workload is less interesting as it is similar to that of CTC). Comparing this workload with the previous two, the question is why no major differences are observed between the performance of the two schedulers in this case.

A special feature of the CTC and Jann workloads is that they have very many serial jobs, and moreover, that most long jobs are serial (Figure 4). We may therefore conjecture that the

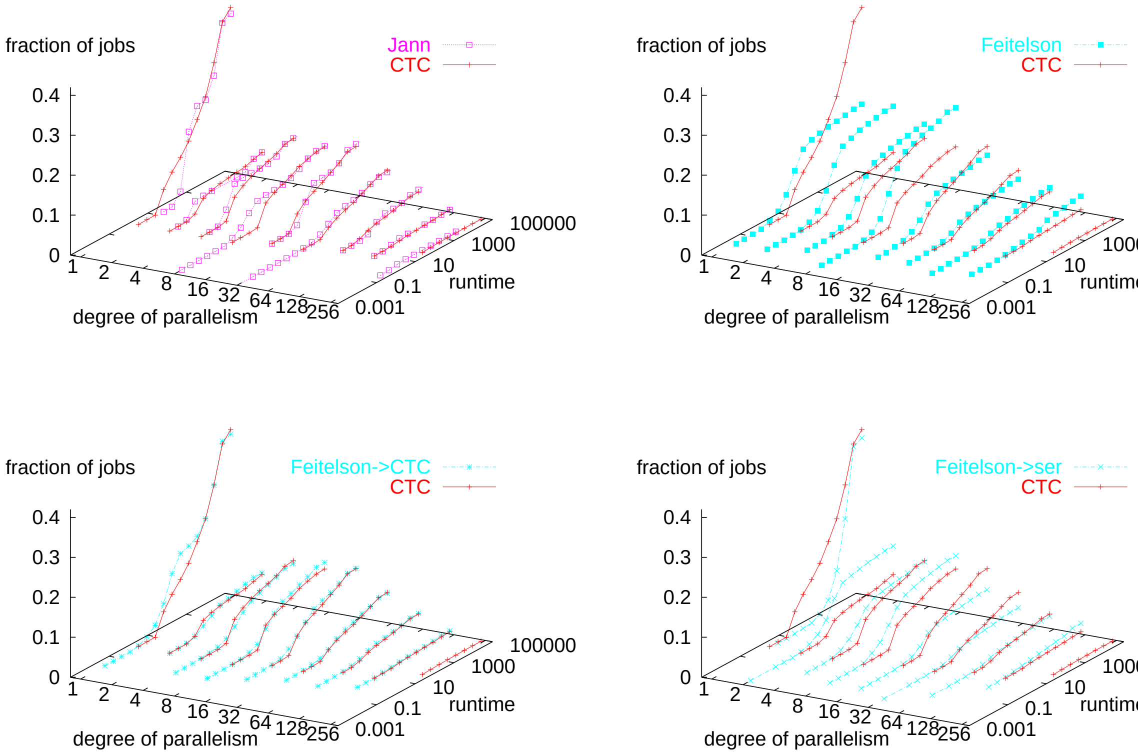


Figure 5: *Detailed comparison of different workloads, using the CTC workload as a reference. The second modification of the Feitelson model only affects serial jobs, and does not in general resemble the CTC or Jann workloads.*

performance differences we saw emanate specifically from the backfilling of long serial jobs, and the subsequent delaying of larger short jobs. In the Feitelson workload, in contrast, serial jobs tend to be shorter than larger ones.

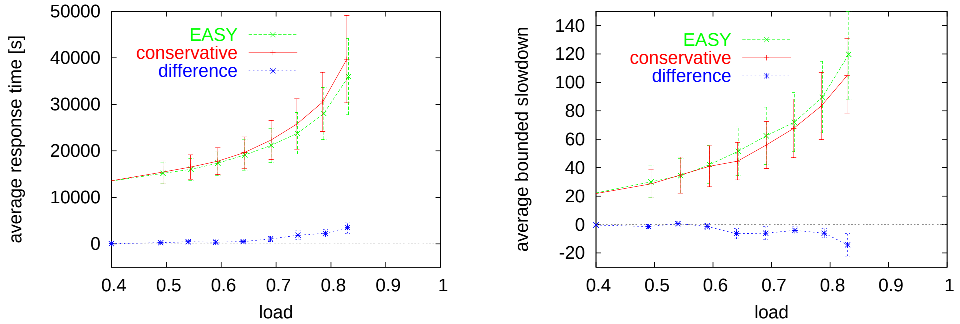
To verify this conjecture, we modified the Feitelson workload and re-ran the simulations. Part of this has been done in the past, as reported in [13]. That paper suggested modifying the following two attributes of the Feitelson model (Figure 5 “Feitelson→CTC”):

- Modify the distribution of job sizes to emphasize small jobs, and
- Modify the distribution of runtimes to emphasize longer jobs.

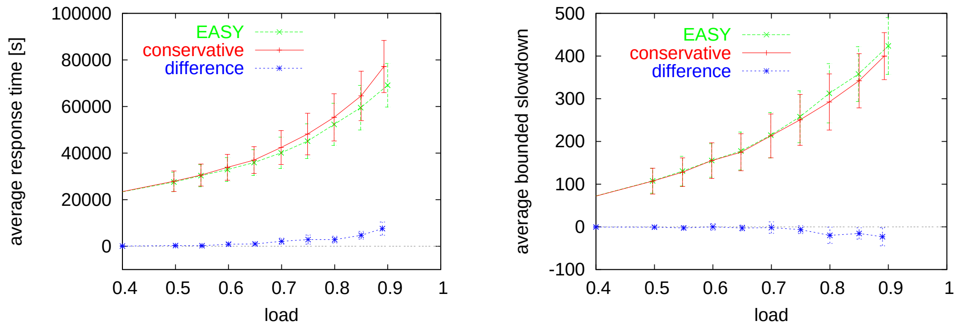
The details of implementing these modifications were hand tailored to create a workload that closely resembled both the CTC and Jann workloads. In particular, the distribution of sizes emphasized serial jobs and jobs in the range of sizes from 8 to 31, and serial jobs also received special treatment in terms of making them longer. Note that this is at odds with the original Feitelson model, in which a weak correlation exists between job size and runtime [4]. Simulating the behavior of the EASY and conservative schedulers on this modified Feitelson workload indeed showed behavior similar to that of the Jann workload (Figure 6 top). However, the magnitude of the effect was smaller.

Based on our current understanding of the matter, we can suggest a simpler modification:

Feitelson workload modified to resemble Jann/CTC (Feitelson→CTC)



Feitelson workload modified to include numerous long serial jobs (Feitelson→ser)



Feitelson workload modified to include numerous long serial jobs, using 139 nodes

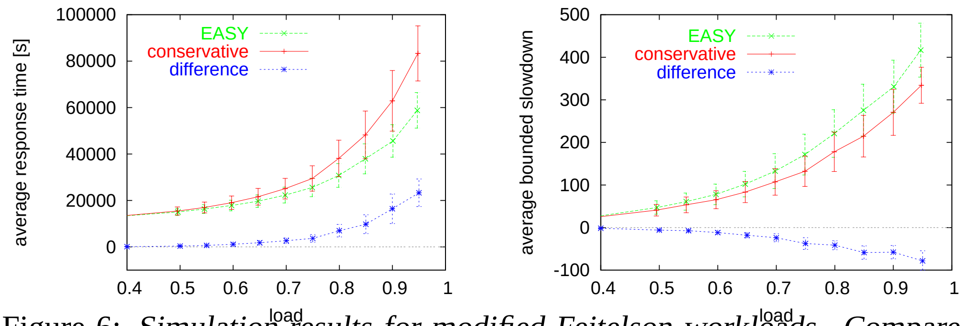


Figure 6: *Simulation results for modified Feitelson workloads. Compare with results shown in Figure 1.*

- Specifically create many long serial jobs.

This leads to a model that is quite different from the Jann and CTC workloads, because only the serial jobs have been changed (Figure 5 "Feitelson→ser"). However, making this change and re-running the simulations produced results that are quite similar to the previous, more elaborate modified model (Figure 6 middle). Thus we can indeed claim to have identified the crucial aspect of the workload that is needed in order to create the distinctive results seen with the Jann and CTC workloads — the presence of long serial jobs.

However, the results are still much less pronounced than those obtained from the Jann workload. Given that the original modification of the Feitelson workload (Feitelson→CTC) is extremely similar to the CTC and Jann workloads, the explanation for this is not the workload statistics. Rather, it is the size of the machine used in the simulation: using 139 nodes instead of 128 significantly increases the gap in performance observed between EASY and conservative (Figure 6 bottom). The reason for this is that most job sizes are powers of two, so using a machine size that is not a power of two leads to more idle processors and more options for backfilling. Both the CTC and Jann workloads specify non-power-of-2 sizes.

5 Conclusions

Simulations of the relative performance of EASY and conservative backfilling, using different workloads and metrics, exhibit discrepant results. However, detailed analysis of the specific circumstances of each simulation allows us to elucidate the following general conclusions:

- If the workload does not contain many long serial jobs, both backfilling policies lead to similar performance results.
- If the workload does indeed contain many long serial jobs, as was the case at CTC, and to a lesser degree at SDSC, the relative performance depends on the accuracy of user runtime estimates and on the number of nodes in the system.
  - If user runtime estimates are inaccurate, the EASY policy leads to better results. This is expected to be the more common case.
  - But if user runtime estimates are highly accurate, conservative backfilling degrades the performance of the long serial jobs and enhances the performance of larger short jobs. This leads to better overall slowdown results.
  - This effect is intensified when the number of nodes is not a power of two.

Apart for settling the issue of the relative merits of EASY and conservative backfilling, this work also underscores the importance of workload models and their details. All the workloads we investigated have been accepted as reasonable and representative by researchers who have used them in the past. But using different workloads can lead to completely different results, none of which are universally correct. If this happens, one should find the workload features that lead to the performance differences. This then enables the prediction of performance results for new workload conditions.

In addition to different workloads, seemingly benign assumptions can also have a decisive impact on evaluation results. In our case, the assumption of accurate runtime estimates, which is actually unlikely to be true [13, 9], was now shown to be instrumental in the results obtained using the Jann workload. This underscores the need to collect real data that can serve as the basis for performance evaluation, reducing the risk of using unbased assumptions.

Acknowledgments

This research was supported in part by the Israel Science Foundation (grants no. 219/99 and 167/03). Many thanks are due to the people who deposited their workload logs and models in the Parallel Workloads Archive, specifically the Cornell Theory Center, the HPC Systems group of the San Diego Supercomputer Center, and Joefon Jann.

References

[1] S-H. Chiang, A. Arpaci-Dusseau, and M. K. Vernon, "*The impact of more accurate requested runtimes on production job scheduling performance*". In *Job Scheduling Strategies for Parallel Processing*, pp. 103–127, Springer Verlag, 2002. Lect. Notes Comput. Sci. vol. 2537.

[2] W. Cirne and F. Berman, "*A model for moldable supercomputer jobs*". In 15th *Intl. Parallel & Distributed Processing Symp.*, Apr 2001.

[3] D. G. Feitelson, "*Metric and workload effects on computer systems evaluation*". *Computer* **36(9)**, pp. 18–25, Sep 2003.

[4] D. G. Feitelson, "*Packing schemes for gang scheduling*". In *Job Scheduling Strategies for Parallel Processing*, pp. 89–110, Springer-Verlag, 1996. Lect. Notes Comput. Sci. vol. 1162.

[5] D. G. Feitelson, L. Rudolph, U. Schwiegelshohn, K. C. Sevcik, and P. Wong, "*Theory and practice in parallel job scheduling*". In *Job Scheduling Strategies for Parallel Processing*, pp. 1–34, Springer Verlag, 1997. Lect. Notes Comput. Sci. vol. 1291.

[6] R. Jain, *The Art of Computer Systems Performance Analysis*. John Wiley & Sons, 1991.

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

[8] A. M. Law and W. D. Kelton, *Simulation Modeling and Analysis*. McGraw Hill, 3rd ed., 2000.

[9] C. B. Lee, Y. schwartzman, J. Hardy, and A. Snavely, "*Are user runtime estimates inherently inaccurate?*". In *Job Scheduling Strategies for Parallel Processing*, Springer-Verlag, 2004. Lect. Notes Comput. Sci. vol. 3277.

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

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

[12] M. H. MacDougall, *Simulating Computer Systems: Techniques and Tools*. MIT Press, 1987.

[13] A. W. Mu'alem and D. G. Feitelson, "*Utilization, predictability, workloads, and user runtime estimates in scheduling the IBM SP2 with backfilling*". *IEEE Trans. Parallel & Distributed Syst.* **12(6)**, pp. 529–543, Jun 2001.

[14] S. Srinivasan, R. Kettimuthu, V. Subramani, and P. Sadayappan, "*Selective reservation strategies for backfill job scheduling*". In *Job Scheduling Strategies for Parallel Processing*, pp. 55–71, Springer-Verlag, 2002. Lect. Notes Comput. Sci. vol. 2537.

[15] D. Tsafrir and D. G. Feitelson, *Workload Flurries*. Technical Report 2003-85, Hebrew University, Nov 2003.

[16] D. Zotkin and P. J. Keleher, "*Job-length estimation and performance in backfilling schedulers*". In 8th *Intl. Symp. High Performance Distributed Comput.*, Aug 1999.