Self-Adaptive Scheduler Parameterization via Online Simulation*

Barry Lawson
Department of Math and Computer Science
University of Richmond
Richmond, VA 23173, USA
blawson@richmond.edu

Evgenia Smirni
Department of Computer Science
College of William and Mary
Williamsburg, VA 23187-8795, USA
esmirni@cs.wm.edu

Abstract

High-end parallel systems present a tremendous research challenge on how to best allocate their resources to match dynamic workload characteristics and user habits that are often unique to each system. Although thoroughly investigated, job scheduling for production systems remains an inexact science, requiring significant experience and intuition from system administrators to properly configure batch schedulers. State-of-the-art schedulers provide many parameters for their configuration, but tuning these to optimize performance and to appropriately respond to the continuously varying characteristics of the workloads can be very difficult — the effects of different parameters and their interactions are often unintuitive.

In this paper, we introduce a new and general methodology for automating the difficult process of job scheduler parameterization. Our proposed methodology is based on online simulations of a model of the actual system to provide on-the-fly suggestions to the scheduler for automated parameter adjustment. Detailed performance comparisons via simulation using actual supercomputing traces from the Parallel Workloads Archive indicate that this self-adaptive parameterization via online simulation consistently outperforms other workload-aware methods for scheduler parameterization. This methodology is unique, flexible, and practical in that it requires no **a priori** *knowledge of the workload, it works well even in the presence of poor user runtime estimates, and it can be used to address any system statistic of interest.*

* This work was partially supported by the National Science Foundation under grants CCR-0098278, and ACI-0090221.

Keywords: *batch scheduler parameterization, high-end parallel systems, self-adaptive schedulers, backfilling, performance analysis, online simulation.*

1. Introduction

Large-scale clusters of multiprocessors have emerged as the dominant platform for high-end computing, comprising almost 60% of the Top 500 list (www.top500.org) as of June 2004. Traditional scheduling policies for clusters and other high-end distributed memory systems consider scheduling a single resource, i.e., CPU only, and focus on treating differently interactive versus batch jobs [1] with the goal of maximizing system utilization. Industrial-strength schedulers that are widely accepted by the supercomputing community, including the Maui scheduler [10], PBS [12], and IBM LoadLeveler [4], offer a variety of configuration parameters that allow the system administrator to customize the scheduling policy according to the site's needs. The effectiveness of the scheduler is directly determined by the administrator's choices for the configuration parameters. However, tuning these parameters to optimize performance is extremely challenging because it is difficult to predict the outcome based on the interaction of many different changing parameters. Choosing appropriate scheduling parameters in such complex environments requires not only awareness of the performance effects of existing queuing in the system, but also requires a keen understanding of the effects of transient behavior in the workload. Static definition of scheduling parameters is clearly insufficient to cope with sudden changes in the workload intensities and demands.

In the literature, scheduling policies based on backfilling have been proposed as more efficient alternatives to simple FCFS scheduling [11]. In backfilling, users are expected to provide nearly accurate estimates of the job execution times. Using these estimates, the scheduler scans and reorders the waiting queue, allowing certain short jobs to surpass long jobs in the queue provided those short jobs do not delay certain previously submitted jobs. The goal of backfilling is to decrease system fragmentation and increase system utilization [11, 14] by using otherwise idle processors for immediate execution. Various versions of backfilling have been proposed [6, 11, 13], including algorithms for gang-scheduling [15], and backfilling can be found in most contemporary cluster installation schedulers [2, 5]. For a recent survey of parallel job scheduling in general, including backfilling in particular, we direct the interested reader to [2].

The two important external parameters that affect performance and scheduling decisions in a parallel system are the arrival process and the service process. Variations in the intensities of arrivals and service requirements are responsible for the growth of waiting queues to a certain point beyond the "knee of the curve", i.e., the point where the individual response times of the jobs increase dramatically and the system operates in saturation. Dramatic changes in the resource demands are consistent across actual workloads [7, 8], as evidenced by very skewed run times within each workload, by significant variability in the average "width" of each job, i.e., the number of per-job requested processors, and by very inconsistent job arrival rates across time.

Based on these observations, the authors in [7, 8] proposed a multiple-queue backfilling scheduling policy for homogeneous systems that allows the scheduler to swiftly change certain configuration parameters according to changes in the incoming workload. This multiple-queue policy splits the system into multiple partitions (the number of partitions is fixed), with one queue per partition, and *separates* short from long jobs by assigning incoming jobs to different queues based on user-provided job runtime estimates. Initially, processors are distributed evenly among the partitions, but as time evolves, processors may move from one partition to another so that processors currently idle in one partition may be used for immediate backfilling in another. Hence, the system is able to adjust on-the-fly according to changes in the incoming workload, and, in this manner, the average job slowdown is reduced by diminishing the likelihood that a short job is delayed behind a long job.

However, the solution presented in [8] suffers from two major drawbacks. First, the choice for the fixed number of partitions is not obvious, and it must be determined **a priori**. Even when given a number of partitions, the selection of static criteria for assigning jobs to partitions based on runtime estimates is also not obvious and must be done **a priori**. The observed inconsistencies across time for the arrival patterns and service demands in workloads imply that such *a priori* and *static* selection of the number of partitions and partitioning criteria cannot guarantee good performance across the lifetime of the workload.

Second, prerequisite to the classification of jobs to partitions is the availability of accurate job runtime estimates. User-provided runtime estimates are notoriously inaccurate [9, 11]. Users tend to "overestimate" expected execution time of jobs to avoid having the job terminated by the scheduler due to a short estimate. Furthermore, if a job crashes (which often happens very early in its execution), there will likely be a *significant* discrepancy in the estimated and actual execution time. Such inaccurate estimates often cause jobs to be placed into inappropriate partitions, diminishing system performance [8].

This paper addresses the above shortcomings of multiple-queue backfilling. The stated goals and outline of this work are:

- To present a methodology for automatic recalculation of job partitioning criteria, given a fixed number of queues (see Section 2). This methodology is based on observation of the past workload behavior to predict future workload behavior.
- To present a *general* and *practical* methodology for automating scheduler parameterization. The methodology uses online simulation to select the ideal number of queues on the fly (see Section 3). We further show that online simulation allows the system to recover from incorrect (and possibly unavoidable) decisions.
- To summarize our contributions and outline future work (see Section 4).

2. Multiple-Queue Backfilling: Solutions and Shortcomings

In [7], workload analysis using *actual* runtimes from traces in the Parallel Workloads Archive showed that using four queues provides the best separation of jobs to reduce the waiting time of jobs. The motivation for using four queues was given after examining the statistical

characteristics of workload traces available in the Parallel Workloads Archive. For more details on trace workload analysis, we direct the interested reader to [7].

In Figure 1, we provide a representative sample of the resource demands of two actual supercomputer traces, the first, labeled KTH, from a 100-node SP2 at the Swedish Royal Institute of Technology, and the second, labeled SDSC-SP2, from the 128-node IBM SP2 at the San Diego Supercomputer Center. Figures 1(a)–1(f) illustrate the time evolution of the arrival process by presenting the total number of arriving jobs per week, the time evolution of the service process, and its coefficient of variation (C.V.). We observe significant variability in the job arrival and service processes, which indicates the importance of not only *aggregate* statistics (i.e., the average performance measures obtained after simulating the system using the entire workload trace), but also of *transient* statistics within small windows of time.

Figures 1(c)–1(f) also show the mean service times and their respective C.V.s if jobs are classified according to their duration (dashed lines). We observe very skewed run times within each workload and significant variability in their C.V.s, which significantly reduce if jobs are classified according to their duration. This classification also aims at balancing the length of each of the four queues by keeping the number of jobs per week assigned to each queue nearly the same. Figures 1(g)–1(h) shows that despite the fact that we have **a priori** knowledge of the workload demands, this *static* classification does not result in a well-balanced system across time as the target of maintaining roughly 25% of the number of jobs per class, is not reached across all weeks. The multiple-queue backfilling policy that was developed in [7] is based on this classification, and experiments indicate that a four-queue classification is promising across most of the workload traces from the Parallel Workloads Archive, but no clear answer is given as to what the *ideal number* of queues should be. Furthermore, given a predefined ideal number of queues, no clear answer is given to the problem of how to best *partition* jobs into different classes in order to meet the needs of a dynamically changing workload. We address these two questions in the following sections.

2.1. The Multiple-Queue Backfilling Policy

The multiple-queue backfilling policy splits the system into multiple *disjoint* partitions, with one queue per partition. Initially, each partition is assigned an equal number of processors. As time evolves, processors idle in one partition can be used for backfilling in another partition. In this way, the partition boundaries are dynamic, allowing the system to adapt itself to fluctuating workload patterns. Furthermore, the policy does not starve a job that requires all processors for execution.

In general, the process of backfilling *exactly one* queued job (of possibly many queued jobs to be backfilled) proceeds as follows. Let $\mathcal{P}$ be the partition to which the job is assigned. Define $p_{\mathcal{P}}$ to be the *pivot* (i.e., the first job in the queue) in partition $\mathcal{P}$, and define $t_{\mathcal{P}}$ to be the time when $p_{\mathcal{P}}$ can begin executing. If the job under consideration is $p_{\mathcal{P}}$, it begins executing only if the current time is equal to $t_{\mathcal{P}}$, in which case a new $p_{\mathcal{P}}$ is defined. If the job is not $p_{\mathcal{P}}$, the job begins executing only if there are sufficient idle processors in partition $\mathcal{P}$ without delaying $p_{\mathcal{P}}$, or if partition $\mathcal{P}$ can obtain sufficient idle processors from one or more other partitions without delaying any pivot. This process of backfilling exactly one job is repeated, one job at a time, until all queued jobs have been considered. The multiple-queue backfilling policy, outlined in Figure 2, is executed whenever a job is submitted or whenever an executing job completes.

Although permitting processors to cross partition boundaries begins to address the issue of fluctuating workload patterns, the fixed criteria for defining job classes and assigning them to partitions, as given in [7, 8] (see legend in Figure 1), are too inflexible for a good general solution. These fixed criteria must be determined **a priori**, which is impossible in any real world scenario. A better approach, which we introduce here, is to dynamically recompute the partitioning criteria to better meet the needs of the transient arrival and service demands of evolving workloads.

To this end, at the start of each new week, we evaluate the set of jobs that completed in the previous week. Using this set of jobs only, we define new partitioning criteria by examining the estimated runtimes of the jobs, and from those estimated runtimes select appropriate criteria that would have balanced the number of jobs across all queues. The algorithm for determining the partitioning criteria for a new week is given in Figure 3. As described in the sections to follow, this algorithm provides the multiple-queue backfilling policy more flexibility to respond to workload fluctuations, yet alone it is not a sufficient general solution.

2.2. Experimental Methodology

The simulation experiments are driven by four workload traces from the Parallel Workloads Archive [3]:

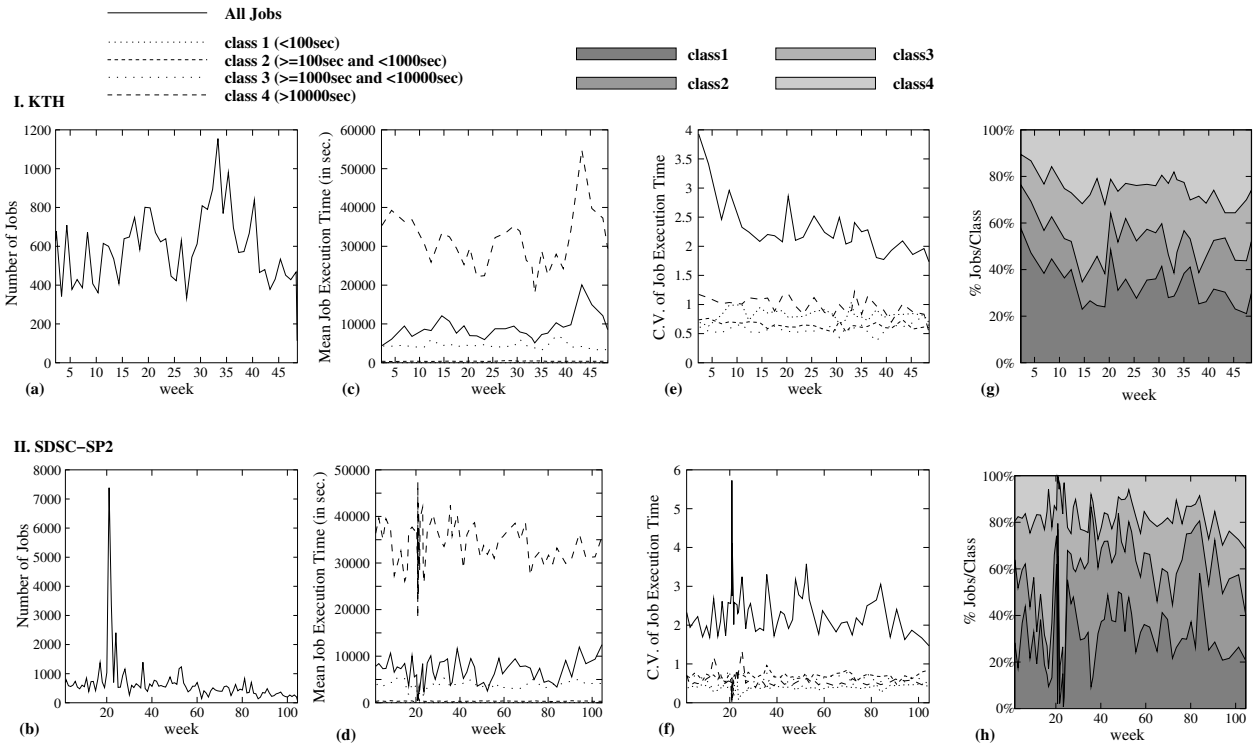

Figure 1. Total number of arriving jobs per week (a)–(b), mean job service time per week (c)–(d), coefficient of variation (C.V.) of job service times (e)–(f), and percentage of jobs per class (g)–(h). All graphs are presented as a function of time (weeks).

- **CTC**: log containing 79 302 jobs executed on a 512-node IBM SP2 at the Cornell Theory Center from July 1996 through May 1997;
- **KTH**: log containing 28 490 jobs executed on a 100-node IBM SP2 at the Swedish Royal Institute of Technology from October 1996 through August 1997;
- **SCDC-SP2**: log containing 73 496 jobs executed on a 128-node IBM SP2 at the San Diego Supercomputer Center from May 1998 through April 2000;
- **Blue Horizon**: log containing 250 440 jobs executed on a 144-node with 8 processors per node IBM SP at the San Diego Supercomputer Center from April 2000 through January 2003.

From the traces, for each job we extract the arrival time of the job (i.e., the submission time), the number of processors requested, the estimated duration of the job, and the actual duration of the job. Because we do not use job completion times from the traces, the scheduling strategies used on the corresponding systems are not relevant to our study.

We evaluate and compare via simulation the performance of multiple-queue backfilling relative to standard single-queue backfilling. We consider *aggregate measures*, i.e., average statistics computed using all jobs for the entire simulation run, and *transient measures*, i.e., per-week snapshot statistics that are plotted versus experiment time to illustrate how the policies react to sudden changes in the workload. The performance measure of interest here is each job's bounded slowdown[1] [11] defined by

$$s = 1 + \frac{d}{\max\{10, \nu\}}$$

where d and ν are respectively the queuing delay time and the actual service time of the job. To compare the multiple-queue policy with the single-queue policy, we

1 The maximizing function in the denominator reduces the otherwise dramatic impact that very small jobs can have on the slowdown statistic. The 10 second execution time parameter is standard in the literature[11].

for (all jobs in order of arrival)

1. $\mathcal{P} \longleftarrow$ partition in which job resides
2. $p_{\mathcal{P}} \longleftarrow$ pivot job (first job in the queue for $\mathcal{P}$)
3. $t_{\mathcal{P}} \longleftarrow$ earliest time when sufficient processors will be available for $p_{\mathcal{P}}$
4. $i_{\mathcal{P}} \longleftarrow$ currently idle processors in $\mathcal{P}$
5. $x_{\mathcal{P}} \longleftarrow$ idle processors in partition $\mathcal{P}$ at $t_{\mathcal{P}}$ not required by $p_{\mathcal{P}}$
6. if (job is $p_{\mathcal{P}}$)
  a. if (current time equals $t_{\mathcal{P}}$)
    I. if necessary, reassign processors from other partitions to $\mathcal{P}$
    II. start job immediately
7. else
  a. if (job requires $\leq i_{\mathcal{P}}$ and will finish by $t_{\mathcal{P}}$) start job immediately
  b. else if (job requires $\leq \min\{i_{\mathcal{P}}, x_{\mathcal{P}}\}$) start job immediately
  c. else if (job requires $\leq \{i_{\mathcal{P}}$ plus some combination of idle/extra processors in other partitions$\}$ such that no pivot is delayed)
    I. reassign necessary processors from other partitions to $\mathcal{P}$
    II. start job immediately

Figure 2. Multiple-queue backfilling algorithm

1. $P \longleftarrow$ current number of partitions in the system
2. $S \longleftarrow$ set of jobs that completed in the previous week
3. $F(t) \longleftarrow$ build the cumulative distribution function using runtime estimates t of jobs in S
4. for (i from 1 to P)
  find t_i such that $F(t_i) = i/P$
5. New week's partitioning boundaries for job with runtime estimate t is

$$\mathcal{P} = \begin{cases} 1, & 0 < t < t_1 \\ 2, & t_1 \leq t < t_2 \\ & \vdots \\ P, & t_{P-1} \leq t < t_P \end{cases}$$

Figure 3. Algorithm for automatically recalculating the job partitioning criteria in multiple-queue backfilling.

define the *slowdown ratio* $\mathcal{R}$ for a job by the equation

$$\mathcal{R} = \frac{s_1 - s_m}{\min\{s_1, s_m\}}$$

where s_1 and s_m are the bounded slowdowns computed for that job using respectively single-queue and multiple-queue backfilling. $\mathcal{R} > 0$ indicates an $\mathcal{R}$-fold gain in performance using the multiple-queue policy relative to a single queue. $\mathcal{R} < 0$ indicates an $\mathcal{R}$-fold loss in performance using the multiple-queue policy relative to a single queue. The aggregate and transient measures are computed as averages of job slowdown ratios.

Note that the use of the $\mathcal{R}$ metric is meaningful in the context of our goals. We compare individually the performance of each job under the two different policies. If one policy outperforms the other with respect to a given job, this will be reflected in the measurement — but a loss in performance relative to the other policy will also be reflected. Our goal is to improve the overall slowdown by reducing the queuing delay experienced by short jobs. Since the bounded slowdown metric is most affected by queuing delay for short jobs, a positive result for $\mathcal{R}$ indicates that, indeed, the multiple-queue policy is assisting the shorter jobs.

2.3. Multiple-Queue Backfilling: Strengths and Weaknesses

In this section, we present results showing that the multiple-queue backfilling policy outperforms the stan-

dard single-queue policy in the presence of exact estimates and, in most cases, in the presence of inexact user runtime estimates. For the figures to follow, we use the multiple-queue backfilling algorithm with four queues enhanced by automatic recalculation of the job partitioning criteria, but without any other modifications.

Figure 4 depicts the aggregate slowdown ratio $\mathcal{R}$ of multiple-queue backfilling relative to single-queue backfilling for each of the four workloads. Figure 4(a) provides an overall comparison of the relative performance of the two policies when exact runtimes are provided by the user and when these same exact runtimes are used for recomputing the job partitioning criteria each week. In other words, we assume that we have the ideal case, where the workload is known **a priori**. As shown by these aggregate performance measures, multiple-queue backfilling provides better overall job slowdown (i.e., $\mathcal{R} > 0$) for all four traces. There are, as one can expect, varying degrees of success from one workload to another, with SDCS-SP2 receiving the best performance gain while CTC receives the most modest gain. Nonetheless, the improvements for CTC and Blue Horizon using multiple-queue instead of single-queue backfilling are nearly two-fold.

Figure 4(b) provides an overall comparison of the relative performance of the two policies using inexact user estimates (i.e., runtime estimates provided in the workload traces), and when the runtime estimates of the previous week are used to recalculate the job partitioning criteria using the algorithm of Figure 3. The figure shows that partitioning using user estimates can negatively impact the performance of multiple-queue backfilling. User runtime estimates are notoriously poor [9, 11]. As a result, user estimates can lead to poor partitioning of jobs relative to their actual runtimes so that the effectiveness of the multiple-queue policy is reduced. Nonetheless, performance improvement relative to single-queue backfilling is achieved for three of the four traces, with improvement for SCSD-SP2 over that using *exact* estimates. Note that for CTC, however, multiple-queue backfilling with inexact estimates performs worse overall than single-queue backfilling.

We now turn to the transient performance behavior of multiple-queue backfilling. Figure 5 depicts transient one-week snapshots of the slowdown ratio versus time for each of the four traces. The dashed lines in the figure represent the improvement of multiple-queue backfilling over single-queue backfilling when using actual runtimes of that same week as exact estimates for job classification (i.e., **a priori** knowledge). Generally, marked improvement in job slowdown is achieved ($\mathcal{R} > 0$)

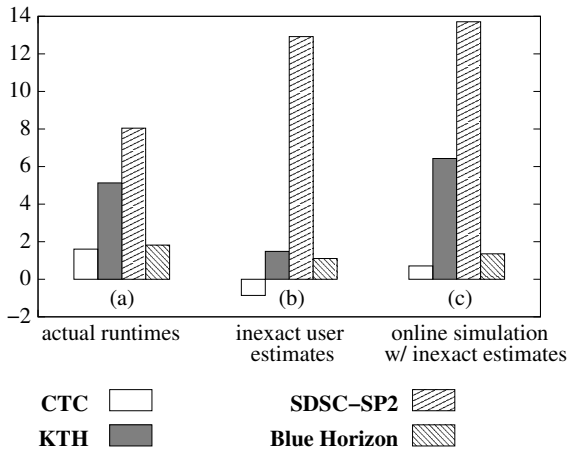

Figure 4. Aggregate slowdown ratio $\mathcal{R}$ for four traces using actual times as exact estimates, using inexact user estimates, and using online simulation.

by using multiple-queue backfilling. Although single-queue outperforms multiple-queue backfilling ($\mathcal{R} < 0$) for a select few of the weeks (e.g., week 11 for KTH, week 101 for Blue Horizon), $\mathcal{R}$ is positive for a majority of the weeks, corresponding to performance gains using multiple-queue backfilling.

The solid lines in Figure 5 represent the practical case when no workload is known **a priori**. In this case, user estimates of the previous week are used to recalculate the partitioning criteria, and each incoming job is allocated to a queue according to its estimated (inexact) runtime. In this context of inexact user estimates, transient analysis is consistent with the generally diminished performance shown in the aggregate results. Relative to the exact-estimate results (i.e., dashed lines), for three of the four traces there is an increase in the number of negative peaks ($\mathcal{R} < 0$) and the magnitude of the positive peaks is diminished. This trend is most notable for CTC, with few prominent positive peaks and several prominent negative peaks (e.g., weeks 30 and 40).

In short, our results show that multiple-queue backfilling with automatic recalculation of partitioning criteria provides dramatic improvement in slowdown relative to single-queue backfilling when exact runtime estimates are known **a priori**. However, in the more realistic setting in which runtimes estimates are *not* known, and are often very inaccurate, the performance benefit of multiple-queue backfilling can diminish or can even result in a performance loss. These observations lead us to consider an improved, more flexible scheduling pol-

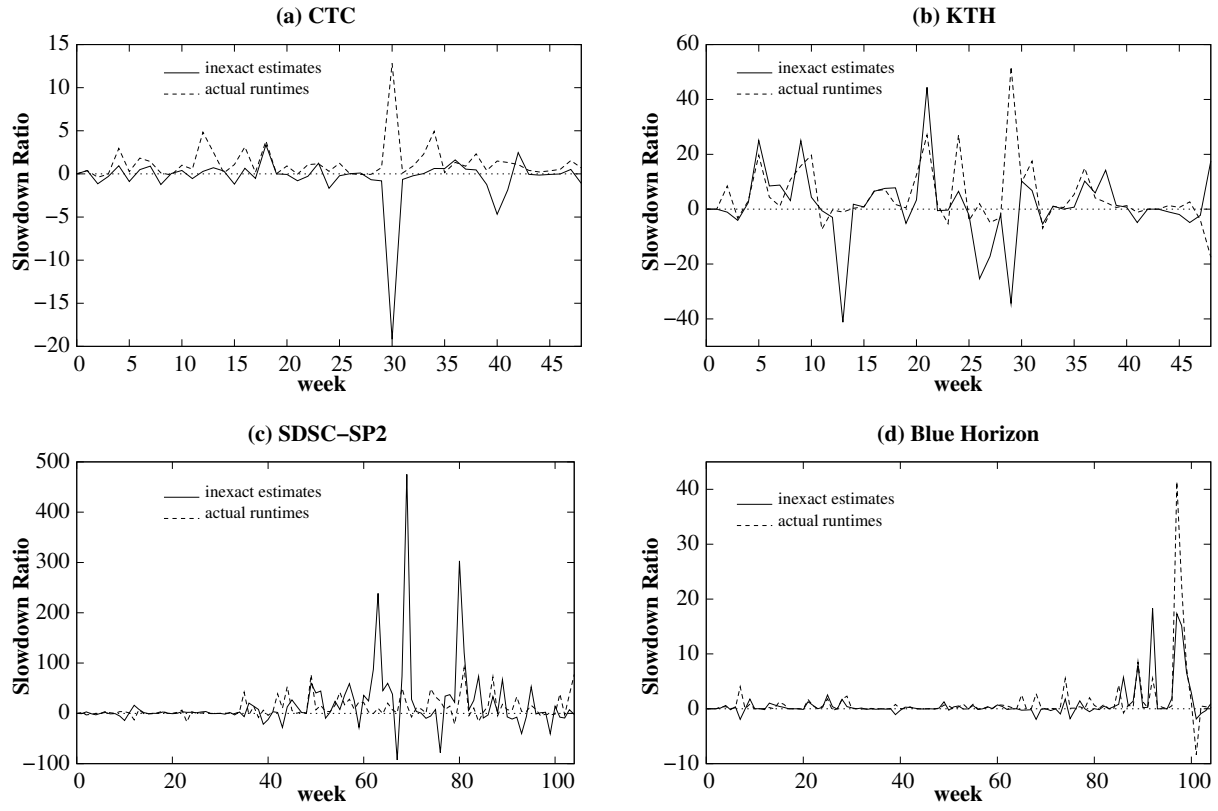

Figure 5. Slowdown ratio $\mathcal{R}$ per week as a function of time for each of the four traces. For dashed lines, actual runtimes were used with a priori knowledge of the workload. For solid lines, inexact user estimates were used with *no* a priori knowledge.

icy discussed in the following sections.

3. Online Simulation Policy

The results in the previous section show that in most instances the multiple-queue policy outperforms the single-queue policy overall, but there are instances where the single-queue policy does perform better, especially (in the practical case) when only inexact user runtime estimates are available. Although with estimates the algorithm looks at the past (previous week) to predict the future (current week) for determining the partitioning criteria, experiments with *different* numbers of queues (e.g., 2, 3, and 8) have also shown that there is no universal number of queues that is ideal across all weeks for each workload. (Due to space restrictions, we do not present those results here.) For some weeks, two queues may perform better than three, but for some others four queues perform best. Based on this observation, we introduce a new scheduling policy here that automates changing the number of partitions on the fly to address transient workload fluctuations. The policy is based on the ability to execute lightweight simulation modules in an *online* fashion.

More specifically, self-adaptive parameterization based on online simulation can be described as follows. Given a current state of the system and given a set of jobs that are waiting for service in the queue, we simulate a different scheduling policy (as defined by a different number of partitions for multiple-queue backfilling with automatic recalculation of partitioning criteria). In this regard, it should be noted that single-queue backfilling is a special case of the more general multiple-

queue backfilling with $P = 1$. **FIX: The main idea is to run a quick trace-driven simulation for the waiting jobs without affecting the operation of the actual system.** The ability to run and compare the performance of several such simulations (each modeling a different policy) within a very short time can help provide educated changes in the policy employed by the scheduler to better serve the waiting jobs. These simulations are triggered whenever a performance measurement (here, bounded slowdown) exceeds a pre-specified threshold. The multiple simulations are executed online but with the initial state of the simulation *exactly* the same as the state of the real system. The number of partitions that provides the best simulated performance is then chosen as the (perhaps new) number of partitions in the real system, with appropriate reconfiguration (if necessary) of the real system to incorporate this number of partitions. In this fashion, we anticipate to reach faster system recovery from "wrong" scheduling decisions (because of inexact user estimates that direct jobs to the wrong queues) that result in a substantial increase in the waiting queue (and consequently significant increase in the average job slowdown).

3.1. Policy Parameterization

More specifically, for the online simulation policy, define the following parameters.

- $\mathcal{T}$: the pre-specified bounded slowdown threshold that triggers online simulation; and
- $\mathcal{L}$: the length of time (weeks) to simulate.

Let P be the current number of partitions in the system, and let P_{max} be the maximum number of partitions allowed in the system. When the aggregate bounded slowdown in the system exceeds $\mathcal{T}$, P_{max} online simulations are executed, using multiple-queue backfilling with automatic recalculation of partitioning criteria, with $1, 2, \ldots, P_{\text{max}}$ partitions respectively. The simulation with the best aggregate bounded slowdown for the simulated period $\mathcal{L}$ determines the number of partitions to be used in the actual system.

We stress that for each experiment we must start the online simulation in *exactly* the same state as the real system, but with a different number of partitions. If the number of partitions is different, we keep the same collection of queued jobs that are present in the real system, but we must partition those jobs differently based on the new partitioning criteria that are adjusted for the new number of partitions. For more details on this and other implementation particulars, we direct the reader to Appendix A.

3.2. Online Simulation Policy Performance

In this section, we present results showing that the online simulation policy, using multiple-queue backfilling with automatic recalculation of partitioning criteria, performs well even in the presence of inexact user estimates. That is, unlike certain instances for multiple-queue backfilling with automatic recalculation of partitioning criteria alone, the online simulation policy is able to perform well even when there is *no* **a priori** knowledge of the workload.

For Figures 4(c) and 6, we have used the online simulation policy parameters $(\mathcal{T}, \mathcal{L})$ for each trace as follows: $(2.5, 2)$ for CTC; $(100, 2)$ for KTH; $(25, 1)$ for SDSC-SP2; and $(10, 1)$ for Blue Horizon.[2] Figure 4 depicts for each of the four workloads the aggregate slowdown ratio $\mathcal{R}$ relative to single-queue backfilling for (a) multiple-queue backfilling using actual runtimes with **a priori** knowledge, (b) multiple-queue backfilling using inexact user estimates and no **a priori** knowledge, and (c) online simulation using inexact user estimates and no **a priori** knowledge. Note that, with each policy in the presence of inexact user estimates, online simulation outperforms multiple-queue backfilling for each of the four traces (compare (b) and (c)). Furthermore, online simulation with inexact user estimates performs nearly as well as multiple-queue backfilling with actual runtimes (i.e., **a priori** knowledge) for CTC and Blue Horizon, and even better for KTH and SDSC-SP2 (compare (a) and (c)). Even though in an actual scheduling context we can not know exact runtimes **a priori**, our results suggest that with online simulation we can perform nearly as well as, or even better than, having **a priori** knowledge of the workload.

Figure 6 depicts transient one-week snapshots of the slowdown ratio $\mathcal{R}$ versus time using online simulation with inexact user estimates. The solid lines represent the results from using online simulation and no **a priori** knowledge of the workload; the dashed lines represent the results from using multiple-queue backfilling assuming **a priori** knowledge of the workload, and are the same as the dashed lines in Figure 5. Again $\mathcal{R} > 0$ indicates a performance gain from using online simulation relative to single-queue backfilling. For the solid

2 We empirically selected these parameters by comparing results from simulation runs for each trace with $\mathcal{T} \in \{2.5, 5, 7.5, 10, 25, 50, 75, 100\}$ and $\mathcal{L} \in \{1, 2\}$.

lines, $\mathcal{R}$ is positive for a majority of the weeks, and the number of negative spikes prevalent in Figure 5 is dramatically reduced, most specifically for CTC. Also notice that online simulation corrects the problems that multiple-queue backfilling experiences with inexact estimates.

We note that online simulation cannot be used indiscriminately. Instead, the online simulation policy parameters $\mathcal{T}$ and $\mathcal{L}$ should be tuned to match the characteristics of a specific system. Fortunately, for online simulation there are few parameters that a system administrator must set. As suggested by the previous figures, appropriate parameters can lead to very good performance even in the presence of inexact runtime estimates. However, inappropriately chosen parameters *can* (but does not often) lead to a loss in performance with respect to single-queue backfilling. Our experiments showed that, in general, even if the parameters are chosen indiscriminately, the online simulation policy will improve the performance of the system.

4. Concluding Remarks

We have presented a first step toward automating the difficult process of job scheduler parameterization in high-end parallel systems. Detailed simulation experiments using actual supercomputer traces from the Parallel Workloads Archive strongly suggest that self-adaptive policies are needed to address the observed variability in workloads to improve policy performance. We have presented a policy that is based on online execution of lightweight simulation modules, each modeling a different scheduling policy. This online simulation approach proves effective in modeling scheduling micro-scenarios and departs from earlier work in that it strives for online tuning of scheduling parameters, thus stressing speed in addition to accuracy. Speed allows for executing these alternative scheduling scenarios while the system is in operation, and for making conclusions on possible scheduler parameter reconfiguration if required. Our experiments indicate that this online methodology does allow for quick system recovery from earlier incorrect scheduling decisions, resulting in schedulers that manage to adapt their parameters to the transient nature of the workload.

In this paper we showed the effectiveness of online simulation when used to define the best parameters for performance in multiple-queue backfilling policies. Our future work will concentrate on developing lightweight simulation modules that can be easily parameterized by the system administrator. More specifically, the system administrator may choose to optimize another measure than average job slowdown, or may define policies that are not based on multiple-queue backfilling. Our target is to provide a library of lightweight simulation modules which the system administrator can use to define different policies and explore online their performance. Finally, we will provide simulation modules that will allow the modeling of policies that provide priorities and reservations to jobs, i.e., the schedule guarantees that a job completes by a certain time, or a job starts execution at a specific time.

Acknowledgments

We thank Tom Crockett for his feedback in earlier versions of this work. We also thank Dror Feitelson for maintaining the Parallel Workloads Archive; Dan Dwyer and Steve Hotovy for the CTC workload; Lars Malinowsky for the KTH workload; Victor Hazlewood for the SDSC-SP2 workload; and Travis Earheart and Nancy Wilkins-Diehr for the Blue Horizon workload.

References

[1] D. G. Feitelson. A survey of scheduling in multiprogrammed parallel systems. Technical Report RC 19790, IBM Research Division, October 1994.

[2] D. G. Feitelson, L. Rudolph, and U. Schwiegelshohn. Parallel job scheduling — a status report. In *Proceedings of the Workshop on Job Scheduling Strategies for Parallel Processing*. Springer-Verlag, 2004. In press.

[3] Parallel Workload Archive. `www.cs.huji.ac.il/labs/parallel/workload`.

[4] IBM LoadLeveler. `www-1.ibm.com/servers/eserver/ecatalog/us/software/4051.html`.

[5] D. Jackson, Q. Snell, and M. Clement. Core algorithms of the Maui scheduler. In *Proceedings of the Workshop on Job Scheduling Strategies for Parallel Processing*. Springer-Verlag, 2001.

[6] P. Keleher, D. Zotkin, and D. Perkovic. Attacking the bottlenecks in backfilling schedulers. *Cluster Computing: The Journal of Networks, Software Tools and Applications*, 3(4):245–254, 2000.

[7] B. Lawson, E. Smirni, and D. Puiu. Self-adapting backfilling scheduling for parallel systems. In *Proceedings of the 2002 International Conference on Parallel Processing (ICPP 2002)*, pages 583–592, Vancouver, B.C., August 2002.

[8] B. G. Lawson and E. Smirni. Multiple-queue backfilling scheduling with priorities and reservations for parallel systems. In D. G. Feitelson, L. Rudolph, and U. Schwiegelshohn, editors, *Job Scheduling Strategies*

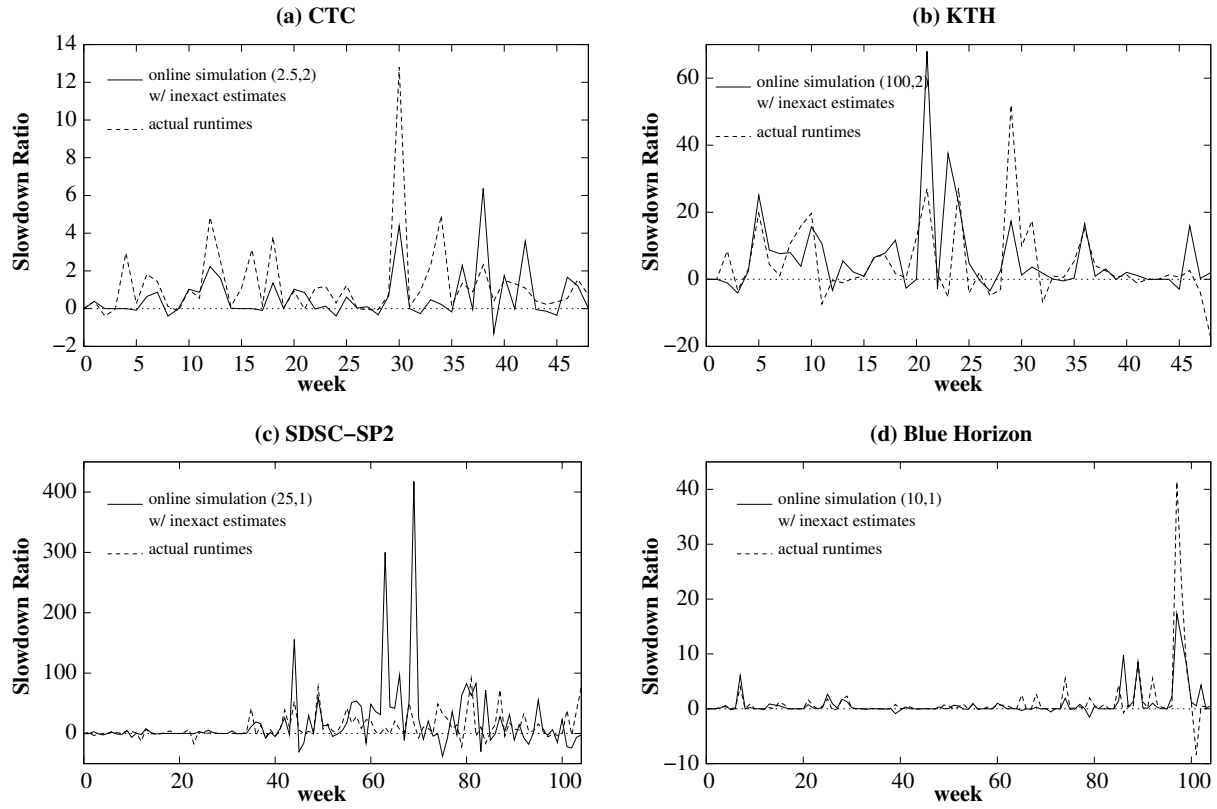

Figure 6. Slowdown ratio $\mathcal{R}$ per week as a function of time for each of the four traces using online simulation. For solid lines, online simulation was used with inexact user estimates (no a priori knowledge of the workload). For dashed lines, multiple-queue backfilling was used with exact user estimates (assuming a priori knowledge).

for Parallel Processing, pages 72–87. Springer Verlag, 2002. Lect. Notes Comput. Sci. vol. 2537.

[9] C. B. Lee, Y. Schwartzman, J. Hardy, and A. Snavely. Are user runtime estimates inherently inaccurate? In *Proceedings of the Workshop on Job Scheduling Strategies for Parallel Processing*. Springer-Verlag, 2004. In press.

[10] Maui Scheduler Open Cluster Software. `mauischeduler.sourceforge.net`.

[11] A. Mualem and D. G. Feitelson. Utilization, predictability, workloads, and user runtime estimates in scheduling the IBM SP2 with backfilling. *IEEE Transactions on Parallel and Distributed Systems*, 12(6):529–543, June 2001.

[12] Portable Batch System. `www.openpbs.org`.

[13] D. Perkovic and P. Keleher. Randomization, speculation, and adaptation in batch schedulers. In *Proceedings of Supercomputing 2000 (SC2000)*, November 2000.

[14] D. Talby and D. Feitelson. Supporting priorities and improving utilization of the IBM SP2 scheduler using slack-based backfilling. In *Proceedings of the 13th International Parallel Processing Symposium*, pages 513–517, April 1999.

[15] Y. Zhang, H. Franke, J. Moreira, and A. Sivasubramaniam. An integrated approach to parallel scheduling using gang-scheduling, backfilling, and migration. *IEEE Trans. Parallel Distrib. Syst.*, 14(3):236–247, 2003.