Analyzing and Adjusting User Runtime Estimates to Improve Job Scheduling on the Blue Gene/P

Wei Tang,* Narayan Desai,† Daniel Buettner,‡ Zhiling Lan*

*Department of Computer Science, Illinois Institute of Technology
Chicago, IL 60616, USA
{wtang6, lan}@iit.edu

†Mathematics and Computer Science Division
‡Argonne Leadership Computing Facility
Argonne National Laboratory, Argonne, IL 60439, USA
† desai@mcs.anl.gov
‡ buettner@alcf.anl.gov

***Abstract*—Backfilling and short-job-first are widely acknowledged enhancements to the simple but popular first-come, first-served job scheduling policy. However, both enhancements depend on user-provided estimates of job runtime, which research has repeatedly shown to be inaccurate. We have investigated the effects of this inaccuracy on backfilling and different queue prioritization policies, determining which part of the scheduling policy is most sensitive. Using these results, we have designed and implemented several estimation-adjusting schemes based on historical data. We have evaluated these schemes using workload traces from the Blue Gene/P system at Argonne National Laboratory. Our experimental results demonstrate that dynamically adjusting job runtime estimates can improve job scheduling performance by up to 20%.**

***Keywords*-job scheduling; runtime estimates; Blue Gene**

I. Introduction

Job scheduling is a critical task on large-scale systems, where small differences in scheduling policies can result in significant variation in system performance. Although various scheduling policies have been designed, first-come, first-served (FCFS) is still the prevalent default scheduling policy used in batch schedulers [10]. FCFS is straightforward, easy to implement, and fair. However, it suffers from poor utilization when idle resources cannot meet the needs of the job at the top of the queue. Moreover, a short job may be left waiting for a much longer job to finish, thus deteriorating the average waiting time.

The addition of backfilling partially addresses this issue. It allows lower-priority jobs to run earlier if they do not delay higher-priority jobs. If a low-priority job has a walltime smaller than the time needed to drain resources for the next job, it is executed out of order. While this optimization is simple, it dramatically improves resource utilization [16].

Another enhancement for average waiting time is job reordering, allowing a late-arriving job to run before one that was submitted earlier. "Largest expansion factor first" is a combination of FCFS and "shortest job first," considering both job arrival order and job length, where expansion factor is the ratio of the job's wait time plus requested runtime to the requested runtime [6]. A variant of this approach, called WFP [2][26], is used for production scheduling on the Blue Gene/P system at Argonne. WFP favors large and old jobs, adjusting their priorities based on the ratio of wait time to their requested wall clock times.

Both backfilling and order relaxation schemes depend on user-reported estimates of job runtimes. Backfilling considers the runtimes of currently running jobs as well as the length of candidates, for backfilling to ensure that response time for high-priority jobs is minimally impacted. For relaxed FCFS, estimated job runtimes are used to calculate job priorities. For this reason, these approaches are highly dependent on the accuracy of user estimates of job runtimes, which have been repeatedly demonstrated to be highly inaccurate [7][29][5].

Does the inaccuracy of user runtime estimates impact job scheduling? In other words, will more accurate user runtime estimates improve performance? A number of studies have addressed the problem. Some have shown that the inaccuracy of runtime estimation barely degrades performance [20]. Such results have led to the suggestion that estimates should be doubled [31] or randomized [21] to make them even less accurate. However, others have shown that using more accurate estimated runtimes can improve system performance far more significantly than previously suggested [5][25][27].

For this study, we quantified the effects of runtime estimation inaccuracy on backfilling, in both FCFS and relaxed FCFS scheduling, in a Blue Gene/P [1] setting. First, we studied how the inaccuracy of user runtime estimates impacts job scheduling policies. Using different priority policies with the same backfilling scheme, we investigated how more accurate runtime estimates affect the scheduling. Next, we customized the scheduler code to explore which part of the scheduling (job queue prioritizing or backfilling) is more sensitive to the accuracy of runtime estimates. Based on the results, we designed several estimation-adjusting schemes

978-1-4244-6443-2/10/$26.00 ©2010 IEEE

using historical workload data. The experimental results demonstrated that our adjusting schemes can achieve up to 20% improvement on job-scheduling performance measured by average waiting time, unitless wait, and slowdown.

Our motivation for this study is based on the fact that previous studies have failed to agree on how the inaccuracy of runtime estimation impacts scheduling. We focus on the Blue Gene/P for two reasons. First, most previous work on backfilling enhancement has been focused on cluster systems, such as IBM SP2 [20] and O2K [5], whereas backfilling on Blue Gene architecture is different from that on cluster systems. Therefore, previous results cannot directly be applied to Blue Gene/P systems. Second, the scheduling policies deployed on the Blue Gene/P at Argonne heavily rely on user job runtime estimation because the scheduler uses backfilling and short-job-first prioritizing schemes. We wanted to explore whether there was room for enhancement with more accurate runtime estimations.

In this paper, several terms regarding job runtime are used repeatedly. For example, we use job *actual runtime* (t_{act}) for job execution time. We use *user-requested runtime* (t_{req}) to represent the runtime estimates provided by users at job submission. The resource manager kills jobs when this time expires, so such time is also called job *walltime*. We use t_{sched} to represent the job walltime used by scheduler for prioritizing and backfilling jobs. Usually, t_{sched} equals to t_{req}. But in this work, t_{sched} can be other adjusted values.

The remainder of this paper is organized as follows. Section II introduces some background, including a short description of the Blue Gene/P system at Argonne and the scheduling policy used on the system. Section III discusses some related work. Section IV presents our empirical study of how the inaccuracy of user runtime estimates impacts job scheduling. Section V presents our user runtime estimate-adjusting schemes and the experimental evaluation. Section VI presents a brief summary and suggestions for future work.

II. Background

We begin with a brief description of the Blue Gene/P at Argonne. We then discuss some previous work with user runtime estimates, setting the background for our studies.

A. Blue Gene/P

The IBM Blue Gene/P [1] platform is a high-performance computing architecture. It is highly scalable, with three out of the top ten systems on the June 2009 Top500 list [4] belonging to the Blue Gene family. Blue Gene systems use a novel partitioned 3D torus interconnect in order to provide good network performance to concurrent jobs running on the system. This network partitioning isolates jobs from one another, providing extremely reproducible job performance, as well as improved network performance compared with a single, shared-torus network.

While network partitioning provides compelling features, they come at a cost. Nodes are grouped into 512-node midplanes, corresponding to an $8 \times 8 \times 8$ building block that can be combined with other midplanes to form partitions larger than 512 nodes. Midplanes can be grouped only with its neighbors along larger blocks. These larger blocks must also have a uniform length in each dimension.

Also, midplanes are connected into larger blocks by using a shared set of wiring. Wire sets are shared between midplanes in the same 1D slice of the system. Each midplane is connected to three sets of wires (one for each dimension of the torus) and can be connected either back to itself or to other midplanes on the wire set. Except for the X dimension, where extra wiring resources are available, only one multi-midplane partition can use a wire set at once; all other midplanes connected to that wire set must be connected back to themselves, limiting them to belonging to partitions of length one in that dimension. Mesh blocks, which use less wiring resources, can also be used. These provide less network performance, however, can be advantageous for some workloads [9].

These restrictions greatly reduce the interchangeability of resources on Blue Gene systems and can make it considerably more difficult to drain resources appropriate to a given job than on a traditional cluster using a switched network. The partitioned structure also makes job allocation on Blue Gene systems more restricted than on other mesh/torus-connected machines [15][30]. In practice, we have found all these make Blue Gene systems sensitive to backfill performance.

B. Intrepid

Intrepid is a 557 TF, 40-rack Blue Gene/P system deployed at Argonne National Laboratory. This system comprises 40,960 quad-core nodes, with 163,840 cores, associated I/O nodes, storage servers, and an I/O network. It debuted as No. 3 in the TOP 500 supercomputer list released in June 2008 and was ranked No. 8 in the latest list released in November 2009 [4].

Intrepid has been in full production since the beginning of 2009. The system is used primarily for scientific and engineering computing. The vast majority of the use is allocated to awardees of the DOE Innovative and Novel Computational Impact on Theory and Experiment (INCITE) program [3].

Intrepid is a capability platform, that is, it is specifically intended to run jobs that individually use a substantial fraction of the resources of the entire system. On Intrepid, smaller jobs are run, 8K and 16K-node jobs are common, and 32K-node jobs run regularly without administrator intervention. Because of a combination of the resource allocation issues described in the previous section, and this capability workload, the scheduling workload on Intrepid is quite different from the workloads seen on smaller systems.

C. Job Scheduling on Intrepid

Cobalt [2] is used on Intrepid for job scheduling. It is an open-source, component-based resource management tool, used on a number of Blue Gene systems worldwide. In order for system owners to easily customize scheduling policies, Cobalt uses utility functions to prioritizing jobs [26]. In particular, on Intrepid, the following utility function is used for production scheduling:

$$S_i = (t_{queue}/t_{req})^3 \times n_i \quad (1)$$

Here, S_i denotes job i's utility score; and t_{queue}, t_{req}, and n_i denote the queue waiting time, user-requested runtime, and the number of nodes of job i, respectively. At each scheduling point, for instance, every 10 seconds, the scheduler sorts the jobs by their utility score and picks the top-score job run. A partition blocking least other partition will be selected to allocate the scheduled job [26]. If no suitable partitions can run the job, backfilling may occur.

In order to adapt to the special architecture design of Blue Gene, a modified backfilling scheme is used. If the job at the top of the queue cannot start because of resource insufficiency, it drains a partition that can run it; the partition should be freed first in the future, according to the estimated end time of all running jobs. At this point, jobs that are not at the top of the queue can run on any idle partitions other than the drained partitions. Jobs with estimated runtime shorter than the *backfilling window* can be backfilled, where the backfilling window is determined by the longest-running job within the drained partitions. Figure 1 illustrates partition draining and backfilling.

The scheduling policy depicted in Figure 1 depends on user runtime estimates in three ways:

- *Job Prioritizing.* In job prioritizing, user runtime estimates directly influence the job's priority. A smaller estimate leads to higher priority. Thus, inaccurate estimates may cause inappropriate queue orders.
- *Job Backfilling.* Selection of a queuing job to backfill requires knowing the job's runtime in order not to delay the top-score job. Overestimated runtime of a job may cause the job to lose its backfilling chance. For example, if job g's end time is overestimated to be later than that of job a, job g cannot be backfilled.
- *Partition Draining.* When the job at top of the queue cannot start because of resource insufficiency, it waits for the partition that is anticipated to be freed first. That is, selecting where to drain relies on the runtime estimates of all running jobs. Inaccurate runtime estimates for partition draining bring two effects. First, it may cause the wrong decision in selecting a partition to drain. For example, if the estimated runtime of job a is longer than that of job c, job e will drain partition $R3R4$ instead of $R1R2$. Second, it may change the backfilling window, thereby impacting other jobs' chances of backfilling. For example, if job a is overestimated, other jobs (e.g., job g) are more likely to backfill.

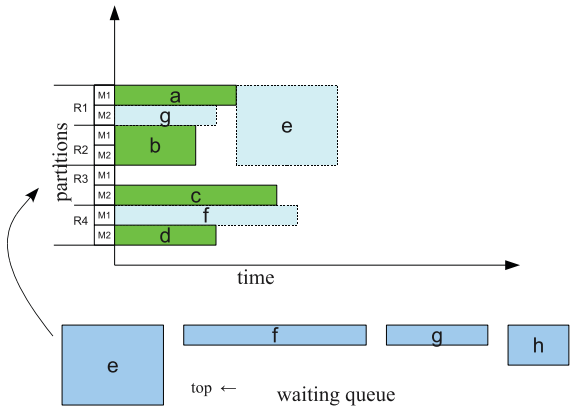

Figure 1. Backfilling on Intrepid. R1, R2, R3, and R4 represent 4 racks. M1 and M2 represent two midplanes within each rack. Current time is at the start point of the time axis. Job a, b, c, and d are currently running on partition $R1M1$, $R2$, $R3M2$, and $R4M2$, respectively. Jobs e, f, g, and h are waiting in the queue, ordered from left to right. In this scenario, top-priority job e cannot start because of resource insufficiency. It will drain the partition $R1R2$, waiting for the completion of job a. Job a determines the backfilling window. Job f is longer than the backfilling window, but it can start immediately at partition $R4M1$. Job g is shorter than the backfilling window, so it can be backfilled into partition $R1M2$.

III. Related Work

Past studies have focused on various aspects of runtime estimation, including the accuracy and its impact on job scheduling and schemes for improving the accuracy.

A. Inaccuracy of User Estimation

User-provided runtime estimates are known to be inaccurate. For example, Cirne and Berman [7] showed that in four different traces, 50% to 60% of jobs use less than 20% of their requested time. Ward et al. [29] reported that jobs on the Cray T3E used on average only 29% of their requested time. Chiang et al. [5] studied a certain workload and found that users grossly overestimate their jobs' runtime, with 35% of jobs using less than 10% of their requested time. Similar patterns are seen in other workload analyses [20][25].

In this study, we provide a statistic of user runtime estimates on the production Blue Gene/P system at Argonne, which also presents highly inaccuracy.

B. Impact of User Runtime Estimates on Job Scheduling

Considerable work has been done on backfilling job scheduling and the dependence on runtime estimation. Many results suggest that using more accurate requested runtime has only minimal impact on system performance [20][24][31]. Additional results in [20] show that doubling user-requested runtime slightly improves, on average, the slowdown and response time for IBM SP workloads using

FCFS-backfill. Tsafrir and Feitelson [28] present analytical explanations of why inaccuracy could better the scheduling. Others also report controversial results. Chiang et al. [5] have examined this question on the NCSA Original 2000 (O2K) and shown that more accurate requested runtime can improve system performance much more significantly than suggested in previous studies. Srinivasan et al. [25] have studied the effect of various backfilling schemes on different priority policies and observed that inaccurate estimates can cause significant deterioration of the overall slowdown. Zhang et al. [32] shows that even though the average job behavior is insensitive to the average degree of overestimation, individual jobs can be affected; under common backfilling schemes, users that provide more accurate runtimes are favored over ones that do not.

Unlike those studies, this paper focuses on analyzing and adjusting user runtime estimates for better scheduling on Blue Gene systems, which is a new petascale platform for high-performance computing. Basically, our conclusion is that under different job scheduling policies, the impact of job runtime estimates differs. Moreover, the job queuing policy is more sensitive to runtime estimates in impacting system performance than job allocation is.

C. Efforts on Improving User Estimations

Numerous efforts have been devoted to improving the accuracy of user runtime estimates. Lee et al. [19] tried to improve user estimation by removing the threat of job killing at walltime expiration and providing tangible reward for accurate estimates. However, the experiments show their method leads to only insubstantial improvement in the overall average accuracy. Considerable research has focused on using system-generated prediction to better the estimation accuracy. Suggested prediction schemes include using the top of a 95% confidence interval of job runtime [12], a statistical model based on the (usually) long uniform distribution of runtime [8], using the mean plus 1.5 standard deviations, genetic algorithms [24][23], instance-based learning [22], rough set theory [18], and three-phase adaptive prediction [13][14]. Tsafrir et al. [27] proposed a simple runtime predictor that just averages the runtime of the last two jobs by the same user.

Our work differs from previous work in the following ways. First, unlike the theoretical studies that focus on runtime prediction, our study presents a practical approach to improve job scheduling on production Blue Gene/P systems. Second, different from the studies that use only users' recent information, our runtime adjustment method uses both similar and recent information. Further, our work is not intended to predict the user runtime; rather, it predicts the accuracy of the user estimation, which can dynamically create more accurate runtime estimates.

IV. Analyzing User Runtime Estimates

We begin by analyzing user runtime estimates on the Blue Gene/P.

A. Job Trace

We collected about seven months of job traces from Argonne's Intrepid, from the time the 40-rack Blue Gene/P went into production at the start of 2009. We separated the job trace into two workloads: WL-I and WL-II (see Table I for details). WL-I, which contains 48,056 jobs, was used as the historical workload. WL-II, which contains 10,479 jobs, was used as the testing workload. From the job trace, we obtained a series of attributes for each job, such as job id, job size (number of computing nodes), submission time, user requested runtime (runtime estimates), actual runtime, user name, and project name.

Table I
Basic Information about Workloads WL-I and WL-II

Name	Jobs	Time Span	Users	Projects
WL-I	48,056	Jan–Jun 2009	207	84
WL-II	10,479	Jul 2009	103	61

Figure 2 presents the characterization of WL-II by grouping the jobs by scale and length. By job scale, we have four job groups: *very small*, *small*, *large*, and *very large*. By job length, jobs are divided into four categories: *very short*, *short*, *long*, and *very long*. The grouping criteria and the job proportion of each group are shown in the figure. In the experiment, we examines the performance impact based on different job categories.

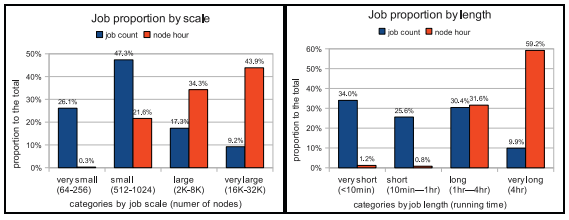

Figure 2. Workload Characterization of WL-II.

B. Accuracy of User Runtime Estimates

To measure the accuracy of user runtime estimates, we define R as follows:

$$R = \frac{t_{act}}{t_{req}}. \quad (2)$$

Here, R is the ratio of the job's actual runtime to the user-requested runtime; a higher R value correlates to higher runtime estimation accuracy. Normally, users tend to overestimate a job's runtime, in order to avoid having the job killed before completion. Thus, R is usually smaller than one. Theoretically, R should also be no larger than

one because the job will be killed at the requested runtime expiration. But, in fact, the job traces show jobs with R value slightly larger than one. The reason is that those jobs complete or are killed at the requested runtime expiration but take some extra time to get the resource cleaned and returned. Each job has an R value. Figure 3 shows the R value distribution of the jobs in the two traces.

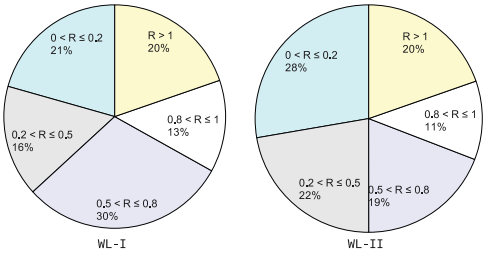

Figure 3. Distributions of R.

As shown in the figure, 20% of the jobs are slightly larger than 1; their runtimes are either correctly estimated or underestimated (and killed). Approximately 11%–13% of the jobs can be considered accurate. Most job estimates are not accurate ($R < 0.5$); indeed, more than 20% of the jobs have an R value under 0.2. In summary, user runtime estimates for jobs on Intrepid are highly inaccurate.

C. Effect of More Accurate Runtime Estimates on Job Scheduling

In this subsection, we examine the question "How much do more accurate job estimations improve job scheduling?" We first explain our methodology and then discuss our results.

1) Methodology: Our study has two goals. First, we want to measure whether higher estimation accuracy improves job scheduling performance. Second, we want to explore which aspect of job scheduling—job prioritizing, backfilling, or partition draining—is more sensitive to the runtime estimates.

For the first goal, we designed experiments as follows. From the job traces we can get the user runtime estimate (t_{req}) and actual job runtime (t_{act}). Normally t_{req} is no less than t_{act}. In practice, t_{sched} is equal to t_{req} because t_{act} is unknown at job submission. In this study, we investigated system performance by using different t_{sched} values. In particular, we introduced a parameter a, a real number between 0 and 1, to tune t_{sched} between t_{req} and t_{act}. That is,

$$t_{sched} = a \times t_{act} + (1 - a) \times t_{req}. \quad (3)$$

Clearly, a larger a makes t_{sched} closer to the actual runtime, meaning a more accurate estimation. When a is 0, t_{sched} equals the user-provided runtime estimation; when a is 1, t_{sched} equals the actual runtime, representing a perfect estimation. Note that t_{sched} is used only for scheduling; the job will still be killed at the expiration of the original user-requested runtime.

By varying a using the values 0, 0.25, 0.50, 0.75, and 1, we examined system performance using FCFS-backfilling and WFP-backfilling on the WL-II trace. WFP refers to a job prioritizing policy using the utility function as shown in Equation 1. We wanted to examine whether a more accurate estimate improves job scheduling and how sensitive it is to different prioritizing schemes.

For the second goal, we customized the job scheduler to use actual runtime in different parts of the whole scheduling scenario that rely on runtime estimates, such as job queuing (prioritizing), job backfilling, and partition draining (as stated in Section II-C). Specifically, we conducted five simulations with different combinations as described in Table II. In each table cell, 0 means using the user-provided estimation ($a = 0$), and 1 means using actual runtime ($a = 1$), for each of the three aspects. For example, $S001$ means using user estimates for queuing and backfilling, while using actual runtime for partition draining; this approach avoids overestimating running jobs and hence may decrease other jobs' backfilling chances and may help top-priority jobs obtain appropriate locations to drain. On the contrary, $S010$ means using actual runtime only for backfilling jobs; this approach may increase the chances of backfilling by reducing the overestimation for jobs to be backfilled.

Table II
SELECTIVE METHODS

Name	Job Queuing	Job Backfill	Partition Draining	Where To use t_{act}
$S001$	0	0	1	only for partition draining
$S010$	0	1	0	only for job backfilling
$S100$	1	0	0	only for job queuing
$S110$	1	1	0	for queuing and backfilling
$S111$	1	1	1	for all cases

Three metrics are used to measure system performance.

- *Waiting Time ($wait$)*: the time period between job arrival and start.
- *Slowdown ($slowdown$)*: the ratio of the job's response time to its actual running time, which is defined by $slowdown = \frac{t_{queue}+t_{act}}{t_{act}}$.
- *Unitless Waiting Time ($uwait$)*: the ratio of the job's wait time to its user-requested or estimated runtime, $uwait = \frac{t_{queue}}{t_{req}}$.

Both $uwait$ and $slowdown$ factor in the job running length; they both capture the fact that, for example, waiting an hour before running is more painful for a 10-minute job than for a 6-hour job. The difference is the way job length is measured: $slowdown$ relies on the actual job runtime (t_{act}), whereas $uwait$ depends on the user-estimated runtime

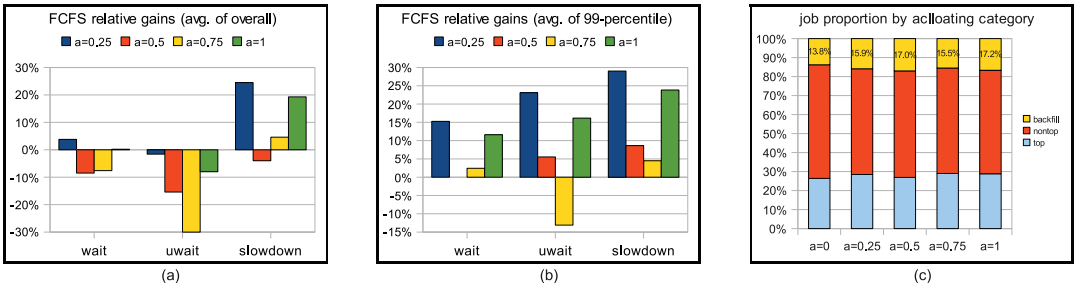

Figure 4. Simulation results of FCFS with synthetic accuracy.

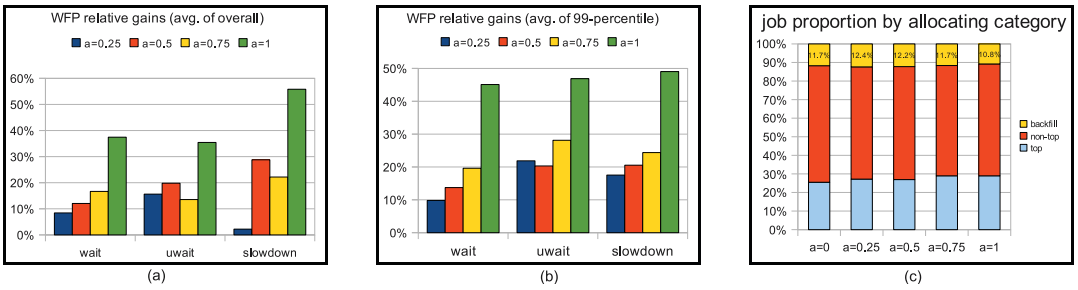

Figure 5. Simulation results of WFP with synthetic accuracy.

(t_{req}). Because of the inaccuracy of user estimates, t_{req} and t_{act} are always different. Hence, $slowdown$ and $uwait$ are not necessarily consistent. In other words, $slowdown$ reflects the response time normalized by job length from the system's perspective, whereas $uwait$ reflects the metric from the user's point of view. For example, if we assume that a job is requested by a user for 6 hours but actually ends in 10 minutes (perhaps crashes because of a bug), from the user's point of view this job is a 6-hour job, but from the system's point of view it is a 10-minute job. So this job's $slowdown$ is much higher than its $uwait$.

We usually measured the average value of those metrics among all the jobs. But since the distribution of results in job scheduling is easily skewed by a small proportion of outliers, merely consulting the average value among all the jobs does not suffice [11]. Therefore, we also measure the average values of jobs that are below the 99th percentile, removing 1% of the jobs as outliers, in order to provide a more general view of scheduling performance.

2) Results: We conducted simulation using the job scheduling simulator along with the Cobalt release named Qsim [26]. We first used the user estimates from job trace ($a = 0$) to simulate the job scheduling on workload WL-II, on both FCFS/backfilling and WFP/backfilling. Table III presents the simulated results. Some cells contain two numbers: the upper numbers are the average among all jobs, and the lower numbers are the average among jobs below 99th percentile. The table also presents the number of backfilled jobs. We call this set of results the baseline; later results are compared with the baseline to calculate their relative gains.

Table III
BASELINE PERFORMANCE

Sched. Policy	Wait (min)	Uwait	Slowdown	Backfilled Jobs
FCFS/backfill	80.9 65.5	1.92 1.18	10.14 5.79	1791
WFP/backfill	62.9 50.9	0.96 0.64	6.36 3.65	1293

Clearly, the WFP outperforms FCFS on every performance metric. The number of backfilled jobs of WFP is less than that of FCFS. The reason is that in WFP, more short jobs get high priority to run at the top of the queue instead of being backfilled. Our results also implicitly suggest that the overall scheduling performance is more sensitive to priority function than to the number of backfillings.

Tuning a to 0.25, 0.50, 0.75, and 1, we ran simulations on the same workload. Figure 4 shows the results with FCFS as the priority function. Figures $4(a)$ and $4(b)$ show the relative gains compared with the baseline results, on different average values for each metric. Since the metrix are all "the smaller the better," the relative gains are calculated by $(V_{base} - V_{new})/V_{base}$, where V_{base} represents the baseline value and V_{new} represents the new value to be compared to baseline value. Figure 4(c) illustrates the percentage of jobs categorized in one of three ways: started from the top of the queue, started not from the top of the queue, and started by backfilling. For example, in Figure 1, jobs e, f, and g belong to those three categories, respectively.

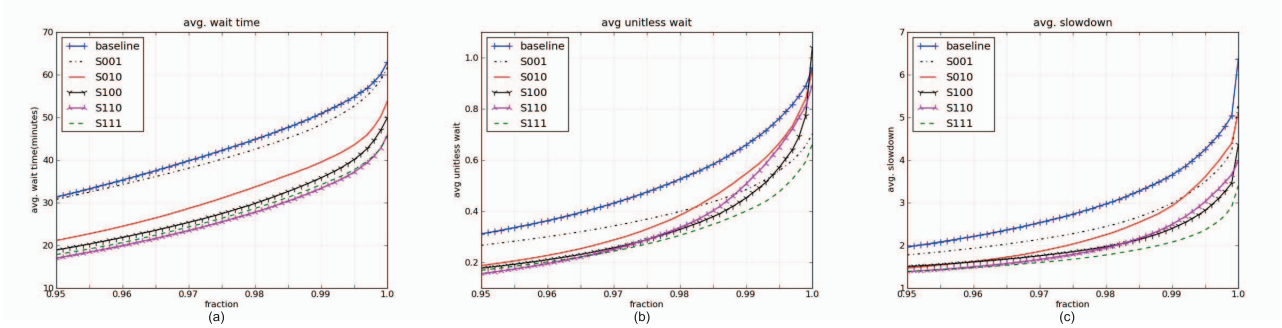

Figure 6. Simulation results of WFP with selective methods using actual runtime.

As shown in Figure $4(a)$, using more accurate runtime estimations does not necessarily improve performance, compared with using user-provided estimates. Specifically, the wait time is improved by less than 5% when $a = 0.25$ and remains almost the same when using the actual runtime ($a = 1$). Even worse, the other two cases cause nearly 10% degradation. For the unitless wait time, all the efforts to make the estimation more accurate generate worse results than using the user estimation. The slowdown is the metric most sensitive to the estimation accuracy. Except for $a = 0.5$, all cases get positive gains on slowdown.

Figure $4(b)$, regarding 99% of the jobs, shows that the average performance values are generally improved by more accurate runtime estimations. This result implies that the 1% of the jobs excluded in Figure $4(b)$ get very bad performance, thus making the overall average gains not as good as shown those in Figure $4(a)$. Therefore, we can argue that more accurate runtime estimation for FCFS with backfilling can improve the average performance for most of the jobs, but with a very small number of jobs significantly degrading in *wait* and *uwait*. Thus the overall average improvement is not seen clearly.

We notice that the results show no deterministic performance improvement. From the figures, we can see that probably $a = 0.25$ is best and $a = 1$ second best. These results make it difficult to improve the performance on FCFS with backfilling by improving the estimation accuracy. We do not even know what accuracy is the best because we cannot conclude from the results that higher accuracy brings higher overall performance.

Arguably, higher accuracy results in more backfilling. We believe the explanation lies in the following: The accuracy makes it easier to backfill small jobs, more than compensating for the impact on the draining jobs by shortening their backfilling.

The simulation results on WFP with backfilling are different. As shown in Figures $5(a)$ and (b), the averages of the overall jobs as well as for the 99th percentile correlate almost completely with the accuracy. The higher the accuracy, the greater the performance improvement. For perfect accuracy, the gains for the overall average for the three metrics are significant, achieving 30% to over 50%. As shown in Figure 5(c), the greater the accuracy, the more the number of jobs that will be started at the top of the queue. This result suggests that we can improve WFP with backfilling by improving the accuracy of the runtime estimation as much as possible.

Figure 6 shows the simulation results for the five scenarios in Table II. The y-axis represents the average value of the evaluated metrics. The x-axis represents the fraction of total jobs used to calculate the average value. For example, 1.0 means the average among all the jobs, 0.99 means the average among jobs below the 99th percentile, and so on. As shown in the figure, all the scenarios achieve better performance as against the baseline. Hence, using the actual runtime in any of the dependent places helps enhance performance. We also observe that the performance enhancement of different scenarios differs. Generally, $S001$ and $S010$ achieve comparatively less improvement than do the other three. That is, the runtime estimation accuracy in queuing job priority is more important than that used in the backfilling algorithm. We note that $S111$ and $S110$ achieve the best enhancement because they both use the actual runtime in queuing job prioritizing and backfilling job length. The difference is that $S110$ keeps the running time overestimated, potentially allowing more jobs to be backfilled. On $wait$, $S110$ gets slightly better performance than does $S111$. It appears that the strategy of relaxing the backfilling window is effective here. But for the total job average, $S111$ and $S110$ show very close average value, meaning that for $S110$ there are more large outliers or more jobs get a very large wait time to increase the total average value. For $uwait$ and $slowdown$, this trend is seen more clearly: the plots of $S110$ soar abruptly when the fraction is larger than 0.97. In sum, the strategy of using more accurate estimation except for calculating the backfilling window of running jobs is good for performance for the majority of the jobs, but it tends to cause top-queue job starvation than is the strategy of using actual runtime for all kinds of jobs.

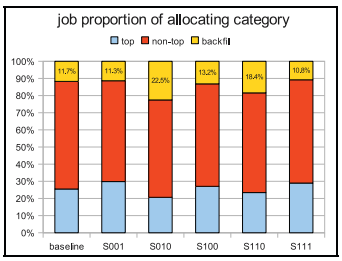

Figure 7. Job proportions of different allocation categories for selective methods.

Figure 7 shows the job proportions of different allocating categories, comparing the different selective schemes. Clearly, $S010$ and $S110$ have the largest numbers of backfilling jobs, because they both relax the backfilling windows and make the backfilling jobs less overestimated. Further, the number of backfilling jobs of $S110$ is less than that of $S010$. The reason is that $S110$ moves some very short jobs to the top of the queue, while those jobs may be backfilled in the $S010$ scheme. The number of backfilling jobs for $S111$ is the least. But our earlier results show that $S111$ achieves the best performance with regard to good average values and less starvation. Thus, we conclude that backfilling is not the best way to increase performance improvement. It can be considered only as a way of mitigating some resource loss caused by some scheduling policy suffering from fragmentation, or "holes." For example, in Figure 1, partition R1M2 during the current time through the end of job a is such a hole. Rather than filling up the holes, it is more effective to design scheduling and allocating schemes that cause fewer holes, since the jobs in the waiting queue do not often match those holes well. In our experience, the short-job-first prioritizing policy and proper job allocation schemes can help to decrease such holes. This conclusion is similar to the results seen in [17], which compares the roles of processor allocation and job scheduling in achieving good performance on hypercube computers and shows that job scheduling has far more impact on performance than does processor allocation.

V. Adjusting User Runtime Estimates

In this section, we explore the effect of adjusting user runtime estimates. As before, we begin by explaining our methodology, followed by presenting our results.

A. Methodology

Based on the results from the preceding section, we designed several schemes to adjust user runtime estimates in order to achieve better scheduling performance.

Instead of predicting the job's actual runtime at submission, we predict the accuracy of runtime estimation (i.e., R value). Thus, according to the user-provided estimation and the predicted accuracy, we can get the adjusted runtime estimates used by scheduling as follows:

$$t_{sched} = t_{req} \times A, \tag{4}$$

where A is an adjusting parameter. A can also be considered as the "predicted estimation accuracy." At job submission, we predict the A value based on historical job R values. The historical job R values are obtained from the historical workload, for instance, WL-I, in our simulation. According to different ways of getting A and t_{sched}, we have the following adjusting schemes:

- *User-based scheme:* In this scheme, we consult the historical jobs belonging to the same user. Assuming a job j belongs to user u, and user u has N jobs in the historical workload, we set A as the 80th-percentile of those N jobs' R values. That is, A is larger than 80% of those N jobs' R values. If the user u has few jobs (less than a threshold) in the historical workload, we prefer not to adjust the user estimates by setting the A value to 1. In our experiment, we set the threshold to 50.
- *Project-based scheme:* In this scheme, we consult the historical jobs belonging to the same project. The project-based scheme is similar to the user-based scheme except that we change the "user" to "project."
- *Combined scheme:* Besides predicting A based on either user or project information, we can combine both. If a job belongs to user u and project p, let A_u be the A value from the user-based scheme, A_p be the A value from the project-based scheme. Then the combined A value can be calculated from following scheme. If both A_u and A_p are 1 because of lacking historical data, we set the combined A as 1, meaning not to adjust the runtime estimation. If only one kind of data is not sufficient, we set A to the one based on the data that is sufficient. For example, if A_u is 1 and A_p is not, we set A to A_p, vice versa. If we get both valid A values based on user and project, we use the average of the A_u and A_p as the combined A value. Equation 5 describes the scheme to get the combined A value:

$$A = \begin{cases} 1 & \text{if } A_u = 1, A_p = 1 \\ A_p & \text{if } A_u = 1, A_p < 1 \\ A_u & \text{if } A_u < 1, A_p = 1 \\ avg(A_u, A_p) & \text{if } A_u < 1, A_p < 1 \end{cases} \tag{5}$$

- *Selective scheme:* We also designed a selective scheme that imposes runtime estimation adjustment only on certain kinds of jobs. Specifically, we designed a selective adjusting scheme similar to $S110$ in Table II that uses adjusted runtime estimates (by the combined scheme) for prioritizing queued jobs and for selecting backfilling jobs, while using the user-estimated runtime to drain partitions. Using adjusted estimates for queuing jobs

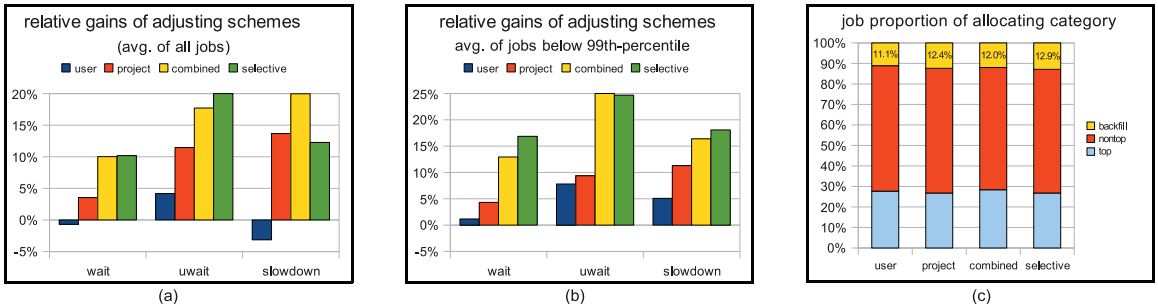

Figure 8. Simulation results of using runtime estimates adjusting schemes on WFP with backfilling.

can achieve more accurate job priority order. Using adjusted estimates for jobs to be backfilled and using user-provided overestimated runtime for partition draining both contribute to increase the chances of backfilling. This scheme differs from $S110$ that the $S110$ uses actual runtime for queue prioritizing and backfilling while the selective adjusting scheme uses the combined scheme to predict A value and then gets an adjusted runtime estimates (i.e. t_{sched}) for scheduling.

We implemented these schemes in Cobalt by introducing a functionality component to periodically parse the historical workload and maintain a table of R values for all the users and projects. We also made a few code changes in the queue manager to predict the A value and provide the adjusted runtime estimates t_{sched} to the scheduler. An important advantage of the implementation is that it is totally transparent to the user. Users submit jobs as usual; jobs are still killed at the expiration of the user-requested runtime.

B. Results

Figure 8 shows the simulation results. We see that the combined approach is better than the user-based or project-based scheme. The user-based scheme is even worse, perhaps because the number of users is more than the number of projects, so for each user the historical data is not as rich as each project. In addition, there are some new users in the simulation workload, whereas the projects are comparatively more stable. Since the combined scheme makes use of both user and project historical data, it can achieve better performance.

The performance of the selective scheme is also better than that of the user-based and project-based schemes. Compared with the combined scheme, the selective scheme is better on the average of jobs below the 99th percentile but not as good as the combined scheme for the entire average. This result suggests that relaxing the backfilling window can help small and short jobs but may impact some top priority jobs, thereby creating some performance outliers.

As for the relative gains, the combined scheme can improve the average performance of total jobs by 10.0% for waiting time, 17.7% for unitless wait, and 20.0% for slowdown. The selective scheme can improve these metrics by 10.2%, 20.21%, and 12.3%, respectively. For the average among jobs below the 99th percentile, the relative gains are up to 25% for both the combined and selective schemes. These values are comparable to those in the simulation with synthetic accuracy $a = 0.75$ (shown in Figure 5). The numbers of backfilling jobs are close for different schemes, as shown in Figure 8(c).

Figure 9 shows the performance for job categories by job size, that is, the number of computing nodes. Generally, the smaller the job, the better the performance. As shown in the figure, the combined and selective adjusting schemes both outperform the baseline for most categories. As for the relative gains, very small or small jobs achieve greater improvements than larger ones do. Comparing the combined and selective schemes, we noticed that for the average waiting time, the smaller job in the selective scheme outperforms the combined one, whereas the larger job in the combined scheme outperforms that in the selective scheme. This result is in line with our earlier analysis showing that the selective scheme tends to make large jobs in draining more likely to let other small jobs backfill, reducing the chance of getting started earlier than the drained deadline. A similar trend can be seen for *uwait* and *slowdown*, but it is not as clear.

Figure 10 shows the performance for job categories by runtime length. Generally, the longer the job, the longer it must wait. But longer jobs achieve relatively small *uwait* and *slowdown*. As shown in the figure, the combined and selective adjusting schemes both outperform the baseline in almost all the categories. As for relative gains, shorter jobs achieve greater improvement by estimates-adjusting schemes than longer jobs do. Comparing the combined and selective schemes, we find that for the average waiting time, the selective scheme outperforms the combined scheme for all the three categories except for very long jobs. For the other two metrics, the combined scheme and selective scheme each performs better for certain categories.

Figure 9. Performance categorized by job size.

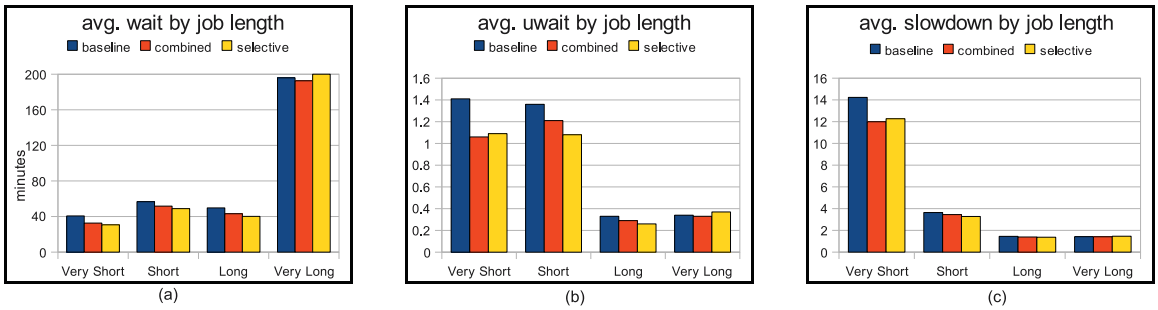

Figure 10. Performance categorized by job length.

VI. Conclusion

We have studied the impact of user runtime estimates on job scheduling policies. Using different priority policies with the same backfilling scheme, we have investigated how more accurate runtime estimates affect the scheduling. We have also customized the scheduler code to explore which part of the scheduling is more sensitive to the accuracy of runtime estimates.

Our analysis indicates that FCFS is not sensitive to user runtime estimates, whereas more accurate runtime estimates can considerably improve performance of the scheduling policies that favor short jobs. We have also found out that more accurate runtime estimates improve job scheduling mainly by setting proper short-job-first order; its influence on scheduling through backfilling is not as significant as through queue prioritizing. Based on the analysis, we have designed four estimation-adjusting schemes using historical workload data. Our experimental results indicate that the combined scheme that utilizes historical information of both user and project outperforms user-based and project-based schemes. According to our experiments with real system workloads, the combined scheme can improve system performance by up to 20%. In addition to performance improvement, our adjusting schemes are transparent to users and easy to deploy.

We plan to extend this study in several directions. First, we will develop an aging algorithm to minimize the influence of aged data. Second, we will use more information besides user and project for runtime estimation. Third, we will deploy the combined scheme on production systems like Intrepid and get on-site performance measurement.

Acknowledgments

This work is supported in part by National Science Foundation grants CNS-0834514, CNS-0720549, and CCF-0702737. The work at Argonne National Laboratory is supported by the U.S. Department of Energy, under Contract DE-AC02-06CH11357.

References

[1] Blue Gene Team, "Overview of the IBM Blue Gene/P project," *IBM Journal of Research and Development* 52(1/2), 199–220, 2008.

[2] Cobalt project. *http://trac.mcs.anl.gov/projects/cobalt*

[3] DOE INCITE program. *http://www.er.doe.gov/ascr/incite*

[4] TOP500 Supercomputing web site, *http://www.top500.org*

[5] S.-H. Chiang, A. Arpaci-Dusseau, and M. Vernon, "The impact of more accurate requested runtimes on production job scheduling performance," in *Proc. of Job Scheduling Strategies for Parallel Processing*, 2002.

[6] S.-H. Chiang and M. Vernon, "Production job scheduling for parallel shared memory systems," in *Proc. of International Parallel and Distributed Processing Symposium*, 2001.

[7] W. Cirne and F. Berman, "A comprehensive model of the supercomputer workload." in *Proc. of IEEE International Workshop on Workload Characterization*, 2001.

[8] A. Downey, "Predicting queue times on space-sharing parallel computers," in *Proc. of IEEE International Parallel Processing Symposium*, 1997.

[9] N. Desai, D. Buntinas, D. Buettner, P. Balaji, and A. Chan, "Improving Resource Availability By Relaxing Network Allocation Constraints on the Blue Gene/P," in *Proc. of International Conference on Parallel Processing*, 2009.

[10] Y. Etsion and D. Tsafrir, "A short survey of commercial cluster batch schedulers," *Technical Report 2005-13*, the Hebrew University of Jerusalem, 2005.

[11] E. Frachtenberg and D. Feitelson, "Pitfalls in parallel job scheduling evaluation," in *Proc. of Job Scheduling Strategies for Parallel Processing*, 2005.

[12] R. Gibbons, "A historical application profiler for use by parallel schedulers," in *Proc. of Job Scheduling Strategies for Parallel Processing*, 1997.

[13] C. Glansner and J. Volkert, "An architecture for an adaptive run-time prediction system," *International Symposium on Parallel and Distributed Computing*, 2008.

[14] C. Glansner and J. Volkert, "A three-phase adaptive prediction system of run-time of jobs based on user behaviour," *International Conference on Complex, Intelligent and Software Intensive Systems*, 2009.

[15] L. K. Goh and B. Veeravalli, "Design and performance evaluation of combined first-fit task allocation and migration strategies in mesh multiprocessor systems", *Parallel Computing* 34(9), 508–520, 2008.

[16] J. Jones and B. Nitzberg, "Scheduling for parallel supercomputing: A historical perspective of achievable utilization," in *Proc. of Job Scheduling Strategies for Parallel Processing*, 1999.

[17] P. Krueger, T. Lai, and V. Dixit-Radiya, "Job scheduling is more important than processor allocation for hypercube computers," *IEEE Transactions on Parallel and Distributed Systems* 5(5), 488–497, 1994.

[18] S. Krishnaswamy, S. Loke, and A. Zaslavsky, "Estimating computation times of data-intensive applications," *IEEE Distributed Systems Online*, vol. 5, no. 4, Apr. 2004.

[19] C. Lee, Y. Schwartzman, J. Hardy, and A. Snavely, "Are user runtime estimates inherently inaccurate?" in *Proc. of Job Scheduling Strategies for Parallel Processing*, 2004.

[20] A. Mu'alem and D. 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, 2001.

[21] D. Perkovic and P. Keleher, "Randomization, speculation, and adaptation in batch schedulers," *IEEE/ACM Supercomputing Conference*, 2000.

[22] W. Smith, "Prediction services for distributed computing," in *Proc. of IEEE International Parallel and Distributed Processing Symposium*, 2007.

[23] W. Smith, I. Foster, and V. Taylor, "Predicting application runtimes with historical information," *Journal of Parallel and Distributed Computing* 64(9), 1007–1016, 2004.

[24] W. Smith, V. Taylor, and I. Fostor, "Using run-time predictions to estimate queue wait times and improve scheduler performance," in *Proc. of Job Scheduling Strategies for Parallel Processing*, 1999.

[25] S. Srinivasan, R. Kettimuthu, V. Subramani, and P. Sadayappan, "Characterization of backfilling strategies for parallel job scheduling," in *Proc. of the International Conference on Parallel Processing Workshops*, 2002.

[26] W. Tang, Z. Lan, N. Desai, and D. Buettner, "Fault-aware, utility-based job scheduling on Blue Gene/P systems," in *Proc. of IEEE International Conference on Cluster Computing*, 2009.

[27] D. Tsafrir, Y. Etsion, and D. Feitelson, "Backfilling using system-generated predictions rather than user runtime estimates," *IEEE Transactions on Parallel and Distributed Systems* 18(6), 789–803, 2007.

[28] D. Tsafrir and D. Feitelson, "The dynamics of backfilling: Solving the mystery of why increased inaccuracy may help," in *Proc. of IEEE International Symposium on Workload Characterization*, 2006.

[29] W. Ward, C. Mahood, and J. West, "Scheduling jobs on parallel systems using a relaxed backfill strategy," in *Proc. of Job Scheduling Strategies for Parallel Processing*, 2002.

[30] B. Yoo and C. Das, "A fast and efficient processor allocation scheme for mesh-connected multicomputers," *IEEE Transactions on Computers* 51(1), 46–60, 2002.

[31] D. Zotkin and P. Keleher, "Job-length estimation and performance in backfilling schedulers," in *Proc. of IEEE International Symposium on High Performance Distributed Computing*, 1999.

[32] Y. Zhang, H. Franke, J. Moreira, and A. Sivasubramaniam, "Improving parallel job scheduling by combining gang scheduling and backfilling techniques," in *Proc. of IEEE International Parallel and Distributed Processing Symposium*, 2000.