J. Parallel Distrib. Comput. 73 (2013) 926–938

Contents lists available at SciVerse ScienceDirect

J. Parallel Distrib. Comput.

journal homepage: www.elsevier.com/locate/jpdc

Job scheduling with adjusted runtime estimates on production supercomputers

Wei Tang [a,*], Narayan Desai [b], Daniel Buettner [b], Zhiling Lan [a]

[a] Illinois Institute of Technology, Chicago, IL 60616, USA
[b] Argonne National Laboratory, Argonne, IL 60439, USA

HIGHLIGHTS

- We studied the inaccuracy of user runtime estimates in large amount of job traces.
- We proposed a set of runtime adjusting schemes to better the estimation accuracy.
- We refined our schemes to avoid impact of too much adjusting (underestimates).
- We used real job trace to evaluate our schemes and got positive results.

ARTICLE INFO

Article history:
Received 29 August 2011
Received in revised form
4 February 2013
Accepted 12 February 2013
Available online 6 March 2013

Keywords:
Job scheduling
Runtime estimates
Walltime prediction

ABSTRACT

The estimate of a parallel job's running time (walltime) is an important attribute used by resource managers and job schedulers in various scenarios, such as backfilling and short-job-first scheduling. This value is provided by the user, however, and has been repeatedly shown to be inaccurate. We studied the workload characteristic based on a large amount of historical data (over 275,000 jobs in two and a half years) from a production leadership-class computer. Based on that study, we proposed a set of walltime adjustment schemes producing more accurate estimates. To ensure the utility of these schemes on production systems, we analyzed their potential impact in scheduling and evaluated the schemes with an event-driven simulator. Our experimental results show that our method can achieve not only better overall estimation accuracy but also improved overall system performance. Specifically, the average estimation accuracy of the tested workload can be improved by up to 35%, and the system performance in terms of average waiting time and weighted average waiting time can be improved by up to 22% and 28%, respectively.

1. Introduction

In a supercomputing systems, the job runtime estimate, also called requested walltime, is an important job attribute provided by users at job submission. Although this value was originally used by resource managers to kill a job at its expiration, the value is also heavily used in job scheduling. Backfilling [15], for example, needs to know the expected runtime of both running and waiting jobs so that it can fill short jobs into backfilling windows, reducing fragmentation without delaying high-priority jobs. Some schedulers favor short jobs in order to achieve improved average response time [23]; they need to know the runtime estimates of the waiting jobs when sorting the queue. Moreover, job runtime estimates are essential to other resource management strategies, such as advance reservation [11], queuing time prediction [7,20], and walltime-aware job allocation reducing fragmentation on torus-connected systems [24].

However, user estimates of job running time have been repeatedly demonstrated to be highly inaccurate [3,30,2]. Indeed, a large number of jobs consume only a small portion of the walltime requested. A number of studies have been done to investigate whether such inaccuracy can impact job scheduling performance. Surprisingly controversial results have been reported. On one hand, some claimed inaccuracy is helpful. For example, Mu'alem et al. [15] reported that the inaccurate runtime estimates have the potential to be beneficial because of backfilling; such results have led to the suggestion that estimates should be doubled [34] or randomized [17] to make them even less accurate. On the other hand, some others suggested accuracy is more favorable. Studies have shown that using more accurate runtime estimates can improve system performance far more significantly than previously suggested [2,21,28].

In this paper, we present a set of walltime adjustment schemes that can be used by large-scale production systems directly. First,

* Corresponding author.
E-mail address: wtang6@iit.edu (W. Tang).

0743-7315/$ – see front matter © 2013 Elsevier Inc. All rights reserved.
http://dx.doi.org/10.1016/j.jpdc.2013.02.006

we studied workload characteristics based on a large amount of historical data (275,000 jobs in 30 months) from a leadership-class computer. Next, we proposed a set of walltime adjustment schemes to produce more accurate estimates, and we discussed how to configure each scheme for real computer systems. We evaluated the performance of our walltime adjustment schemes on production machines using simulations with real workloads. Our experimental results show that our method can achieve not only better overall estimation accuracy but also improved overall system performance. Specifically, the average and median of estimation accuracy of the tested workload can be improved by up to 35% and 42%, respectively. Moreover, the system performance in terms of average waiting time and weighted average waiting time can be improved by up to 22% and 28%, respectively.

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 walltime* (t_{req}), or simply walltime, to represent the runtime estimates provided by users at job submission; the resource manager kills jobs when this time expires. We use t_{sched} to represent the job walltime used by scheduler for prioritizing and backfilling jobs. Usually, t_{sched} equals t_{req}; but in this work, t_{sched} can be other adjusted values. In this context, the term "walltime adjustment" refers to the effort of a system to adjust the user's estimates to create possibly more accurate walltime estimates. The term "walltime estimate" refers to the runtime estimate either provided by users or adjusted by the system.

The remainder of this paper is organized as follows. Section 2 discusses some related work. Section 3 presents our study of historical job traces. Section 4 presents our walltime adjustment schemes and analytical evaluation. Section 5 presents our analysis of the impact of imperfect prediction and an enhancement for utilizing walltime adjustment. Section 6 presents a performance evaluation of scheduling using enhanced walltime adjustment. Section 7 summarizes our conclusions.

2. Related work

In this section we review some related studies that have focused on various aspects of runtime estimation, including accuracy and impact on job scheduling, and we present schemes for improving the accuracy.

2.1. Inaccuracy of user estimation

User-provided runtime estimates are known to be inaccurate. For example, Cirne and Berman [3] showed that in four different traces, 50%–60% of jobs used less than 20% of their requested time. Ward et al. [30] reported that jobs on the Cray T3E used on average only 29% of their requested time. Chiang et al. [2] studied a certain workload and found that users grossly overestimated their job runtime, with 35% of jobs using less than 10% of their requested time. Similar patterns are seen in other workload analyses [15,21]. We studied a large amount of data from a production Blue Gene/P system [1] and found that although the accuracy is better than previously reported, the user estimates are still highly inaccurate: half the jobs use less than 50% of their requested walltime.

2.2. 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 [15,20,34]. Additional results in [15] show that doubling the user-requested runtime slightly improves, on average, the slowdown and response time for IBM SP workloads using FCFS–backfill. Other results are conflicting. Chiang et al. [2] examined this question on the NCSA Origin 2000 (O2K) and showed that more accurate requested runtime can improve system performance much more significantly than suggested in previous studies. Srinivasan et al. [21] studied the effect of various backfilling schemes on different priority policies and observed that inaccurate estimates can significantly deteriorate the overall performance. On the other hand, Tsafrir et al. [29,26] offered two reasons that inaccuracy could better the scheduling: (1) an invalid model (F-model) is used in modeling user estimates, and (2) the improved performance is due to the "heel and toe" effect—that is, the FCFS–backfilling is switched into a short-job-first type of policy. Zhang et al. [33] showed 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 who provide more accurate runtimes are favored over ones that do not.

Our work also reveals the relationship between the accuracy and scheduling performance, but we do not use artificial or modeled inaccuracy. Instead, we evaluate the performance change using the workload after applying real walltime adjustment schemes, where the requested walltime is adjusted based on the real inputs.

2.3. Motivation for improving user accuracy

Besides the goal of improving the job scheduling performance directly [22,28], a wide range of related work shares the common motivation of improving the accuracy of runtime estimation. One example is advance reservations for grid allocation and collocation, shown to benefit considerably from better accuracy [11,20,14]. Another is scheduling moldable jobs that may run on any number of nodes [7,20,4]. The scheduler's goal is to minimize response time, considering whether waiting for more nodes to become available is preferable over running immediately. Thus, a reliable prediction of how long it will take for additional nodes to become available is crucial. Recently, Yuan et al. [32] proposed PV-EASY backfilling; this scheme, used to guarantee strict fairness and protect the interests of blocked top-priority jobs, also needs more accurate walltime estimates. Similarly, our previous work [24] proposed a walltime-aware job allocation strategy that tries to pack jobs with similar size and length together to reduce fragmentation on torus-connected system; its performance improves with better walltime estimation accuracy.

Our primary motivation is to better the backfilling and queue sorting. But, with more accurate estimates, our schemes benefit all the other motivating problems that requiring more accurate estimates.

2.4. Efforts to improve user estimations

Numerous efforts have been devoted to improving the accuracy of user runtime estimates. Lee et al. [13] tried to improve user estimation by removing the threat of job killing at walltime expiration and providing tangible reward for accurate estimates. However, experiments showed that 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 [8], a statistical model based on the (usually) long uniform distribution of runtime [7], using the mean plus 1.5 standard deviations, genetic algorithms [20,19], instance-based learning [18], rough set theory [12], and three-phase adaptive prediction [9,10]. Tsafrir et al. [28] proposed a runtime predictor that averages the runtime of the last two jobs by the same user. Wu et al. [31] proposed an adaptive hybrid method (AHModel) for Grid load prediction within confidence windows.

Table 1
Basic information about the job trace.

Attribute	Value
Number of jobs	275,858
Time span	January, 1, 2009–June 30, 2011
Number of computing nodes	40,960
Avg. job running time	5199 s
Avg. job requested walltime	9096 s
Number of projects	208
Number of users	503

The main focus of this paper is how to use walltime adjustment to obtain more accurate walltime estimates and limit the adverse effect caused by inevitable undesirable estimates (e.g., underestimates). Our study is based on a large amount of recent data from leadership-class computer workloads. Our goal is to design effective strategies capable of being deployed on production supercomputer centers.

3. Studying the inaccuracy

In this section we introduce the job traces and the characteristics of the original user runtime estimates.

3.1. Job traces

We collected 30 months of job traces from Argonne's Intrepid, from January 2009, when the 40-rack Blue Gene/P system went into production, to June 2011, when this study started. The system comprises 40,960 quad-core nodes, with 163,840 cores, associated I/O nodes, storage servers, and an I/O network. This 0.5-petaflop machine debuted as No. 3 in the TOP500 supercomputer list released in June 2008 [25]. More background about the machine can be found in our previous work [6,22,24]. Here we focus only on the workload characteristics relevant to walltime estimates.

The job trace contains totally 275,858 completed jobs. For each job, we obtained a series of attributes, such as job id, job size (number of computing nodes), submission time, user-requested walltime (runtime estimates), start time, end time, user name, and project name. Table 1 shows some basic information of the job trace. A portion of the job trace can be found in the Parallel Workload Achieve in standard workload format [16].

3.2. Accuracy of user estimates

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

$$R = \frac{t_{act}}{t_{req}}. \quad (1)$$

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 1. Theoretically, R should also be no larger than 1 because the job will be killed at the requested runtime expiration. In fact, however, the job traces show jobs with an R value slightly larger than 1. 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. For simplicity, we consider all the R value of all these jobs to be 1.

Fig. 1 shows a rough distribution of the R values of every job in our trace. As shown in the figure, the R values of each job range from 0 to 1. Only approximately 12% of all these jobs have R values close to 1. Although most of them are underestimated jobs that are killed by the system at walltime expiration, they can be considered as perfect estimates from the scheduler's perspective because they are indeed provided with accurate estimates matching their actual runtime. Though killed, these jobs do not harm the system or other jobs from the scheduler point of view (although the unsaved work could impact the system and other jobs indirectly which is out of the scope of this paper). Including the killed jobs, only 25% of the jobs have an R value larger than 0.8, which can be considered as being highly accurate. On the contrary, about half the jobs consume only half of their requested walltime. Even worse, 28% of the jobs use less than 20% of their walltime. Although some of these jobs may have encountered software bugs, the substantial discrepancy between their requested walltime and actual runtime may interfere with job scheduling policies that depend on requested walltime.

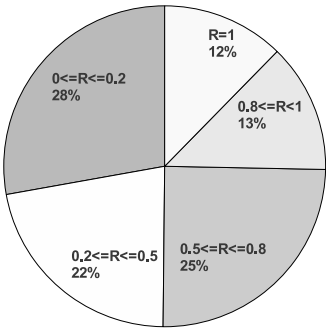


Fig. 1. Distributions of R.

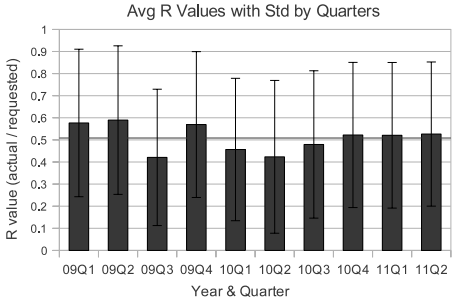


Fig. 2. Average R values by quarters.

Fig. 2 shows the average R values of all the jobs grouped by quarters. The average R values of all jobs are 0.50, with a standard deviation of 0.33. The average by quarters ranges from 0.42 to 0.59. In the next section, we introduce some adjustment schemes to increase the estimate accuracy, which can be reflected by increased average R values.

4. Walltime adjustment schemes

In this section, we explore the effect of a set of walltime adjustment schemes. We begin by explaining our methodology and then present our evaluation results.

4.1. Basic methodology

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 runtime estimate and the predicted accuracy, we can get the adjusted walltime estimate used by scheduling as follows:

$$t_{sched} = t_{req} \times A, \quad (2)$$

where A is an adjustment parameter. By multiplying A, the estimated walltime is expected to be closer to the actual runtime.

At job submission, we predict the A value based on the R values of the relevant historical jobs. These values can obtained from the historical job trace. Specially, when a job is submitted, we find a group of eligible historical jobs relevant to the newly submitted job; then, by calculating the R values of those historical jobs, we set A to certain metric of the historical R. Thus, it is essential to define how to choose the eligible historical jobs and how to get the adjustment parameter based on the historical accuracy values, which can be determined by a walltime adjustment scheme.

The walltime adjustment scheme is defined by three aspects: similarity, recency, and aggressiveness. Similarity and recency together can determine which historical jobs are eligible historical jobs. In other words, an eligible job should be "similar" to the newly submitted job, sharing some common feature or attribute. We can select similar jobs based on certain keys, such as user name, project name, or a combination of them; for example, we can use the user and project name as a double-key. Besides similarity, the eligible historical job should be within a limited "recent" time, since information about older jobs may no longer be relevant. The recency can be represented by a historical time window that rules out old jobs outside the window. Once the eligible historical jobs are determined, we can use a statistical metric to determine the A value. Such metrics include the average, median, maximum, or percentile. Different metrics represent different levels of "aggressiveness" to adjust walltime estimates. For example, if we use the median as the metric, the A value of a job is set to the median of the R values of its eligible historical jobs.

In summary, a scheme is determined when we determine a specific key type, historical time window, and statistical metric that represents the level of aggressiveness. For example, if we configure the key type to "user name", the time window to one-month, and the metric to the 80th percentile, then the adjustment scheme can be described as follows: When a job is submitted, assuming its user name is *usr*, its adjustment parameter A equals the 80th percentile of the historical jobs with user name *usr*, and the job ended within the past month, then the adjusted walltime can be calculated by Eq. (2).

After studying the characteristics of different keys, we identified four kinds of key types: only by user name, only by project name, by user and project combination, and by the combination of user, project, and walltime. We call the adjustment schemes with the first two key types single-key schemes; that with the third key type we call a double-key scheme; and that with the fourth key type we call a triple-key scheme.

We use these attributes as keys because we found when we group jobs by these keys, the standard deviation (*std*) of job R values are smaller than the *std* among the total jobs. The average *std*s of the grouped jobs R values in four cases are 0.308, 0.294, 0.285, and 0.253, which are all smaller than the total *std*, 0.335. The smaller *std* within a job group indicates that the R values within a group are less scattered and can be expected to provide more consistent adjustment. We use the user-provided walltime as one of the keys because this information can reflect the user knowledge of the input data set. For example, for the same application, a larger size of input data usually corresponds to a longer walltime, and vice versa. Since most of the jobs uses only a few number of requested walltime values [27], using walltime as a key is feasible, similar to user and project. Table 2 shows detailed information about the four key types. Note that the number of job groups are counted after ignoring some groups with job number less than 10. The reason is that those small groups tend to have very small *std*, which stretches the results.

For the historical time window, we have considered following options: all the available jobs, 12 months, 6 months, 3 months, and 1 month. The fewer months used, the more recency represented but the less information available. The effect of using less historical data may be twofold: it may result in a lack of information, yet on the other hand it discards some aged, misleading data. To avoid the scheme's being easily gamed by users, we do not consider a time window less than 1 month.

Table 2
Average standard deviation of job groups by different key(s).

Key type	No. of job groups	Avg std (R)
All jobs	1	0.335
Project	188	0.308
User	432	0.294
Double-key	614	0.285
Triple-key	2272	0.253

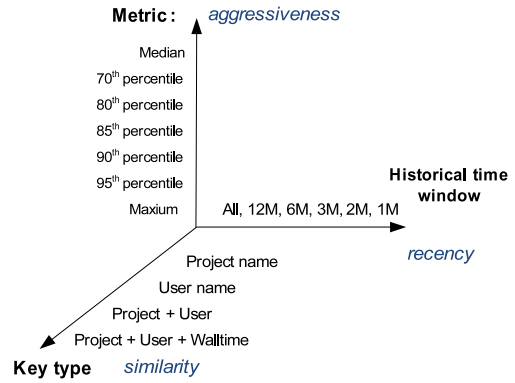


Fig. 3. Possible options for each of the three dimensions to determine a specific walltime adjustment scheme.

For the aggressiveness level, we can use several options: maximum (100th percentile), 95th percentile, 90th percentile, p85 percentile, 80th percentile, and median (50th percentile). As the percentile lowers, the aggressiveness level increases. A higher aggressiveness levels may produce a more accurate estimate but also may result in underestimates. Although we will not kill the job at the expiration of adjusted walltime, too many underestimated jobs in the system may violate the scheduling policy. We do not use the average because it is close to the median in our historical workloads.

Fig. 3 summarizes the possible options for each of the three dimensions.

4.2. Scheme evaluation

Having defined our methodology, we next evaluate the impact of the different schemes.

4.2.1. Evaluation metric

The effectiveness of the schemes is determined by two metrics: average accuracy of the workload with walltime adjustment and the proportion of underestimated jobs.

We measure the average accuracy of the workload as follows.

$$\text{Accuracy} = \begin{cases} 1 & \text{if } T_{act} = T_{est} \\ T_{act}/T_{est} & \text{if } T_{act} < T_{est} \\ T_{est}/T_{act} & \text{if } T_{act} > T_{est}. \end{cases} \quad (3)$$

Here T_{act} is the actual runtime of a job and T_{est} is the runtime estimate of a job, which is either the requested walltime provided by user or the system-adjusted walltime estimate. The R value model is part of this definition since all the estimates are larger than the actual runtime. This accuracy metric also covers the case that walltime can be underestimated, so T_{est} is smaller than T_{act}. Its value is between 0 and 1; and the larger the value is, the more closer the estimate to the actual runtime. This metric was used by Tsafrir et al. in a study of the dynamics of backfilling [28].

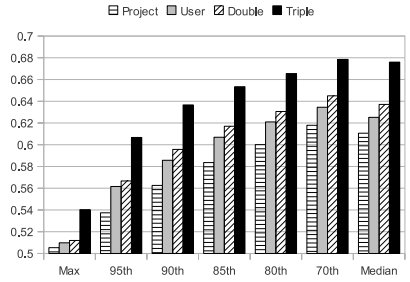


Fig. 4. Average accuracy comparison among schemes using different key types.

4.2.2. Similarity and aggressiveness

We test the schemes with different key types and aggressiveness levels. Specifically, we fix the historical time window to infinity first; all similar earlier jobs are considered eligible. Fig. 4 shows the results of average accuracy. In the figure, all the accuracy values are better than the base value, that is, the original average accuracy of all jobs, which is 0.50. A general trend of accuracy is that the lower the percentile used, the higher the accuracy achieved. And with the same percentile, the more keys used, the higher the accuracy achieved. In particular, the triple-key scheme is better than the other three schemes. Specifically, the highest accuracy is seen with using the triple-key scheme at the 70th percentile: the accuracy is 0.679, or 35% greater than the original average accuracy. We noticed "Median" has slightly lower accuracy than 70th-percentile. This may be because that "Median" is too aggressive that causes more over adjustment which result in underestimates.

Because the system-adjusted estimates are used in production scheduling, merely achieving better overall accuracy does not suffice to evaluate a scheme. A good scheme should be able to limit the number of underestimates as the by-product of walltime adjustment. The term "underestimate" here means that the system-adjusted walltime is shorter than the actual runtime. The counterpart term, "overestimates", means that the system-adjusted walltime falls between the actual runtime and the user-provided estimation. By definition, overestimation is desirable because the new walltime estimate is closer to the actual runtime, which corresponds to higher accuracy. Although "underestimate" sometimes means higher accuracy, it may have an undesirable effect on the job scheduler.

In addition to these two, there is another kind of prediction level: "no prediction". This means that, after walltime prediction, the job's walltime estimate remains the same as the original one provided by user. This situation is caused mainly by the lack of historical data in the same job category.

The impact of these different schemes is analyzed in Section 5. Here we simply bear in mind that underprediction is not desirable, and we provide statistics to evaluate the effectiveness of the walltime prediction schemes.

Fig. 5 shows the proportions of jobs with various adjustment extents, which we divide into four levels: no adjustment (NA), overestimation (OE), underestimation (UE), and bad estimation (BE). NP and OP are as described earlier. UE and BE both refer to underestimation, but UE means that the gap between the underestimated walltime and the actual runtime is less than 30 min, whereas the gap with BE is greater than 30 min. Thus, BE is highly undesirable, the reason being that too much underestimates will cause problems such as violation of backfilling reservations. More details will be discussed later.

As shown in the figure, for the triple-key scheme, when the percentile gets smaller, the proportion of NA jobs declines, while that of UE and BE jobs increases, and that of the OE jobs first increases and then drops. The accuracy tends to increase when the percentile gets smaller. Balancing the accuracy and the underestimate rate, percentiles between 80 and 90 are desirable: accuracy is reasonably high (larger than 0.6, 20% higher than that of the original trace); desirable OP jobs are above 50%, which also is higher than the original; and more important, the BE jobs' rate is at a very low level (1.2%–2.2%) and that of the UE is also comparably low (9%–17%).

The double-key scheme has similar UE and BE proportions, but the proportion of OP jobs is higher than that of the triple-key scheme after the 90th percentile. Because of space limitation, we do not show the results of the single-key scheme hereafter, because the trends are similar: the absolute accuracy improvement is lower than that of the triple-key and double-key schemes.

4.2.3. Threshold of adjustment parameter

To further decrease the proportion of underestimated jobs, we tested an enhancement that imposes a threshold on the adjustment parameter A. If A obtained by an adjustment scheme is less than a threshold, we set it to the threshold value. We reran the walltime adjustment schemes on triple-key and double-key schemes with a threshold of 0.5 and obtained the results shown in Figs. 6 and 7.

As shown in Fig. 6(a) and (b), the proportion of underestimated jobs drops at each percentile by 17%–35% compared with schemes without the threshold. This threshold will not influence the number of NA jobs, so the decreased job proportion is added to the UP jobs, as is desirable. By using the threshold, the improved accuracy is impacted slightly but can be ignored.

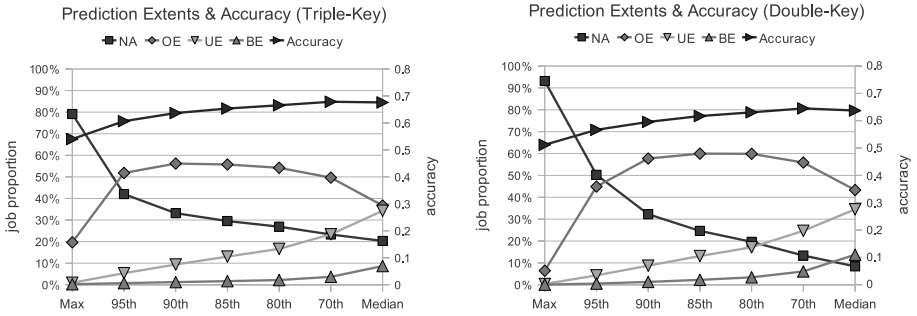
Fig. 7 looks similar to Fig. 6, but it represents the effect of the threshold on the double-key scheme. Imposing a lower-bound threshold on the adjustment parameter can reduce the proportion of underestimated jobs with average accuracy slightly decreased. Actually, the threshold not only contributes to the proportion of the underestimated jobs but also limits the likelihood of a job being underestimated. For example, with a lower-bound of 0.5 for the adjustment parameter, the underestimated job will achieve an accuracy of at least 0.5, meaning its estimation is larger than half the actual runtime. But without the lower-bound, a job after walltime adjustment may have an estimate that is only a small fraction of the runtime (e.g., 0.1), which is misleading to the scheduler.

4.2.4. Recency

We also studied the impact of the historical time window. We ran the same set of previous tests but added the historical time window as a constraint. That is, only those jobs within the recent historical time window could be used for future adjustment. Specifically, we tested time windows for 1 month, 2 months, 3 months, 6 months, and 12 months.

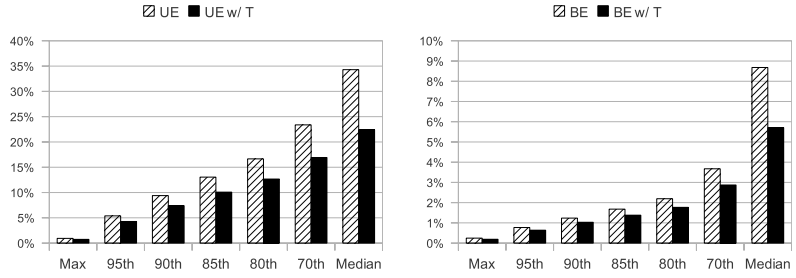
Fig. 8 shows the accuracy variance caused by the using different historical time windows. For the triple-key scheme, the smaller time window generally results in slightly higher accuracy. Although this trend is clearly seen only for percentiles larger than the 90th, the accuracies of 1M (the smallest window) are larger than using ALL jobs (the largest window), with the exception of the 70th percentile, which has close accuracy values for different time windows. For the double-key scheme, the trend that smaller windows result in higher accuracy is seen consistently.

We also examined the job proportion of underestimated jobs (both BE and UE). As shown in Fig. 9(a), the proportion of BE jobs varies with different time windows at each percentile used. Generally, the smaller the window, the fewer the number of BE jobs (although some exceptions can be found in the middle). The smallest window (1M) always has fewer BE jobs than the largest window does. Such trends are especially clear for the 70th percentile and median cases. The double-key scheme shows a similar trend (Fig. 9(b)) when the percentile is smaller than the 85th. For other cases the smaller window may correspond to more BE jobs, but the numbers are close.



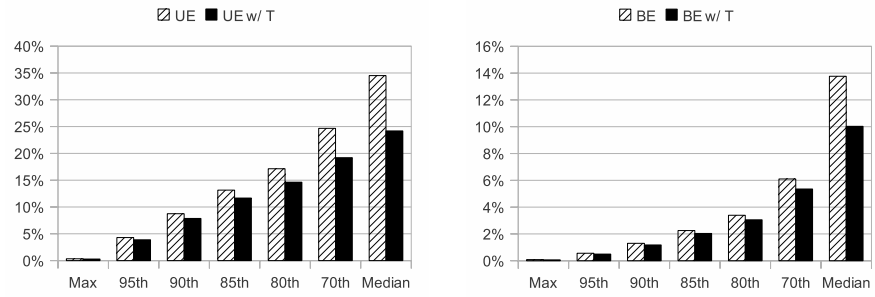
(a) Triple-key. (b) Double-key.

Fig. 5. Proportions of jobs with various adjustment extents and average accuracies. NA—No adjustment, OE—Overestimation, UE—Underestimation, BE—Bad estimation (30 min or more under).



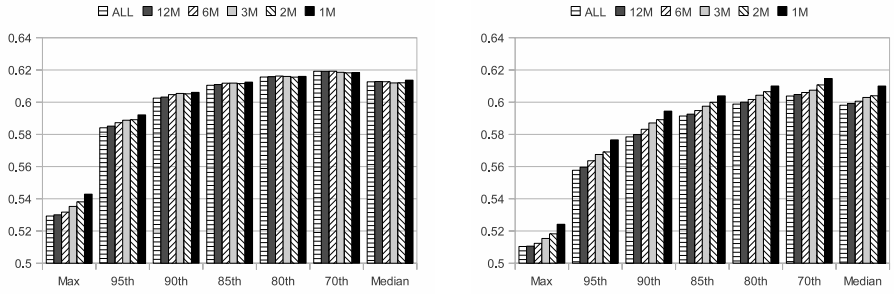
(a) Underestimation (UE). (b) Bad estimation (BE).

Fig. 6. Effects of threshold of adjustment parameter on triple-key schemes.



(a) Underestimation (UE). (b) Bad estimation (BE).

Fig. 7. Effects of threshold of adjustment parameter on double-key schemes.

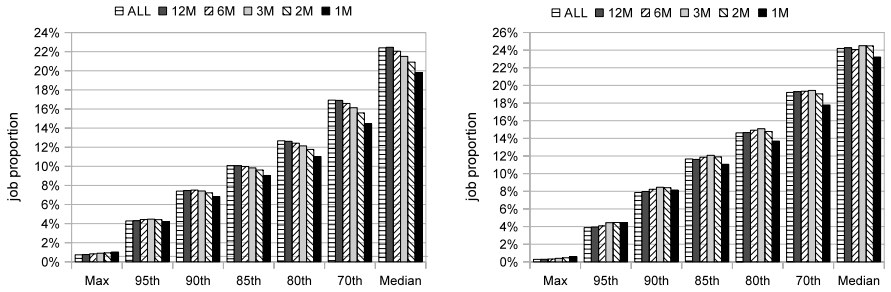


(a) Triple-key. (b) Double-key.

Fig. 8. Impact on avg. accuracy by using different historical time windows.

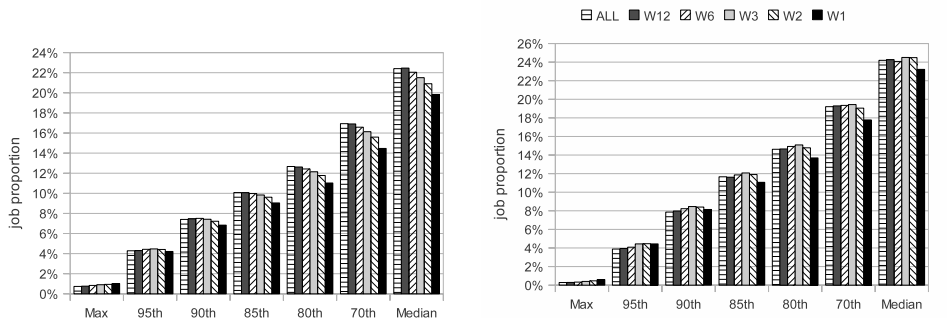
Fig. 10 shows the proportion of UE jobs using different time windows. Again, the smaller the window, the fewer the number of BE jobs. In particular, the proportion of UE jobs using a one-month window is significantly less than those using all jobs. Specifically, the proportion decreased by 10%–15% when we used one-month's recent jobs instead of all historical jobs with a percentile smaller than 90. For the double-key scheme, this decrease (up to 7%) is also clearly seen with percentiles smaller than 90. For the other percentiles tested, the smaller window's UE job proportions are very close to (if not less than) those of the larger windows' UE job proportions.

To compare the detailed distribution change of the job accuracies before and after walltime adjustment, we plotted the cumulative distribution of the job accuracy values for original



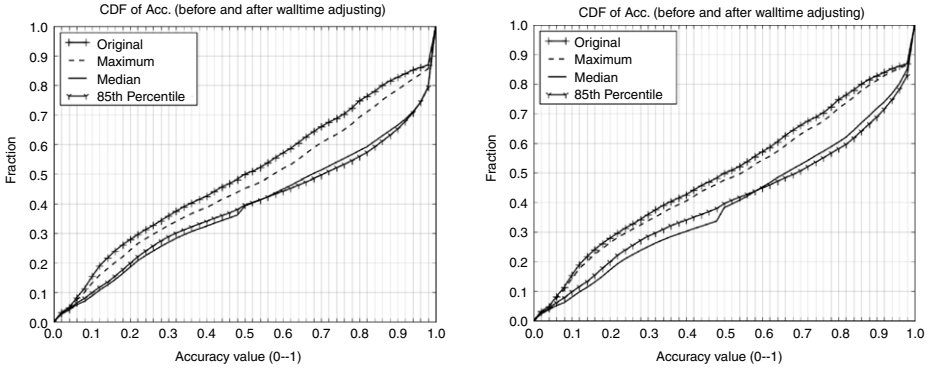
(a) Triple-key. (b) Double-key.

Fig. 9. Impact on BE job proportions by using different historical time windows.



(a) Triple-key. (b) Double-key.

Fig. 10. Impact on UE job proportions by using different historical time windows.



(a) Triple keys. (b) Double keys.

Fig. 11. Cumulative distribution of accuracy value: before and after walltime adjustment.

workload and several workloads after walltime adjustment (Fig. 11). In each figure, four lines are depicted. One is for the original workload, whose distribution is consistent with the proportions shown in Fig. 1. The other three lines represent the accuracy distribution for workloads after walltime adjustment. Maximum means using the 100th percentile, representing the most conservative scheme. Median means using the median as adjustment metric, representing the most aggressive scheme. Medium means using the 85th percentile scheme. Other percentile plots are close to the 85th-percentile line. For convenience of analysis, we omit those lines in the figure.

Fig. 11(a) shows the distribution comparison for the triple-key scheme. Each y-axis value represent a percentile of the total accuracy. For example, at the 60th percentile, the accuracy of each workload is 0.64, 0.70, 0.83, and 0.86 (from left to right, the x values on the lines when $y = 0.6$). Collectively, the lower the lines, the better. Thus, the adjusted walltimes have better accuracies than do the original ones. The "Maximum" is closest to the "Original" line, meaning it has the least improvement, consistent with the results observed earlier. Earlier results also indicated that the average accuracy using the median is better than the accuracies obtained using the 85th percentile. But in fact the median-scheme is better only for those jobs with low percentiles—that is, only when y is smaller than 0.4 is the "Median" line lower than "85th-percentile" line. If we compare the median accuracy instead of the average, the scheme using the 85th percentile is the best, with the value 0.71, slightly larger than median-scheme's 0.68, and significantly larger (42%) than the originals 0.5.

Fig. 11(b) shows that the double-key scheme has similar distributions except that the adjusted lines are closer to the original lines, a result that confirms the earlier results that the overall accuracy with a double-key scheme is lower than with a triple-key scheme.

Of note in the figure is an abrupt increase from $x = 0.48$ to $x = 0.50$. This is the effect of applying a threshold value of 0.5. It is most clearly seen for the most aggressive adjustment schemes shown in this figure, using the median metric.

Table 3
Effects of using adjusted walltimes.

Job	Overestimation (OE)	Underestimation (UE & BE)
Waiting	Backfill chance increased, no delay	Chance increased, may delay
Running	Shorter backfill window	Shorter window and delay

4.2.5. Discussion

The selection of an adjustment scheme is determined by resource management goals. Nevertheless, based on our own experience, we present here four general guidelines that can be used in selecting a particular scheme.

First, we prefer the scheme with the most increased overall accuracy (which can be measured by average or median accuracy among all jobs). Second, we prefer the scheme that yields fewer underpredicted jobs, especially badly estimated (BE) jobs, which may interfere with the scheduling. Third, to make the walltime prediction process simple and efficient, we prefer to minimize the amount of historical data. Fourth, to prevent users from gaming the system, we prefer to avoid using too few historical jobs.

For the first guideline, using a triple-key scheme and more aggressive confidence level is desirable (Fig. 4). With the second guideline, we can narrow down the scheme options to those with a confidence level from the 80th to the 90th percentile (Fig. 5). Placing a threshold on the adjustment parameter can effectively reduce the number of underestimated jobs (Figs. 6 and 7). In accordance with the third guideline, we found (fortunately) that less historical data not only does not degrade the prediction effectiveness but also is helpful (Figs. 8–10); thus, we suggest using only one-month historical jobs as the prediction base, both for simplicity and for better accuracy. In accordance with the fourth guideline, we do not consider using less data; indeed, we add a threshold that at least 10 similar jobs must exist to be eligible to adjust a job's walltime estimate. We want to ensure that users not achieve more priority by always submitting highly inaccurate (faked) jobs purposely.

5. Utilizing adjusted walltime

To deploy walltime adjustment schemes on production machines, we need to know the potential impact on the system. To this end, we analyzed the impact of walltime adjustment on job schedulers. Based on this analysis, we propose a selective walltime adjustment scheme that minimizes potential adverse effect introduced by walltime adjustment.

5.1. Impact of walltime adjustment

The walltime adjustment schemes affect the system differently in various scenarios. These scenarios include overestimation or underestimation, using adjusted walltimes for waiting jobs or running jobs, and using adjusted walltime for job prioritizing or backfilling.

A typical backfilling scenario is illustrated in the left part of Fig. 12, where job 1 is running and job 2 is currently scheduled (with top-priority in the queue) but it cannot get enough nodes to start running. In this case, job 2 will get a reservation at the time and location illustrated in the figure. The reservation determines the "backfill window", where lower-priority waiting jobs can be filled in this window to run ahead of time, since they are not going to delay the start of job 2. Without walltime adjustment, only job 5 can be backfilled in this example.

For waiting jobs, overestimated jobs can have a greater chance of being backfilled and will not harm other jobs. As shown in the figure, job 3 cannot be backfilled with user estimation, but it can with system-adjusted walltime. If a job is underestimated, however, although it is also more likely to be backfilled, it may delay the top-priority reserved job. For example, job 4 can be backfilled with the underestimated walltime, but it will delay the start of job 2 because job 4 will actually finish later than job 1.

For running jobs, overestimation and underestimation also have different impacts. If we adjust the walltime for running jobs (both OE and UE), the backfilling windows will be shortened (as shown in the right part of Fig. 12). The difference is that overestimation does not result in other bad effects, but underestimation will additionally cause delay for the job with a reservation at its expected end time, because the running job will not actually end at the adjusted walltime expiration.

Besides backfilling, job prioritizing is also impacted by system prediction if the queuing policy depends on the job walltime (e.g., short-job-first queuing policy). In this case, an overestimated waiting job will earn more priority than it can get without walltime adjustment because the requested walltime is smaller. Since it is closer to the runtime, this effect is desirable. If a waiting job is underestimated, on the other hand, the job will get more priority than it should, thus misleading the scheduler. In general, more accurate adjustment is more helpful to sort the queue in a real short-job-first manner. Table 3 summarizes the effects (both positive and negative) of using adjusted walltimes in job scheduling.

5.2. Selective walltime adjustment

Based on the analysis of walltime adjustment impacts, we propose a selective walltime adjustment scheme as an enhancement to the regular walltime adjustment. The enhancement is based on the fact that walltime adjustment used in different parts of the scheduling procedure can cause different effects. In particular, underpredictions can cause undesirable consequences on some occasions. We cannot control the underestimation, but we can try to avoid the underestimation impact by not using walltime adjustment in certain parts of the scheduling.

To this end, we modified the traditional job scheduler, allowing it to use walltime adjustment only for waiting jobs and to use user-provided walltime estimates for running jobs. In this way, we can achieve more accurate queuing order, create relaxed backfilling windows, have more jobs be backfilled, and avoid reserving a wrong location caused by the underestimation of running jobs. Although the underestimation for waiting jobs in backfilling can cause some delay of top-priority jobs, we still use it because overestimation for waiting jobs has other benefits (increasing the individual job's chance of being backfilled).

6. Performance evaluation

An essential goal of improving the accuracy is to improve the system performance. In this section we evaluate the performance improvement by deploying walltime adjustment in job scheduling. We first introduce the experiment setup and configurations and then present the experimental results.

6.1. Experiment setup

We implemented our schemes into a production resource manager named Cobalt [5] and conducted event-driven simulation with Qsim [22] extended with walltime adjustment schemes. Two strategies were evaluated: (1) applying walltime adjustment schemes to both waiting jobs and running jobs and (2) applying walltime adjustment only for waiting jobs, using user-provided estimates for running jobs.

We ran four sets of simulations. Each set was configured with a combination of a queuing policy (either SJF-like or FCFS) and a

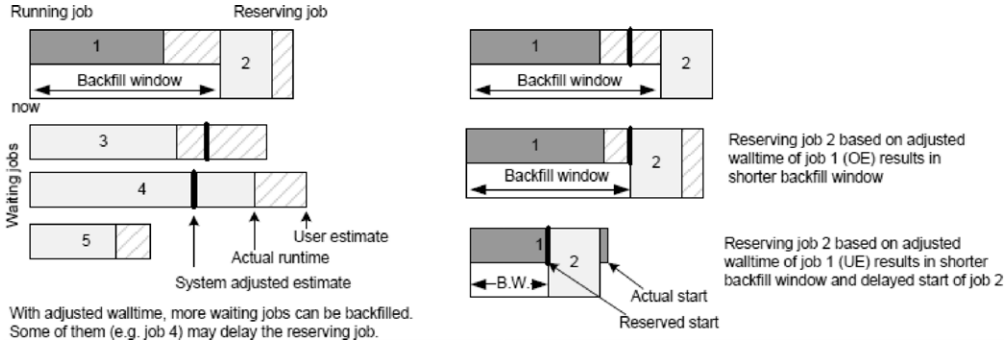


Fig. 12. Impacts of overestimation and underestimation.

walltime adjustment strategy (either regular or selective). To represent the SJF-like queuing policy, we used the priority function in Eq. (5), which is used on Intrepid, the production Blue Gene/P system at Argonne. Note that backfilling is supported for both queuing policies. With the regular scheme, the walltime adjustment scheme is applied to all the jobs in the system; with the selective scheme, walltime adjustment is applied only to waiting jobs, while the original user estimates are used for running jobs.

For each set of simulations, we tested four cases: one is the original scheme without walltime adjustment as baseline, and the other three use different walltime adjustment schemes determined by the three dimensions described earlier. Regarding similarity and recency, we use a triple-key scheme and one-month window, in order to get highest accuracy brought by these two dimensions. Regarding the level of aggressiveness, we use the 95th percentile, 85th percentile, and 70th percentile for the three schemes, which represent conservative, medium, and aggressiveness, respectively. The more aggressive the scheme, the greater the accuracy obtained, but with more underprediction produced.

We ran simulations using the real job trace from Intrepid on a monthly basis. We split the job trace from recent 12 months (July 2010 through June 2011) into 12 workloads. The workloads contain 7700 jobs on average (ranging from over 6000 to over 9000 jobs per month). For different months, the jobs have different average estimation accuracy, either before or after walltime adjustment. As shown in Fig. 13, the original accuracy is around 0.5, and the three walltime adjustment schemes increase the accuracy to above 0.6. Using the 70th percentile, one can obtain the highest accuracy for every month, but it also produces the most underestimation.

To measure the performance, we examined several metrics, including two widely used metrics: average waiting time and average slowdown. The waiting time of a job is the time between the time it is submitted and started. The slowdown, defined by that a job's waiting time plus running time and then divided by its running time, captures the fact that a longer job endures more waiting time than a shorter job does. We also measured a new metric, called weighted average waiting time, that is calculated by

$$WW = \frac{\sum_{i}^{n}(W_i * P_i)}{\sum_{i}^{n} P_i}, \tag{4}$$

where WW means the weighted average waiting time for the whole workload containing n jobs, W_i means the waiting job of a single job i, and P_i means job i's priority score at the time it is started. The priority score can be obtain based on the specific queuing policy. For example, the score of FCFS can be represented by the job's waiting time. For Intrepid, the priority score is calculated by

$$P_i = (t_{queue}/t_{req})^3 \times n_i, \tag{5}$$

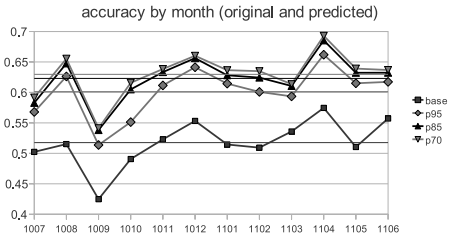


Fig. 13. Average accuracy values of tested workloads.

where t_{queue}, t_{req}, and n_i denote the queue waiting time, user-requested runtime, and the number of nodes of job i, respectively.

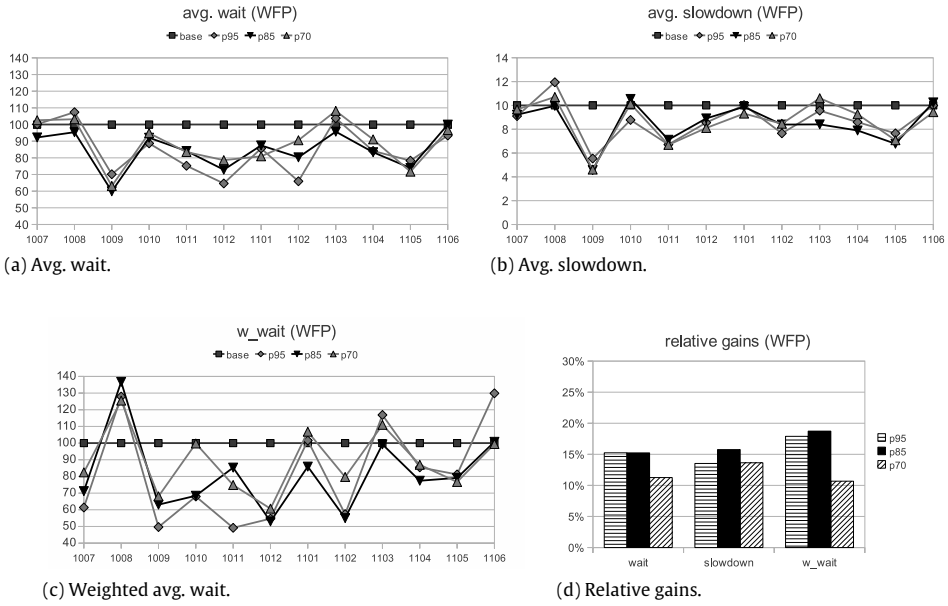
The weighted waiting time can reflect how much the higher-priority jobs are favored by the scheduler. If two workloads have same average waiting time but different weighted waiting time, we can say that the one with lower weighted waiting time has better performance because higher-priority jobs have endured less waiting time.

6.2. Results

We present here our experimental results, beginning with the SJF-like queuing policy and followed by FCFS.

6.2.1. SJF-like policy with walltime adjustment

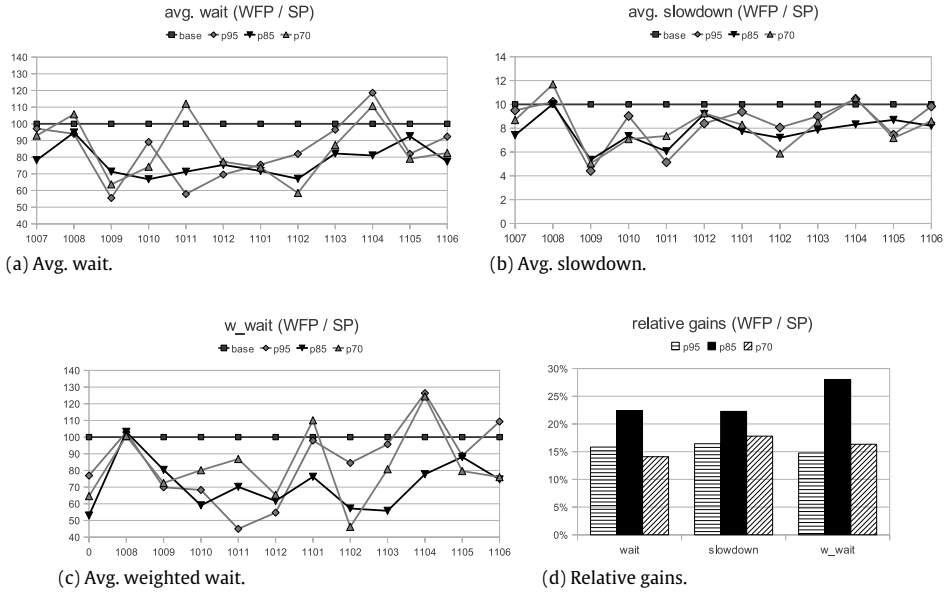
First we present the results of using a SJF-like scheduling policy (namely WFP, described by Eq. (5)). Fig. 14 shows the results for the regular strategy: applying walltime adjustment for both waiting jobs and running jobs. The first three charts plot the performance of the walltime adjustment schemes compared with original job scheduling (without walltime adjustment). The baseline represents the original scheme, and the *pxx* line represents walltime adjustment using the *xx*-percentile. In the plots, we assume the baseline waiting time and average waiting time equal 100 (minutes) and the slowdown equals 10, and normalize other values by these baselines. This is because what we are interested in is the improvement that walltime adjustment can bring. As shown in Fig. 14(a), the waiting times of the walltime adjustment cases are mostly under that of the baseline, meaning that for most cases, using walltime adjustment results in less average waiting times, with only a few exceptions from p95 and p70 (on 1007, 1108, and 1103). While p95 outperforms the other two for most months, p85 performs more consistently well under the baseline. For the best case, p85 saves 40% average waiting time (in 1009). The average slowdown and weighted wait have similar plots (Fig. 14(b) and (c)), which indicate that walltime adjustment can improve system performance, with a few exceptions. Fig. 14(d) shows the average improvement of each scheme on each metric. For the waiting time,



(a) Avg. wait. (b) Avg. slowdown.

(c) Weighted avg. wait. (d) Relative gains.

Fig. 14. Simulation results on monthly basis.



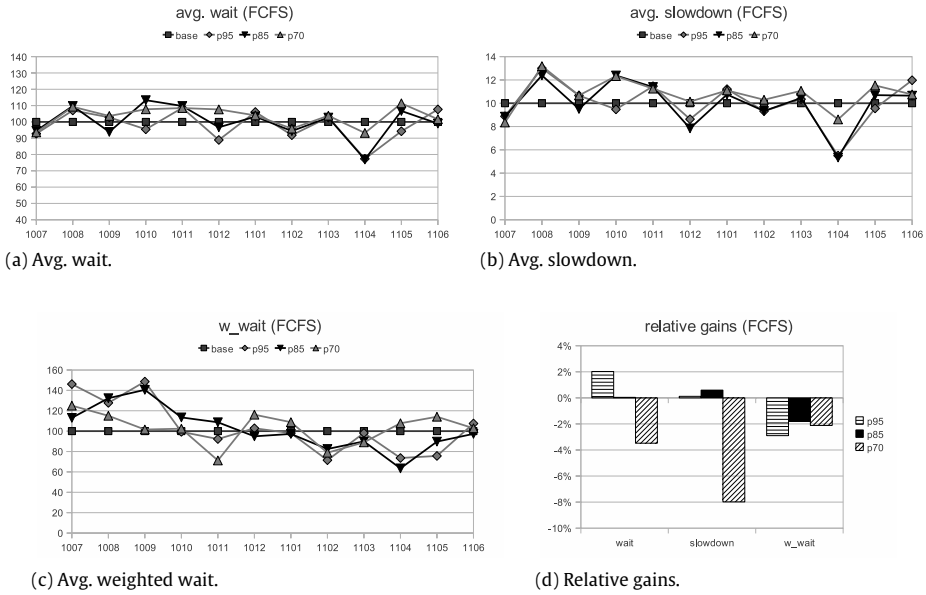
(a) Avg. wait. (b) Avg. slowdown.

(c) Avg. weighted wait. (d) Relative gains.

Fig. 15. Simulation results on monthly basis, using selective walltime adjustment schemes.

p95 and p85 can achieve around 15% improvement compared with the baseline. For the slowdown and weighted wait schemes, p85 outperforms the other two, achieving 16% on slowdown and 19% on weighted wait. P75 performs least effectively even though it has the best accuracy; this result indicates that underpredictions have a negative effect on scheduling. Overall, even with some inevitable underestimation, walltime adjustment schemes can improve the job scheduling performance.

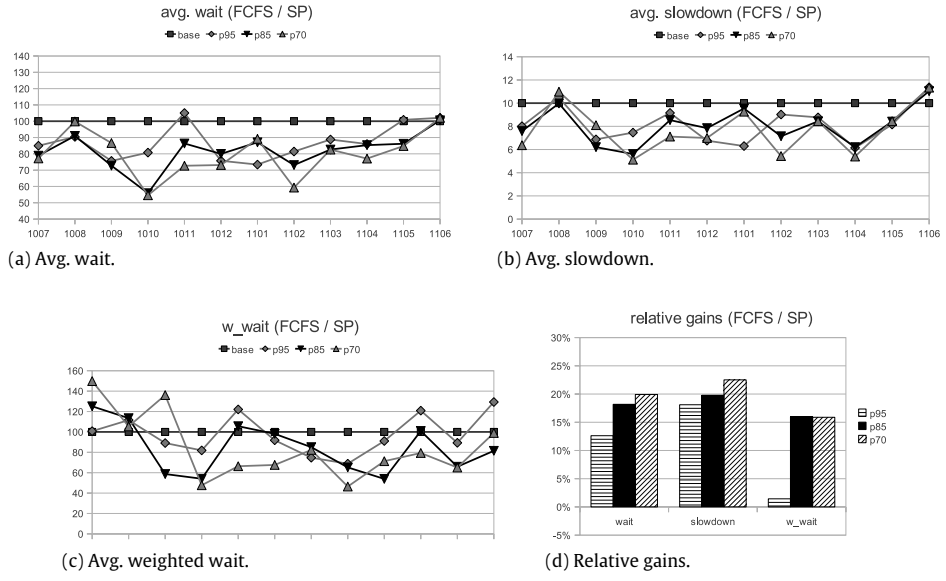
Now we examine the effectiveness of the selective walltime adjustment. As mentioned earlier, the selective scheme means using system-adjusted walltimes for waiting jobs (either in job prioritizing or backfilling) and using user-provided walltime estimates for running jobs (in deciding backfilling windows). In this way, we can get relaxed backfilling windows, increase the possibility of jobs being backfilled, and avoid the impact brought by underestimated running jobs. Fig. 15 shows the simulation results. In general, these results are better than those in Fig. 14. Not only are the lines generally lower than the baseline, but also the points above the baseline also are much reduced. In other words, the selective schemes improve performance more than the regular schemes do. Moreover, p85 performs particularly well; for most months, it can achieve over 20% improvement (under 80 for wait and weighted wait, under 8 for slowdown). The average gains of p85 reach 22%, 22%, and 28% for the three metrics, respectively. The most improved metric is weighted average waiting, which is desirable by the scheduler because the interests of high-priority jobs are better fulfilled. With selective adjustment, p70 has performs better than with regular adjustment schemes while p95 remains the same (or worse in terms of weighted waiting). This fact indicates that more performance gains comes mainly from avoiding underestimation for running jobs, because p70 has much more underestimation while p95 has very little.



(a) Avg. wait. (b) Avg. slowdown.

(c) Avg. weighted wait. (d) Relative gains.

Fig. 16. Simulation results on monthly basis, using selective walltime adjustment schemes.



(a) Avg. wait. (b) Avg. slowdown.

(c) Avg. weighted wait. (d) Relative gains.

Fig. 17. Simulation results on monthly basis, using selective walltime adjustment schemes.

6.2.2. *FCFS with walltime adjustment*

Now we examine the experimental results for using FCFS queuing policy. Figs. 16 and 17 show the simulation results with regular and selective walltime adjustment schemes, respectively. We found that regular walltime adjustment did not help improve system performance using the FCFS queuing policy. As shown in Fig. 16(a), the plots representing walltime prediction schemes oscillate around the baseline, though the extent is not large (mostly within 10% of the base value). The average slowdown and weighted wait have similar trends. Overall, the average relative gains are also not consistent (Fig. 16(d)). Using the 95th percentile realizes the highest gain, with only 2% increase in the average wait. Several cases have less than 1% gain, and others have negative impact. Among them, using the 70th percentile causes the biggest negative impact: −8% on slowdown. Other negative impacts are within −4%.

On the other hand, when we use FCFS with selective walltime adjustment, the results are surprisingly better, with the only change of inhibiting the walltime prediction for running jobs. As shown in Fig. 17, the lines representing the walltime adjustment schemes are mostly under the baseline. They even have fewer outliers compared with the WFP cases. For example, for p85 and p70, there is no performance degradation for a single month. The overall relative gains are not as much as seen by using WFP with selective walltime adjustment (Fig. 15), but they are considerable. P85 and p70 have more gains, achieving around 20% improvement on wait and slowdown. For the weighted average wait, the improvement is slightly over 15%.

7. Summary

In this paper, we have addressed an issue in job scheduling on production supercomputers where users provide inaccurate job

runtime estimates that are heavily used in job scheduling and resource management.

First, we studied a large amount of historical data from Intrepid, the production Blue Gene/P system at Argonne National Laboratory, and found that the user runtime estimates are highly inaccurate: 50% of jobs consumed less than 50% of their requested walltime.

Then, we designed a set of walltime adjustment schemes defined by a three-dimensional vector representing similarity, recency, and aggressiveness. We tested the schemes and showed that these schemes can improve the overall accuracy of runtime estimates. In particular, using a triple-key scheme, the 85th percentile, and one-month's historical window can produce good adjusted accuracy and a low proportion of underestimated jobs.

More generally, using more keys (triple key) and less aged data (short time window) results in better accuracy. However, while the more aggressive confidence window results in better accuracy, it introduces more underestimated jobs. In terms of accuracy, the average and median of estimation accuracy of the tested workload can be improved by up to 35% and 42%, respectively. In terms of underestimated jobs, schemes using the 85th or higher percentile can limit the underestimated rate to under 10% and the bad estimation rate to under 1.5%.

Further, we analyzed the effects of utilizing walltime adjustment in job scheduling, especially the potentially adverse impact brought by underestimation. Based on the analysis, we proposed an enhancement to walltime adjustment, called the selective walltime adjustment scheme, which allows the scheduler to use system-adjusted walltimes only for waiting jobs while using user estimates for running jobs.

To evaluate both the regular and the selective walltime adjustment schemes, we conducted simulation-based experiments under SJF-like and FCFS queuing policies. The experimental results show that under the SJF-like queuing policy, both regular and selective walltime adjustment schemes can improve performance in terms of average waiting time, slowdown, and average weighted waiting time. Moreover, using selective walltime adjustment with the 85th percentile, triple-key scheme, and one-month historical window outperforms all other schemes, achieving 22%, 22%, and 28% improvements on the studied three metrics, respectively. With FCFS, on the other hand, regular walltime adjustment does not improve the scheduling performance; however, using the selective schemes helps, achieving 20%, 22%, and 15% improvements on the three metrics.

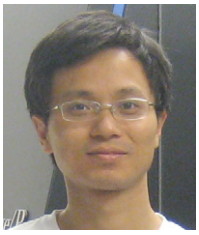
Therefore, we conclude that to achieve performance improvement by increased estimation accuracy, we need either to use a queuing policy favoring short jobs or not to use adjusted walltime estimates for running jobs.

Acknowledgments

This work was supported in part by National Science Foundation grants CNS-0834514, CNS-0720549, and CCF-0702737. The work at Argonne National Laboratory was supported by the US Department of Energy, under Contract DE-AC02-06CH11357. We gratefully acknowledge the use of the resources of the Argonne Leadership Computing Facility at Argonne National Laboratory.

References

[1] Blue Gene Team, Overview of the IBM Blue Gene/P project, IBM Journal of Research and Development 52 (1–2) (2008) 199–220.
[2] S.-H. Chiang, A. Arpaci-Dusseau, M. Vernon, The impact of more accurate requested runtimes on production job scheduling performance, in: Proc. of Job Scheduling Strategies for Parallel Processing, 2002.
[3] W. Cirne, F. Berman, A comprehensive model of the supercomputer workload, in: Proc. of IEEE International Workshop on Workload Characterization, 2001.
[4] W. Cirne, F. Berman, Using moldability to improve the performance of supercomputer jobs, Journal of Parallel and Distributed Computing 62 (10) (2002) 1571–1601.
[5] Cobalt project. http://trac.mcs.anl.gov/projects/cobalt.
[6] N. Desai, D. Buntinas, D. Buettner, P. Balaji, A. Chan, Improving resource availability by relaxing network allocation constraints on the Blue Gene/P, in: Proc. of International Conference on Parallel Processing, 2009.
[7] A. Downey, Predicting queue times on space-sharing parallel computers, in: Proc. of IEEE International Parallel Processing Symposium, 1997.
[8] R. Gibbons, A historical application profiler for use by parallel schedulers, in: Proc. of Job Scheduling Strategies for Parallel Processing, 1997.
[9] C. Glansner, J. Volkert, An architecture for an adaptive run-time prediction system, in: International Symposium on Parallel and Distributed Computing, 2008.
[10] C. Glansner, J. Volkert, A three-phase adaptive prediction system of run-time of jobs based on user behaviour, in: International Conference on Complex, Intelligent and Software Intensive Systems, 2009.
[11] N.H. Kapadia, J.A.B. Fortes, C.E. Brodley, Predictive application-performance modeling in a computational grid environment, in: Proc. IEEE Intl Symp. High Performance Distributed Computing, HPDC, August 1999, p. 6.
[12] S. Krishnaswamy, S. Loke, A. Zaslavsky, Estimating computation times of data-intensive applications, IEEE Distributed Systems Online 5 (4) (2004).
[13] C. Lee, Y. Schwartzman, J. Hardy, A. Snavely, Are user runtime estimates inherently inaccurate? in: Proc. of Job Scheduling Strategies for Parallel Processing, 2004.
[14] H. Li, D. Groep, J. Templon, L. Wolters, Predicting job start times on clusters, in: Proc. Sixth IEEE Intl Symp. Cluster Computing and the Grid, May 2004.
[15] A. Mu'alem, 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) (2001) 529–543.
[16] Parallel workload archive. http://www.cs.huji.ac.il/labs/parallel/workload/.
[17] D. Perkovic, P. Keleher, Randomization, speculation, and adaptation in batch schedulers, in: IEEE/ACM Supercomputing Conference, 2000.
[18] W. Smith, Prediction services for distributed computing, in: Proc. of IEEE International Parallel and Distributed Processing Symposium, 2007.
[19] W. Smith, I. Foster, V. Taylor, Predicting application runtimes with historical information, Journal of Parallel and Distributed Computing 64 (9) (2004) 1007–1016.
[20] W. Smith, V. Taylor, 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.
[21] S. Srinivasan, R. Kettimuthu, V. Subramani, P. Sadayappan, Characterization of backfilling strategies for parallel job scheduling, in: Proc. of the International Conference on Parallel Processing Workshops, 2002.
[22] W. Tang, N. Desai, D. Buettner, Z. Lan, Analyzing and adjusting user runtime estimates to improve job scheduling on the Blue Gene/P, in: Proc. of IEEE International Parallel and Distributed Processing Symposium, 2010.
[23] W. Tang, Z. Lan, N. Desai, D. Buettner, Fault-aware, utility-based job scheduling on Blue Gene/P systems, in: Proc. of IEEE International Conference on Cluster Computing, 2009.
[24] W. Tang, Z. Lan, N. Desai, D. Buettner, Y. Yu, Reducing fragmentation on torus-connected supercomputers, in: Proc. of IEEE International Parallel and Distributed Processing Symposium, 2011.
[25] TOP500 Supercomputing web site. http://www.top500.org.
[26] D. Tsafrir, Using inaccurate estimates accurately, in: Proc. of Job Scheduling Strategies for Parallel Processing, 2010.
[27] D. Tsafrir, Y. Etsion, D. Feitelson, Modeling user runtime estimates, in: Proc. of Job Scheduling Strategies for Parallel Processing, 2005.
[28] D. Tsafrir, Y. Etsion, D. Feitelson, Backfilling using system-generated predictions rather than user runtime estimates, IEEE Transactions on Parallel and Distributed Systems 18 (6) (2007) 789–803.
[29] D. Tsafrir, 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.
[30] W. Ward, C. Mahood, J. West, Scheduling jobs on parallel systems using a relaxed backfill strategy, in: Proc. of Job Scheduling Strategies for Parallel Processing, 2002.
[31] Y. Wu, K. Hwang, Y. Yuan, W. Zheng, Adaptive workload prediction of grid performance in confidence windows, IEEE Transactions on Parallel and Distributed Systems 21 (7) (2010) 925–938.
[32] Y. Yuan, G. Yang, Y. Wu, W. Zheng, PV-EASY: a strict fairness guaranteed and prediction enabled scheduler in parallel job scheduling, in: Proc. ACM Intl Symp. High Performance Distributed Computing, 2010, pp. 240–251.
[33] Y. Zhang, H. Franke, J. Moreira, A. Sivasubramaniam, Improving parallel job scheduling by combining gang scheduling and backfilling techniques, in: Proc. of IEEE International Parallel and Distributed Processing Symposium, 2000.
[34] D. Zotkin, P. Keleher, Job-length estimation and performance in backfilling schedulers, in: Proc. of IEEE International Symposium on High Performance Distributed Computing, 1999.



Wei Tang received his B.S. degree from Hefei University of Technology in 2002 and M.S. degree from Huazhong University of Science and Technology in 2005, both in Computer Science. He is currently a Ph.D. student in the Department of Computer Science at Illinois Institute of Technology. His research interests include parallel and distributed computing, job scheduling and resource management in large scale systems, etc.

Narayan Desai is currently a Principle Experimental System Engineer in the Mathematics and Computer Science Division at Argonne National Laboratory. His research interests include high-performance computing, resource management and job scheduling, etc.

Daniel Buettner was a Senior Software Specialist in the Argonne Leadership Computing Facilities at Argonne National Laboratory. He was one of the major developers working on the Cobalt resource manager project.

Zhiling Lan received her B.S. degree in Mathematics from Beijing Normal University in 1992, her M.S. degree in Applied Mathematics from the Chinese Academy of Sciences in 1995, and her Ph.D. in Computer Engineering from Northwestern University in 2002. She is currently an associate professor of computer science at the Illinois Institute of Technology. Her research interests are in the area of parallel and distributed systems, in particular, fault tolerance, dynamic load balancing, and performance analysis and modeling.