Walltime Prediction and its Impact on Job Scheduling Performance and Predictability

Dalibor Klusáček[1] and Mehmet Soysal[2]

[1] CESNET a.l.e., Brno, Czech Republic
klusacek@cesnet.cz
[2] Steinbuch Centre for Computing,
Karlsruhe Institute of Technology, Germany
mehmet.soysal@kit.edu

Abstract. For more than two decades researchers have been analyzing the impact of inaccurate job walltime (runtime) estimates on the performance of job scheduling algorithms, especially the backfilling. In this paper, we extend these existing works by focusing on the overall impact that improved walltime estimates have both on *job scheduling performance* and *predictability*. For this purpose, we evaluate such impact in several steps. First, we present a simple walltime predictor and analyze its accuracy with respect to original user walltime estimates captured in real-life workload traces. Next, we use these traces and a simulator to see what is the impact of improved estimates on general performance (backfilling ratio and wait time) as well as predictability. We show that even simple predictor can significantly decrease user-based errors in runtime estimates, while also slightly improving job wait times and backfilling ratio. Concerning predictions, we show that walltime predictor significantly decreases errors in job wait time forecasting while having little effect on the ability of the scheduler to provide solid advance predictions which nodes will be used by a given waiting job.

Keywords: job, scheduling, backfilling, walltime estimate, prediction

1 Introduction

This paper is focusing on the problem of inaccurate job runtime estimates as provided by users. We use existing results [9, 6, 12] and we try to understand the impact that inaccuracy has on various aspects of job scheduling performance. Importantly, we study whether a technique improving runtime estimates has some significant impact on system's behavior. For this purpose, we use a simple runtime predictor which we have developed on our own. This predictor uses historic data to generate runtime estimates for newly arriving jobs. As we have learned recently, this predictor uses similar idea to the predictor used in the past [11].

The main contribution of this paper is as follows. First, we demonstrate that even simple predictor can significantly improve inaccurate job runtime estimates

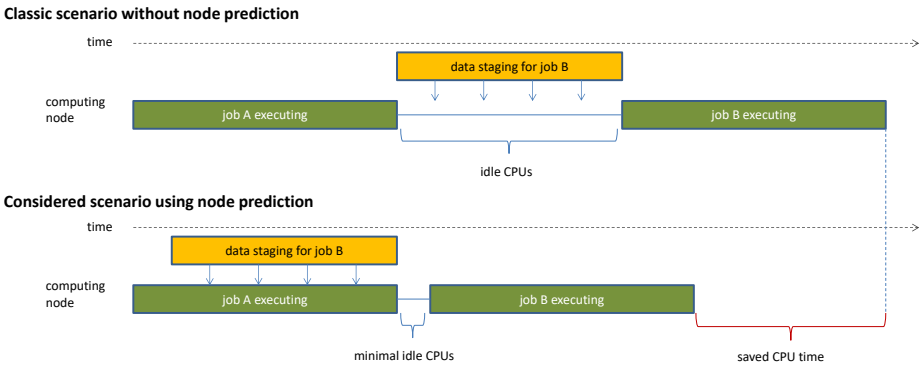

Fig. 1. Normal scenario for data staging is wasting CPU cycles during data staging (top). Considered solution relying on accurate node predictions uses advance data staging onto compute node(s) before job start, reducing idle CPU time (bottom).

(Section 2). Second, we use detailed simulations to analyze the impact of refined estimates on the job scheduling performance. We analyze how the improved estimates impact the number of backfilled jobs and job wait times (Section 3). Last but not least, in Section 4 we analyze deeply if refined runtime estimates can improve predictability of system behavior. To achieve this goal we use two different scenarios. In the first one, we analyze the accuracy of job waiting time predictions (Section 4.1). This scenario obviously focuses on system users that naturally want to know how long their jobs will have to wait before being processed by the system, i.e., here the question is "when will a job start?". In the second scenario (Section 4.2) we analyze the ability of the scheduler to correctly predict (in advance) which node(s) will be selected for each waiting job. In this case, instead of focusing on the question "when?" we rather try to answer the question "where?".

The motivation here is related to jobs requiring either large amount of data and/or jobs requiring special pre-processing, e.g., an ad hoc and independent local file system. In both cases, the time needed to either stage the data and/or setup the file system can cause temporarily low CPU utilization. Our goal is to determine whether it is possible to stage the data and/or deploy the local file system in advance, thus limiting idle CPU time. Obviously, to make this advance staging/setup possible, the scheduler must provide rather accurate advance job allocations (ahead of actual job start). We illustrate the benefit of this approach in Figure 1 and later describe in full detail in Section 4.2.

2 Job Walltimes, User Estimates and Predictor

In this paper, job walltime (or job runtime) denotes the time it takes to execute the job on a computing node(s). This time is not known in advance. Instead, user is requested to provide an estimate for each job. This walltime estimate

is used as an upper bound by the resource manager, i.e., job is killed when its actual runtime exceeds the walltime estimate. It is not surprising that in practice these walltime estimates are therefore very inaccurate and overestimated in order to prevent jobs from being killed [3, 7]. This causes the relatively high overestimation. Second, scheduling systems also frequently classify jobs according to some default runtime limits. For example, there can be different job queues with different maximum job runtime defaults. Frequently, these default runtime values are then used by many jobs due to users laziness. As a result, most jobs in the system use only few common estimates and therefore "look similar" to the scheduling algorithm (e.g., backfilling).

2.1 Workload Traces

In the remainder of this paper we will be using four different real-life workload traces that come from the Karlsruhe Institue of Technology in Germany (FH1 and FH2), Cornell Theory Center (CTC SP2) and San Diego Supercomputer Center (SDSC SP2). These traces can be obtained at Parallel Workloads Archive [2]. We begin our analysis by showing how user-based estimates are inaccurate and overestimated in all four considered workloads. This is captured in Figure 2, which shows the cumulative distribution functions (CDF) of actual (blue line) and user estimated (red line) job walltimes.

Clearly, user-provided walltime estimates are (very) inaccurate and conservative, i.e., overestimated. Therefore, we introduce a simple walltime predictor, which tries to refine these overly long estimates by more accurate values.

2.2 Walltime Predictor and its Performance

The considered walltime predictor is working on a per-user basis, i.e., a new runtime estimate for a given job of a user is computed using information about previous jobs of that user[1].

The predictor is an extended version of the predictor used in our previous work [4]. It measures the fraction of job's actual runtime and user's estimate (see $usage_{wall}$ in Formula 1), i.e., it measures to what extent the estimated walltime ($est\_walltime$) was actually used. Since the user's estimate is the upper bound of job runtime, $usage_{wall}$ falls between 0.0 and 1.0 representing the relative usage of requested walltime. In other words, the technique measures by how much a user overestimates job's runtime, which is similar to what has been used in [11].

$$usage_{wall}(job_i) = \frac{runtime(job_i)}{est\_walltime(job_i)} \tag{1}$$

$$predicted\_wall(job_i) = est\_walltime(job_i) \cdot \max_{i-5 \leq k \leq i-1} usage_{wall}(job_k) \tag{2}$$

[1] In case that a given user has no completed jobs so far then such historic information is obviously missing, thus we use the user-provided estimate instead.

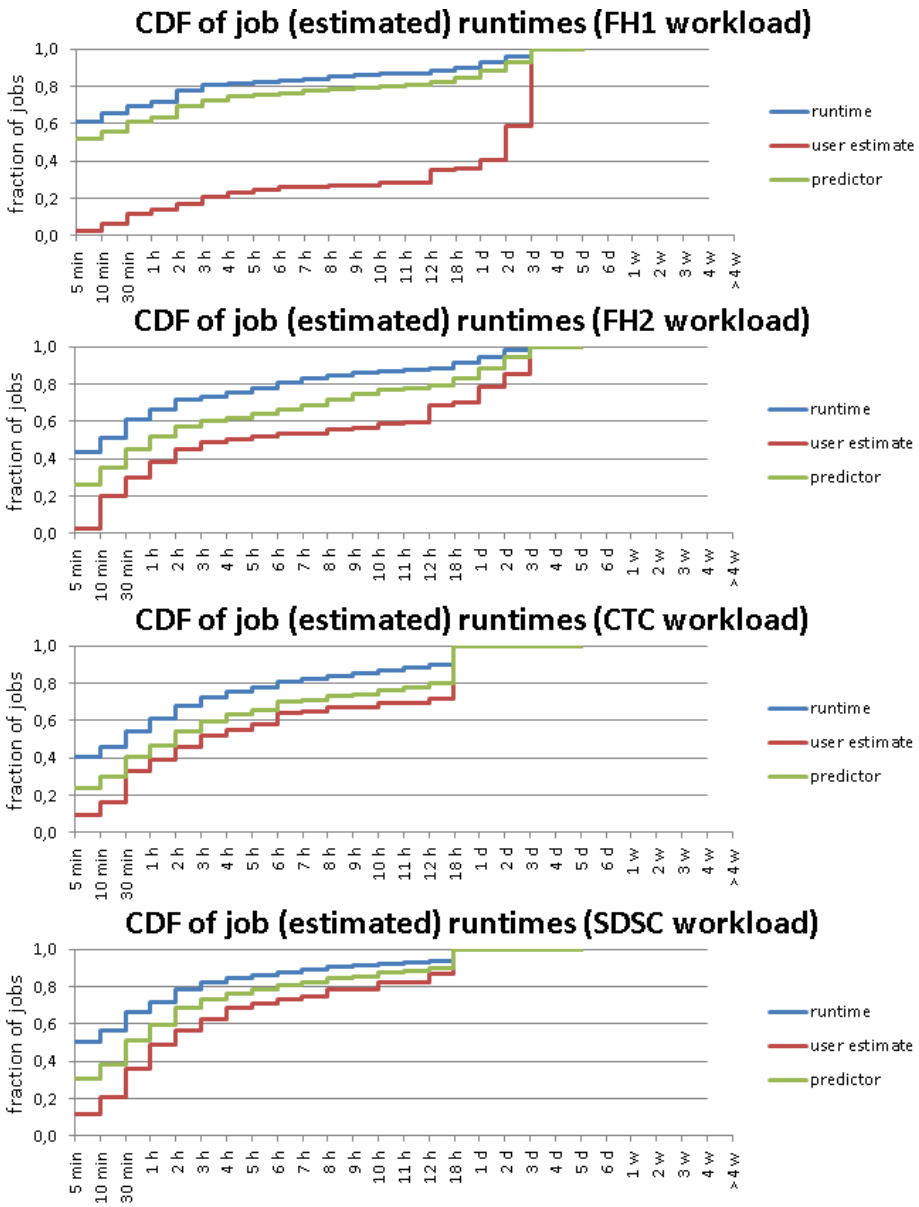

Fig. 2. Cumulative distribution functions (CDF) of actual, user estimated and predicted job walltimes for all four workloads.

Once the $usage_{wall}$ is computed, it is stored to be used in the future, i.e., once a new job of this user arrives in the system. When this happens, the five most recent $usage_{wall}$ values are considered and their maximum is cho-

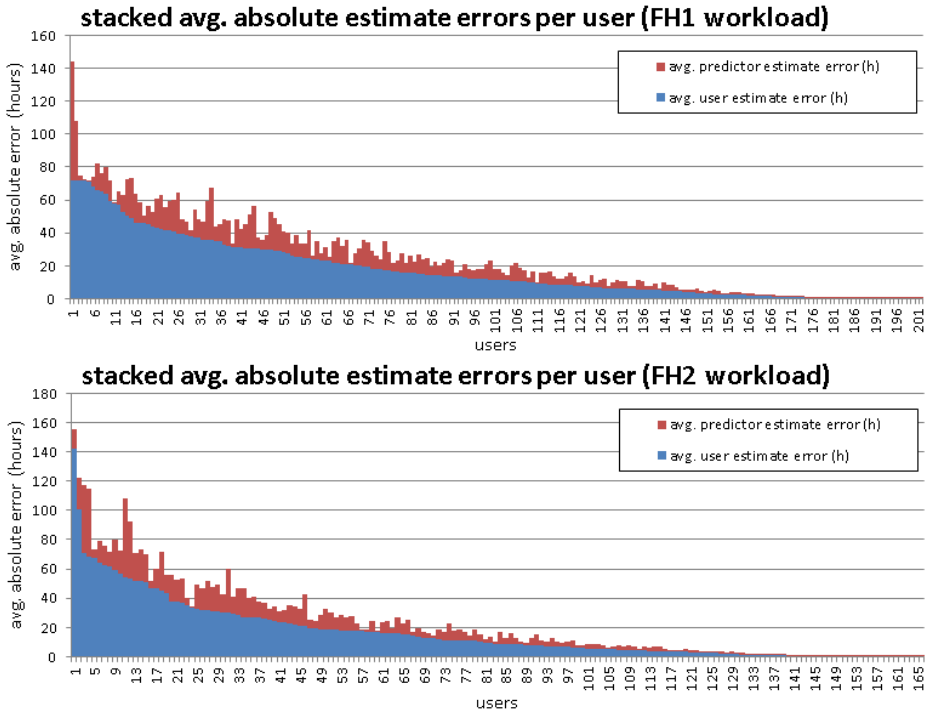

Fig. 3. Stacked chart of avg. absolute errors of runtime predictions (per user) with respect to the walltime being used (user estimate, predictor).

sen. The job's walltime estimate is then multiplied by this maximum (see Formula 2) and the resulting *predicted_wall* is the predictor's output. It represents a conservative strategy, where the predicted walltime is calculated using the known relative accuracy of user's recent estimates. By choosing the maximum $usage_{wall}$ (i.e., choosing a job where the difference between actual and estimated runtime was minimal), this technique aims to minimize the number of cases where the new predicted walltime will be underestimated. At the same time, by ignoring older jobs it reflects aging and orients itself more on the recent user's workload characteristics. When a job turns out to be underestimated ($predicted\_wall(job_i) < runtime(job_i)$) the predictor increases its initial estimate by a factor of two. It does so (over the time) until the estimate is sufficient (and job completes) or until the estimate meets the original user estimate (this is still a hard limit which cannot be exceeded).

The impact of the predictor can be observed in Figure 2 (green line). Clearly, the over-estimated user-based walltimes are now distributed closely to what is the real distribution of actual runtimes (blue line). The improving effect is especially visible when the user-based estimates are very bad, which is the case for the FH1 and also FH2 workloads.

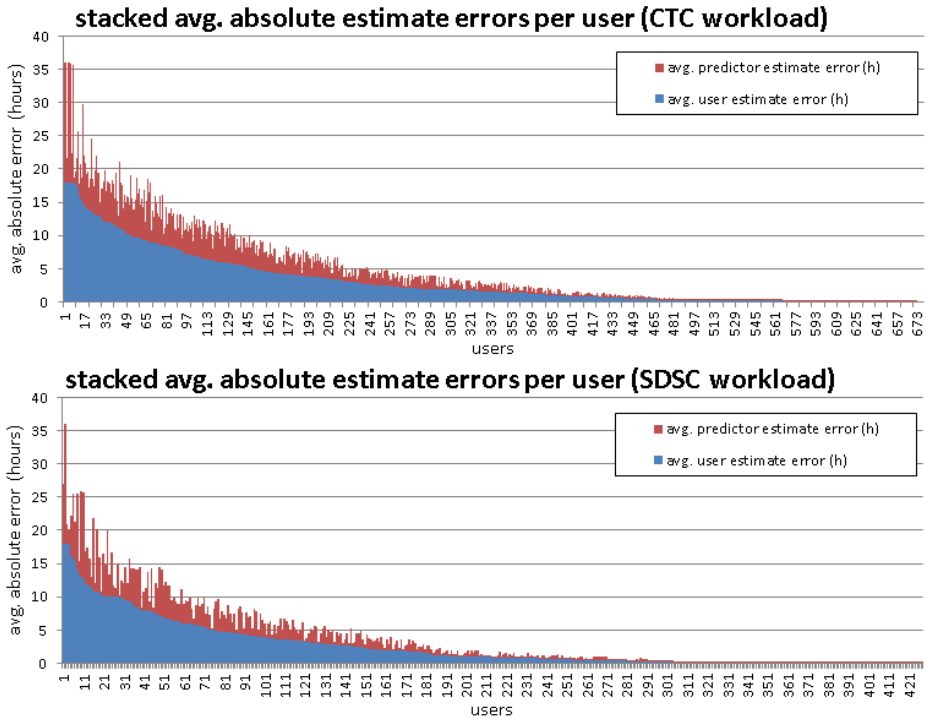

Fig. 4. Stacked chart of avg. absolute errors of runtime predictions (per user) with respect to the walltime being used (user estimate, predictor).

We also analyzed, how our simple predictor performs with respect to *individual users*. Therefore, we have computed the average absolute errors of both user-based estimates and predictor-based walltimes per each user in the system. Then we plotted the results in a stacked chart which we show in Figures 3–4.

These figures illustrate that for some users our simple predictor is not able to reduce the average error very well. Still, it decreases the errors compared to those user-based walltimes quite successfully. Clearly, some users are really poor in judging the duration of their jobs, providing estimates that are (on average) several hours or even days longer than necessary. From this point of view, even a simple predictor like the one we presented makes a good sense to use.

3 Impact on Job Scheduling Performance

So far, we have demonstrated using several existing workload traces that predictor can improve the accuracy of walltime estimates. This section uses detailed simulations to analyze the impact of walltime predictor on the performance of the system. For this purpose, each of those four systems is modeled in Alea simulator [5] and the workload is replayed. For comparison, simulations use either

perfect estimates (i.e., exact job runtimes are used), user-provided or predictor-generated estimates to build the job schedule, respectively.

3.1 Scheduling Policy

In all experiments we use *conservative backfilling* [7] as it heavily relies on provided estimates. The schedule is built using available runtime estimates (perfect, user-provided or predictor-based, respectively). In case a job finishes earlier than expected, the schedule is updated using schedule compression algorithm [7]. During the update, jobs are checked one by one and a start time of each job is adjusted, i.e., it is moved into the earliest possible time slot with respect to previously adjusted jobs (compression phase). Similarly, if a job's runtime is underestimated, it is first prolonged (see the discussion in Section 2.2) and then the existing schedule is updated, i.e., jobs are reinserted using same mechanism as during the aforementioned compression phase.

3.2 Metrics and Results

As our performance indicators we measure the percentage of *backfilled jobs* and the distribution of job *wait times*. Let us first discuss the impact of improved estimates on the backfilling ratio. The intuition suggests that with more accurate estimates (and even more with perfectly known runtimes) the ratio of backfilled jobs should increase. Conversely, when using inaccurate, user-provided and overestimated walltimes, the backfilling ratio should be significantly lower since most jobs "appear to be too long" for existing gaps. The results we obtained (see Figure 5) seem to follow this expectation but the differences are not very dramatic.

Figure 5 shows the percentage of backfilled jobs with respect to the accuracy of estimates being used. With the exception of FH1 workload, there is only

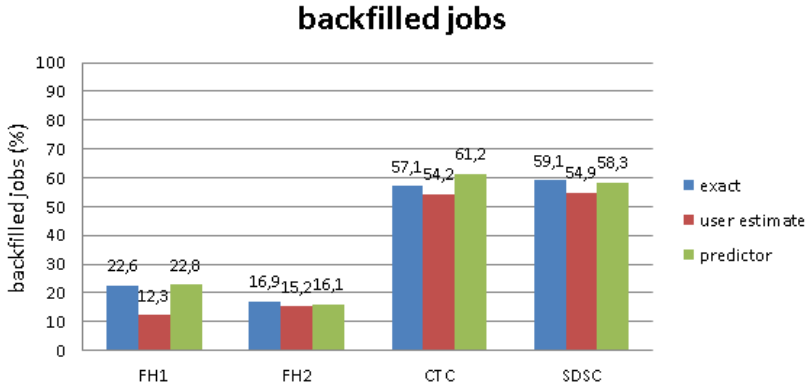

Fig. 5. The percentage of backfilled jobs with respect to the walltime being used (exact runtime, user estimate, predictor).

Fig. 6. Distribution of job wait times with respect to the walltime being used (exact runtime, user estimate, predictor).

slight improvement in the backfilling ratio when accurate or predictor-based walltimes are used. The difference between FH1 and remaining workloads is most likely caused by the very bad original user-based estimates available in FH1. As can be seen in Figure 2 (top), FH1 has the worst user-based estimates among all workloads. It is important to understand that improved walltimes do not guarantee higher backfill ratio. In fact, with better walltime estimates, not only individual jobs "look shorter" but also all available holes in the schedule become "shorter" as job runtimes are less overestimated. Therefore, the probability that a job will be backfilled within existing holes is only slightly higher when estimates are better[2].

We also measured the impact of improved walltime estimates on the distribution of job wait times which is captured in Figures 6–7. The main general

[2] With the exception of poor user-based estimates as shown in case of FH1 workload.

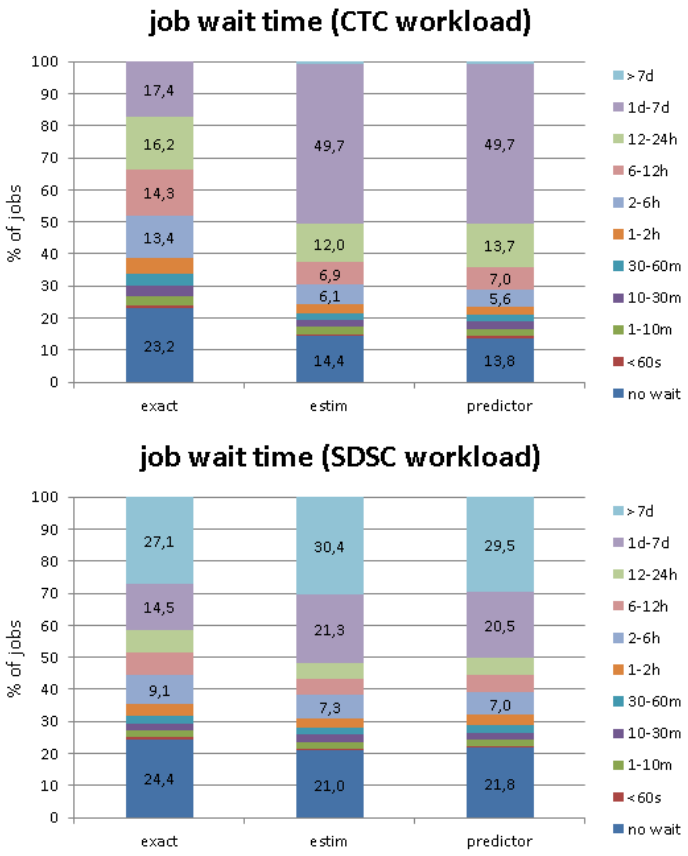

Fig. 7. Distribution of job wait times with respect to the walltime being used (exact runtime, user estimate, predictor).

difference between these two figures is the large amount of jobs that start immediately (see Figure 6). This is caused by the relatively smaller backlog of waiting jobs compared to the CTC and SDCS workloads (see Figure 7). Other than that, most of the workloads show that user-based inaccurate estimates are associated with worse distribution of job wait times, i.e., more jobs fall into categories representing long waiting. As soon as the predictor is used, we see a common tendency where job wait times are decreased.

The only exception is the CTC workload (see Figure 7 (top)), where the use of predictor does not produce better distribution of job wait times, instead it stays very close to the one dictated by user-provided estimates. This phenomenon has been observed in the past, e.g., in [1, 3] and deeply explained in [12] (using the same CTC workload trace). Apart from the explanation provided in [12], we would like to add that metrics like wait time can be easily influenced by even subtle changes in the job processing ordering. Job execution order can be

easily manipulated either purposely (e.g., via fair-sharing mechanism) or "accidentally", e.g., as a side effect of using the predictor. Predictor's estimates may shuffle the order in which jobs are executed as backfilling tends to prefer shorter jobs. Clearly, this *shortest job first*-like scheduling reduces average wait time.

Figure 8 shows a hypothetical example of three different schedules that are however composed of the same set of jobs. All three schedules have the same makespan ($C_{max} = 10$ time units) but exhibit very different job wait times. This figure illustrates the impact of job ordering in a somewhat extreme scale, yet we believe it illustrates nicely that metrics like wait time may oscillate quite wildly.

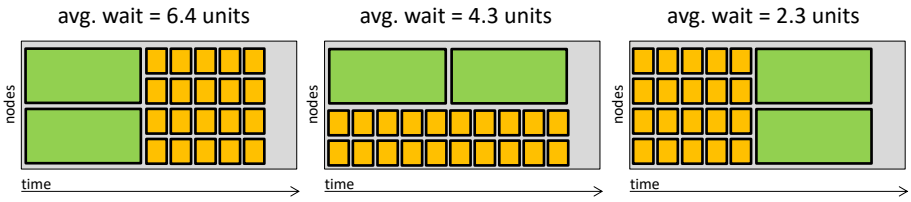

Fig. 8. The impact of job execution ordering on job wait times.

4 Impact on Accuracy of Predictions

This section is focusing on the ability of the scheduling system to provide predictions concerning job wait times and future job-to-node allocations. First, we analyze the accuracy of wait time predictions in Section 4.1. Next, Section 4.2 focuses on the impact that (in)accurate walltime estimates have on the capability of the job scheduler to provide advance node predictions for waiting jobs, i.e., the ability to correctly predict where a waiting job will be executed.

4.1 Wait Time Predictions

It is quite convenient when the scheduling system is able to provide information when a given job is likely to start executing. This is especially useful for interactive jobs. Although the system can use a scheduler-independent solution like QBETS [8], we use this section to analyze the accuracy of predictions that the scheduler can provide on its own. In this case, the scheduler is using the schedule (built by conservative backfilling) to estimate how long a job will wait before its execution will start. Since these wait times can be continuously refined (shortened) as jobs are completing earlier and the schedule is compressed, we use the initial estimate returned by the scheduler at the moment of job arrival. This can be seen as the first "response" the user gets when he or she submits the job. The accuracy of predictions is measured by computing the absolute error of predicted wait time with respect to the actual job wait time.

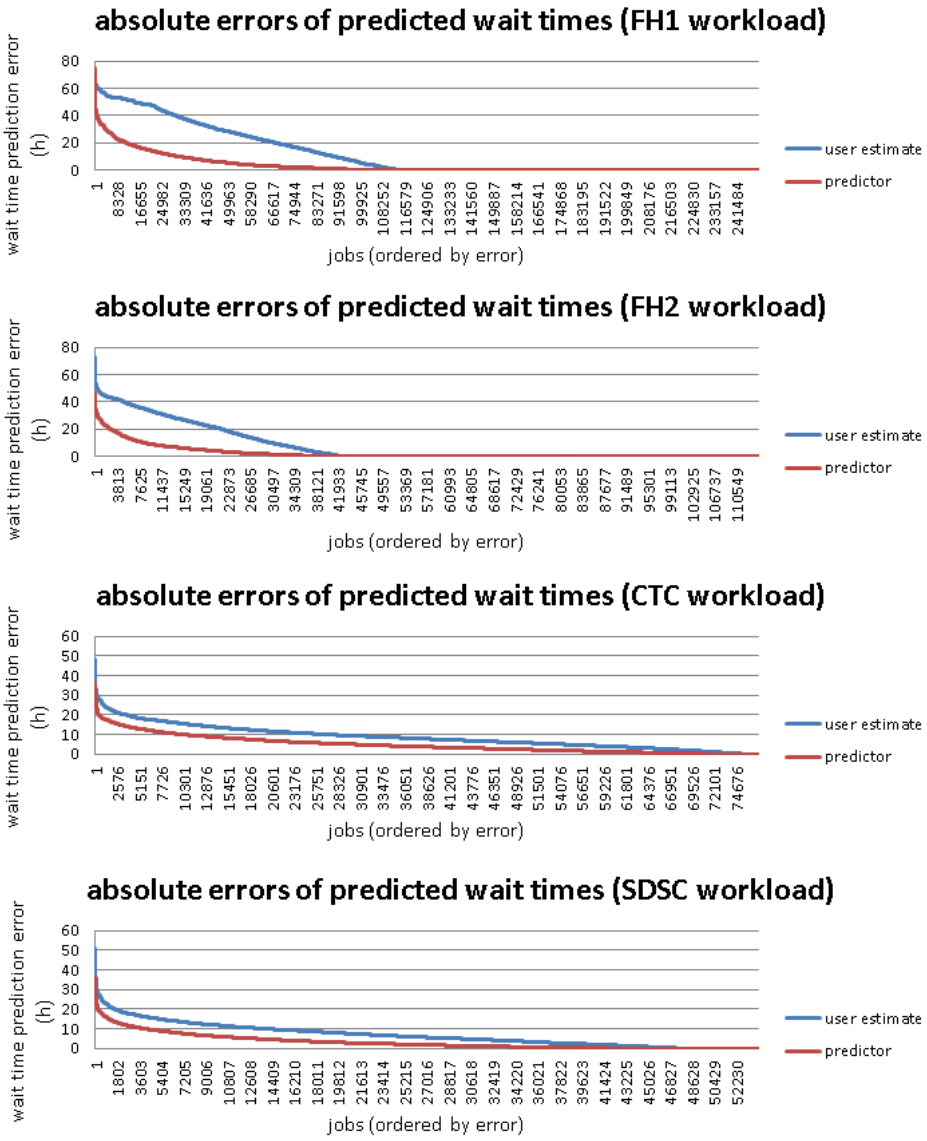

Fig. 9. Absolute errors in predicted wait times of jobs when using either user-estimated walltimes or predictor.

Figure 9 shows the absolute errors in predicted job wait times when using either user-provided walltime estimates or the predictor-based estimates. It nicely illustrates the ability of the predictor to decrease the prediction error. This is mostly visible in case of FH1 and FH2 workloads which have the worst user-based estimates (see Figure 2 and related discussion). From this point of

view, predictor-based estimates can deliver much better predictions of job waiting times, thus giving the users more optimistic responses concerning when their workloads will be executed.

4.2 Node Allocation Predictions

The ability to predict node(s) for waiting job in advance can be very useful. If the target node(s) is known for reasonably long time (e.g., at least a few minutes ahead) this time can be used to stage all job-related data in advance, thus saving CPU cycles. Also, if needed, local ad hoc file system can be setup for a waiting job in advance. We have illustrated this approach in Figure 1.

This section analyzes whether it is feasible to predict the target node(s) ahead. More specifically, we measure for how long the target node has been known prior job start. We called this time period *Valid Node Allocation Time Period* and denote it as T_{node}. To sum up, for each job the T_{node} is computed as the difference between job start time and the time when the final accurate prediction has been made (i.e., the one that was correct). In case a job starts immediately after its submission, T_{node} is equal to zero. Otherwise, the job is waiting and the scheduler is adding that job into the schedule. If perfect walltime estimates (exact runtimes) are used, the schedule being built is accurate and can be used to correctly predict target node(s) for jobs in advance. However, since job walltime estimates are inaccurate, the schedule is constantly adapting to jobs finishing at different times than previously predicted. This impedes the effort of the scheduler to provide reasonably accurate node predictions as planned node and time allocations can change virtually at any time. As we have shown in our earlier work [10], with inaccurate estimates only a small portion of jobs can achieve accurate predictions, i.e., only a fraction of jobs gets a reasonably high (useful) T_{node}.

T_{node} is also negatively influenced (reduced) by backfilling approach [10]. As demonstrated in Figure 10, jobs being backfilled can distort previous node allocations, thus decreasing T_{node} for those jobs being affected. Therefore, instead of backfilling, we use simple First Come First Served (FCFS) policy to built the job schedule in this use case. This means that existing holes in the schedule are

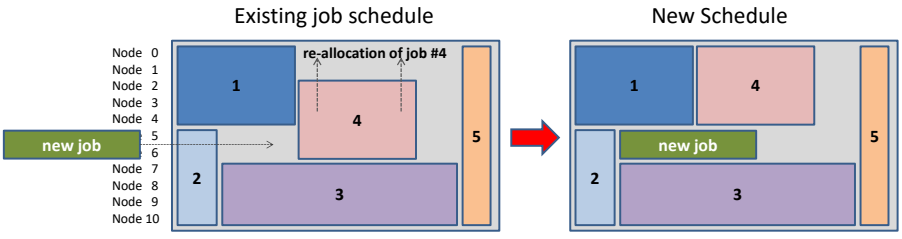

Fig. 10. An example of backfilling distorting previous node allocations for job #4 as a result of its "gap filling" approach.

not being filled with later arriving jobs in order to reduce the negative effect observed in backfilling. FCFS policy simply finds the earliest available free slot *at the end of the schedule*, without searching for possible gaps within the existing schedule. For example, if FCFS was used in the scenario presented in Figure 10 instead of backfilling, FCFS would place *new job* behind job #5 leaving all preceding gaps unused.

Moreover, we have extended the approach used in our previous paper [10] and developed a significant extension to the scheduling policy, which allows us to "pin" jobs to their planned nodes if their predicted start time is near enough. In other words, if a job is about to start soon (e.g., during 1 hour) we do not allow the scheduler to change the planned nodes for this job. Thus this job's node allocation is fixed and its T_{node} cannot decrease. Without node-pinning, jobs can be shifted to different nodes — e.g., as a result of early job completion and the following schedule compression procedure. This reshuffling of the whole schedule then destroys existing predictions while reseting all corresponding T_{node} values to zero. Therefore, pinning helps to keep predictions valid subject to inaccuracies in the job schedule. In our implementation, node-pinning is only activated when the planned job's start time is within a given "pinning interval" x, i.e., that job is planned to be executed within next x minutes.

In the following text, using a series of experiments, we analyze how the T_{node} values are distributed with respect to varying accuracies of walltime estimates (exact, user-estimated, predictor). We also measure what is the effect of the newly developed node-pinning functionality. For this purpose, we analyze the effect of either no node-pinning or pinning with short interval ($x = 1$ hour) or pinning with long interval ($x = 2$ hours). Since the absence of backfilling and the use of node-pinning can potentially lead to poor performance, we also measure this impact by comparing job wait times of all considered scenarios.

Figures 11–12 shows the durations of valid node predictions (distribution of T_{node} values) with respect to the pinning interval (none, 1 hour , 2 hours) and walltime being used (exact runtime, user estimate, predictor). From these distributions we can quickly identify the positive effect that node-pinning has on the ability of the scheduler to provide reasonable advance node predictions. On the other hand, the effect of predictor is rather moderate. It does show positive effect in case of FH1 workload (the one having worst user-based estimates) as it significantly increases the fraction of jobs that have $T_{node} > 0$ seconds. Moderate positive effect can be seen for FH2, CTC and SDSC workloads (comparing estim vs. predictor). However, in most cases the largest benefit is evidently achieved through node-pinning rather than through improved walltime estimates.

The node-pinning functionality developed in this work sadly has some obvious drawbacks. Since we do not use backfilling, nodes can easily become underutilized when waiting jobs are already pinned to different (busy) CPUs. To measure the impact, we provide Figures 13–14 that show wait time distributions corresponding to the experiments focusing on node predictability.

Clearly, the effect of FCFS policy (even increased by node-pinning) is heavily influencing the wait time distributions as can be seen by comparing these distri-

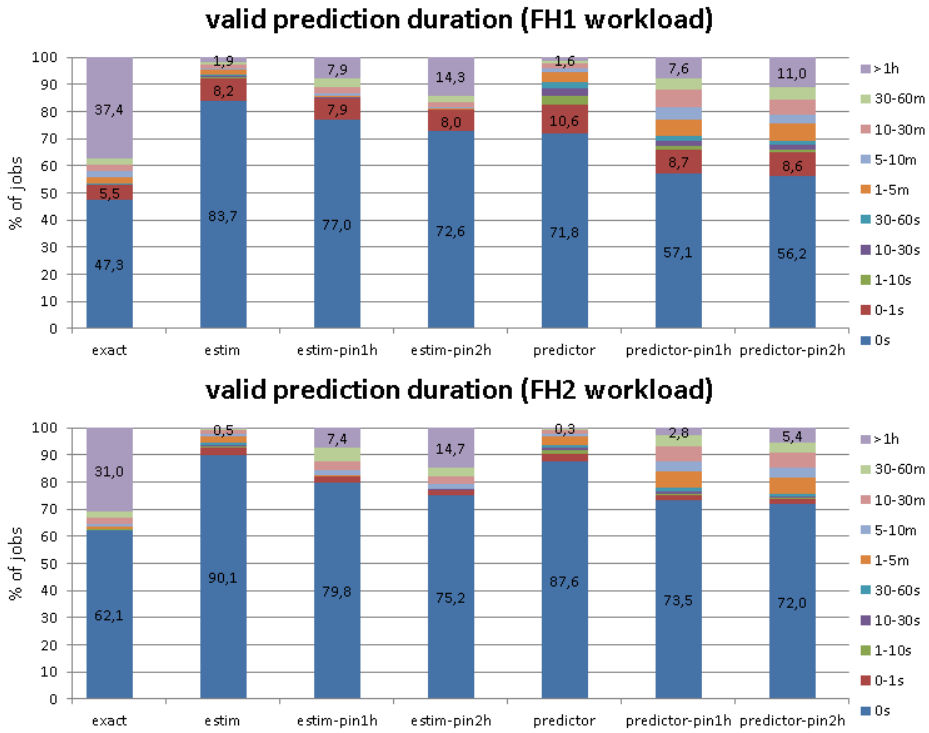

Fig. 11. Distribution of durations of valid node predictions (T_{node}) with respect to the pinning interval (none, 1h, 2h) and walltime being used (exact runtime, user estimate, predictor).

butions to those observed for "pure" backfilling in Figures 6–7. FCFS worsened waiting times while node-pinning added even more delays. Also, it is worth noticing that the accuracy of estimates plays smaller role in this use case since the FCFS policy is less dependent on the quality of estimates than conservative backfilling used in Section 3.

4.3 Summary

To sum this use case, we have shown that proposed node-pinning extensions of the FCFS policy along with the predictor help to increase node predictability (higher T_{node} values) but at the expense of deteriorated wait times and poorer utilization. This clearly is not a desirable outcome and it leaves the problem open for further research. At the same time, we have seen that when using accurate walltimes reasonable node predictions are achievable and for many jobs their T_{node} is sufficiently long to allow for advance data staging. There are several classes of computations where job runtimes can be predicted very accurately and advance data staging is very useful due to the large size of input data. One

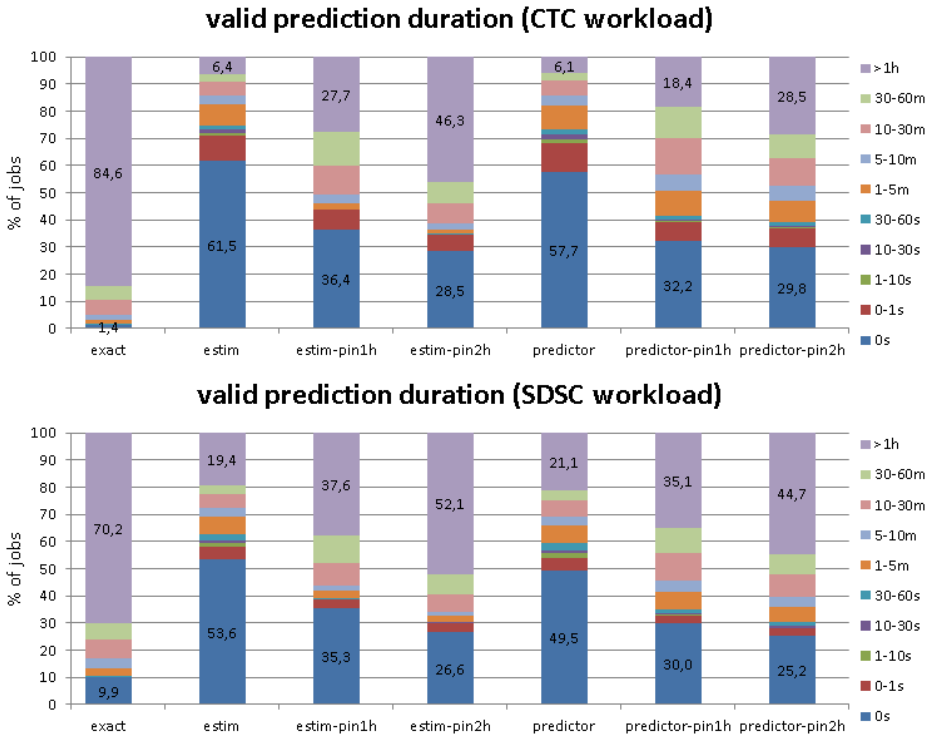

Fig. 12. Distribution of durations of valid node predictions (T_{node}) with respect to the pinning interval (none, 1h, 2h) and walltime being used (exact runtime, user estimate, predictor)

such type of computation represents, e.g., the processing (converting) of raw video data.

Also, not all jobs probably require such a special treatment, i.e., only truly data-demanding jobs benefit from advance data staging. Therefore, it would be interesting to pin only these data-heavy jobs and use the remaining jobs to efficiently "fill the holes", i.e., use selective backfilling. Our results show that the benefits of FCFS does not overweight its poorer performance.

5 Conclusion and Future Work

In this work we have studied walltime estimates and the use of a simple walltime predictor with respect to their impact on several different aspects of job scheduling. This analysis focused on the predictor's accuracy and the effect it has on system performance and predictability. Our main findings are summarized in the following list:

– Even simple predictor improves the accuracy of walltimes

Fig. 13. Distribution of job wait times with respect to the pinning interval and walltime being used.

- Better accuracy improves backfilling opportunities but the effect is not dramatic
- Wait time can be slightly reduced with better estimates (but other effects play role)
- Significant improvement can be achieved in wait time predictions
- With the existing predictor, node allocations cannot be predicted very well
- Better node predictions can be achieved by using heavily modified FCFS-based scheduler using "node-pinning", however job waiting times then deteriorate heavily

In the future, we want to implement more advanced predictors as well as test them in practice using the PBS Pro system being used in the Czech national computing infrastructure *MetaCentrum*. Also, we would like to further improve our node-pinning policy and introduce selective backfilling which should reduce the negative impact on job wait times and utilization.

Acknowledgments. We kindly acknowledge the support and computational resources provided by the MetaCentrum under the program LM2015042 and

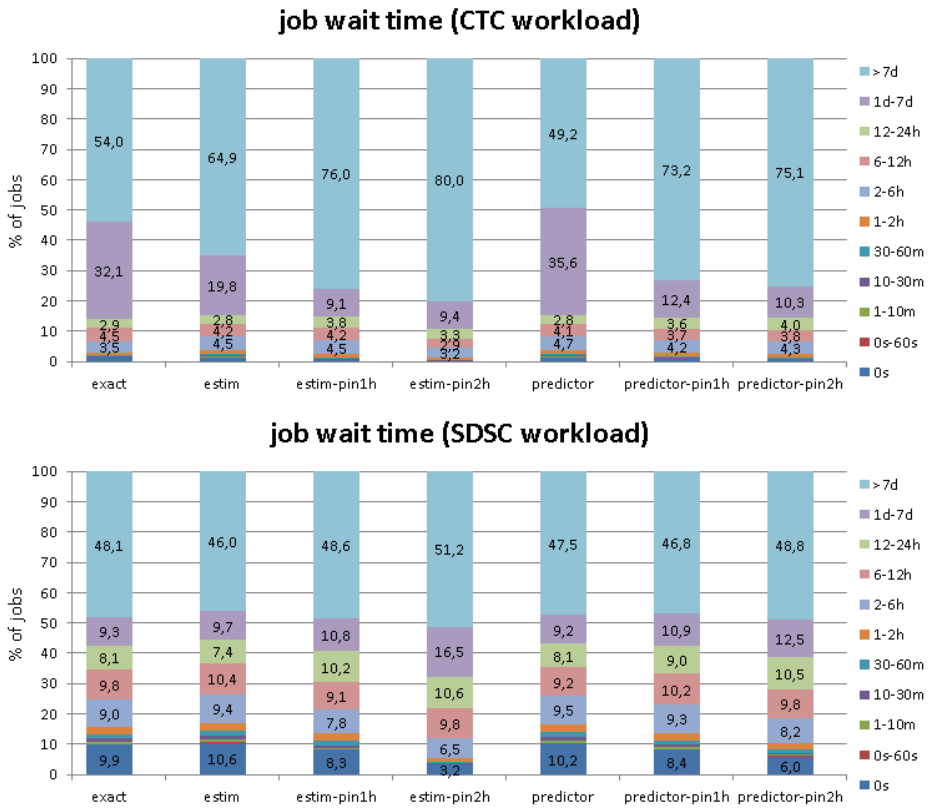

Fig. 14. Distribution of job wait times with respect to the pinning interval and walltime being used.

the CERIT Scientific Cloud under the program LM2015085, provided under the programme "Projects of Large Infrastructure for Research, Development, and Innovations" and the project Reg. No. CZ.02.1.01/0.0/0.0/16_013/0001797 co-funded by the Ministry of Education, Youth and Sports of the Czech Republic. We also highly appreciate the access to the workload traces provided by the Parallel Workloads Archive and the Karlsruhe Institute of Technology.

References

1. S.-H. Chiang, A. Arpaci-Dusseau, and M. K. Vernon. The impact of more accurate requested runtimes on production job scheduling performance. In D. G. Feitelson, L. Rudolph, and U. Schwiegelshohn, editors, *Job Scheduling Strategies for Parallel Processing*, volume 2537 of *LNCS*, pages 103–127. Springer Verlag, 2002.
2. D. G. Feitelson. Parallel workloads archive, February 2018. `http://www.cs.huji.ac.il/labs/parallel/workload/`.

3. D. G. Feitelson and A. M. Weil. Utilization and predictability in scheduling the IBM SP2 with backfilling. In *12th International Parallel Processing Symposium*, pages 542–546. IEEE, 1998.
4. D. Klusáček and V. Chlumský. Evaluating the Impact of Soft Walltimes on Job Scheduling Performance. In N. D. Dalibor Klusáček, Walfredo Cirne, editor, *Job Scheduling Strategies for Parallel Processing*, volume 11332 of *LNCS*, pages 15–38. Springer, 2018.
5. D. Klusáček, Šimon Tóth, and G. Podolníková. Complex job scheduling simulations with Alea 4. In *Ninth EAI International Conference on Simulation Tools and Techniques (SimuTools 2016)*, pages 124–129. ACM, 2016.
6. C. B. Lee, Y. Schwartzman, J. Hardy, and A. Snavely. Are user runtime estimates inherently inaccurate? In D. G. Feitelson, L. Rudolph, and U. Schwiegelshohn, editors, *Job Scheduling Strategies for Parallel Processing*, volume 3277 of *LNCS*, pages 253–263. Springer Verlag, 2004.
7. A. W. Mu'alem and D. G. Feitelson. Utilization, predictability, workloads, and user runtime estimates in scheduling the IBM SP2 with backfilling. *IEEE Transactions on Parallel and Distributed Systems*, 12(6):529–543, 2001.
8. D. Nurmi, J. Brevik, and R. Wolski. QBETS: Queue bounds estimation from time series. In E. Frachtenberg and U. Schwiegelshohn, editors, *Job Scheduling Strategies for Parallel Processing*, volume 4942 of *LNCS*, pages 76–101. Springer Verlag, 2007.
9. W. Smith, V. Taylor, and I. Foster. Using run-time predictions to estimate queue wait times and improve scheduler performance. In D. G. Feitelson and L. Rudolph, editors, *Job Scheduling Strategies for Parallel Processing*, volume 1659 of *LNCS*, pages 202–219. Springer Verlag, 1999.
10. M. Soysal, M. Berghoff, D. Klusáček, and A. Streit. On the quality of wall time estimates for resource allocation prediction. In *ICPP 2019: 48th International Conference on Parallel Processing: Workshops*, 2019.
11. W. Tang, N. Desai, D. Buettner, and Z. Lan. Analyzing and adjusting user runtime estimates to improve job scheduling on the Blue Gene/P. In *IEEE International Symposium on Parallel and Distributed Processing (IPDPS)*, pages 1–11. IEEE, 2010.
12. D. Tsafrir. Using inaccurate estimates accurately. In E. Frachtenberg and U. Schwiegelshohn, editors, *Job Scheduling Strategies for Parallel Processing*, volume 6253 of *LNCS*, pages 208–221. Springer Verlag, 2010.