Improving Accuracy of Walltime Estimates in PBS Professional using Soft Walltimes

Václav Chlumský and Dalibor Klusáček

CESNET a.l.e., Brno, Czech Republic
{vchlumsky,klusacek}@cesnet.cz

Abstract. Job walltime estimates are used by current batch schedulers to optimize the performance and predictability when scheduling parallel jobs on the computing resources. Since the user-provided estimates are inaccurate and often overestimated, system administrators often seek ways to improve them artificially using some form of walltime predictor. In this work, we present our real-life experience with deploying such a predictor using the *soft walltime* feature available in PBS Professional resource manager. Our results indicate that the applied solution is working properly, significantly increasing the accuracy of user-provided estimates. We share our experience when tuning the scheduler, discussing several problems that occurred along the way. Also, we provide a comparison of how the system behavior evolved once soft walltimes were deployed in production. Last but not least, we publish collected workload traces along with this paper to allow other researchers to further study and extend our work.

Keywords: job, scheduling, soft walltime, estimate, prediction, PBS

1 Introduction

This paper is addressing the problem of inaccurate user-provided job walltime estimates [12, 8, 7]. Inaccurate estimates cause holes to be left in the schedule during backfilling [10] as jobs appear to be too long for them to fit in. This may lead to a well known scenario, where only very short jobs can use the holes in the schedule, resulting in an SJF-like behavior (Shortest Job First) that can compromise other system's goals such as fair job ordering [16].

This paper is motivated by our real-life experience when maintaining the PBS Professional resource manager in the Czech national distributed computing infrastructure *MetaCentrum* [9]. As we have already discussed in our previous work [7], our users are no exception to the widely documented behavior of common HPC system users. They tend to use rather overestimated job walltime estimates in order to decrease the chance that their job will be prematurely killed due to running out of time. So the most common scenario is that users choose the maximum allowed runtime of a queue and then use it as the walltime estimate.

This well-known fact [16, 15] motivated several researchers to either develop some form of runtime prediction technique or find a significant incentive for individual users to improve the accuracy of their runtime requests [8, 2]. While the problem itself and various walltime prediction techniques have been studied extensively, we saw very little practical deployment of these techniques in practice. One of the reason was that mainstream resource managers did not provide tools to easily implement walltime predictors. This has changed in 2017 when PBS Professional introduced the so called *"soft walltime"* feature — a tool designed to simplify walltime predictor deployment [14].

In our earlier work [7], we used historic workload traces and a simulator to evaluate the impact of various walltime predictors in simulated HPC system. Those simulations were designed in order to give us a hint whether it is worth the effort to use the soft walltime in practice. Since the results were promising, we have decided to use soft walltimes generated by walltime predictor in our HPC system. This paper summarizes our current experience and lessons learned along the way.

The main contributions of this paper are following. In Section 2, we provide detailed description of our predictor and its integration in the HPC system. We discuss the evolution of the walltime predicting algorithm and — using real-life data — we illustrate problems that were observed and required our attention during the development. Section 3 analyzes the accuracy of the predictor in great detail, showing how even relatively simple predictor can improve the accuracy of job walltime estimates. Next, we analyze how the system performance has changed since the soft walltime has been deployed and used (Section 4). We believe that this paper is one of the first reports that documents the impact of soft walltimes in real environment. We conclude the paper in Section 5 and provide the developed predictor and all workload traces used in this study to the scheduling community.

2 Soft Walltime Adoption in CERIT-SC

In this paper we are using real data from the *CERIT-SC* partition of *MetaCentrum* infrastructure. CERIT-SC manages the second largest partition in our infrastructure [1]. It consists of 6,656 CPUs and it has its own instance of PBS server. This allowed us to use it as a "guinea-pig" in our efforts to introduce soft walltimes in MetaCentrum.

2.1 Inaccurate Estimates

To demonstrate the level of inaccuracy of user-provided estimates we present Figure 1 that shows how jobs are distributed according to their user-provided walltime and their actual runtime. Clearly, the curves are very different and do not match at all as users provide rather overestimated walltimes. This means that the scheduler is using very inaccurate data when constructing reservations

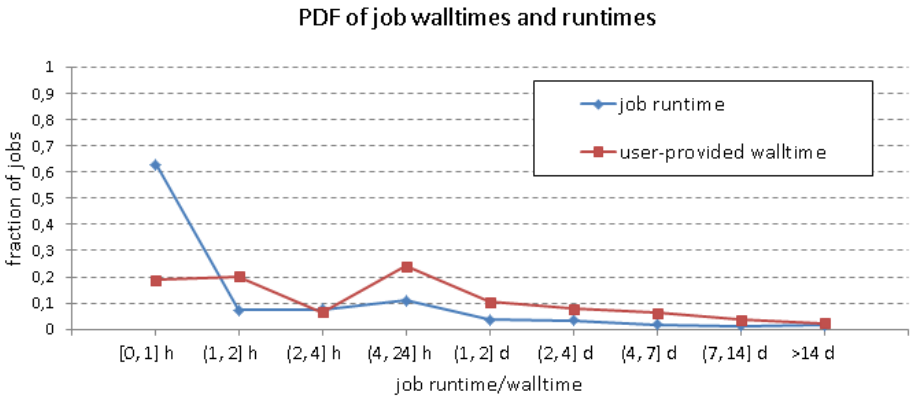

Fig. 1. Probability distribution function of user-provided walltimes and actual job runtimes. User-provided estimates are significantly overestimated.

for top jobs[1], predicting job start times, or trying to find small-enough jobs to be backfilled.

2.2 Soft Walltime Functionality

Soft walltimes in PBS are designed to safely *refine* user-provided job walltime (runtime) estimates. When enabled, the scheduler does not use user-provided estimates but instead uses so-called soft walltimes for all scheduling operations. Most importantly, it uses them to create job reservation(s) and perform backfilling. Soft walltimes are safe from the point of view of the user, because jobs are not killed when their soft walltimes are exceeded. As usual, a job is only killed when it exceeds its original, user-provided estimate. Unlike the walltimes which represent hard limit on job runtime, soft walltimes can be underestimated. In such situation the soft walltime is increased by a factor of two. This process can be repeated until either the job completes or the original walltime limit is reached. Underestimated soft walltimes should be avoided (if possible) because they can invalidate guaranteed start times for top jobs [13]. An important security feature is that soft walltimes cannot be (by default) specified or modified by users. Only the manager (system administrator) is allowed to set them up, typically using the so-called *job hook* script. This guarantees that users cannot obtain unfair priority in backfilling by providing very low (unrealistic) soft walltimes. More details on soft walltimes can be found in the PBS documentation [14, 13].

This paper is not focusing on discussing various runtime prediction techniques. Instead, we will proceed with the details of our solution and we kindly

[1] In PBS Professional, not every waiting job gets a reservation. Only a predefined number of high priority jobs (per queue) has guaranteed (latest) start times and these are called top jobs. Remaining jobs can be backfilled around top jobs provided they will not interfere with their reservations.

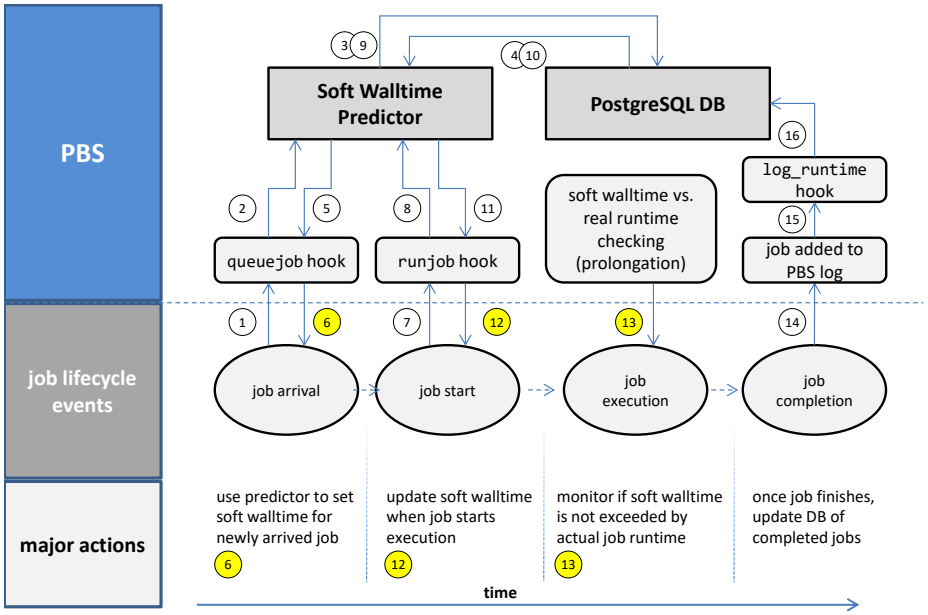

Fig. 2. The scheme of soft walltime generation used in CERIT-SC system.

refer to existing works that discuss various techniques for walltime refinement, e.g., the survey from Seneviratne and Witharana [11] or Soysal et al. [15].

2.3 Soft Walltime Predictor Implementation

The soft walltime feature was enabled by implementing a very simple predictor. The predictor uses a small PostgreSQL database that collect statistics about previously completed jobs of each user. The SQL database is hosted on the main server and all SQL-related operations are performed using PBS hooks. No operations are needed on the MoMs (execution nodes) due to security reasons (no SQL connections to the server-side database). As designed in PBS, soft walltime can be set either upon job arrival, job modification or when a job starts (using hook events: `queuejob`, `modifyjob` and `runjob`).

In our implementation, soft walltime is first generated upon each job arrival and is later updated when the job starts its execution. Once the job completes and is recorded in the PBS accounting log, we add its parameters into the PostgreSQL database to keep it up-to-date. Figure 2 depicts the current implementation and the main events + communication used to generate and maintain soft walltimes.

The predictor used for soft walltime generation is very simple and underwent two major upgrades since its first deployment in October 2021. The first version (v1) used simple arithmetic average of the last two previous runtimes of completed jobs of a given user [17]. This solution was used as a baseline

Algorithm 1 PREDICTOR_V1(*user*, *walltime*, *completed_jobs*)

```
1: previous_runtimes := SELECT j.runtime FROM completed_jobs j WHERE
   j.user_name=user ORDER BY j.job_end_time DESC limit 2;
2: if previous_runtimes = ∅ then
3:    soft_walltime := walltime;
4: else
5:    soft_walltime := MINIMUM(AVERAGE(previous_runtimes), walltime);
6: end if
7: return soft_walltime;
```

due to its simplicity and low computational requirements. The pseudo-code is shown in Algorithm 1. The predictor has three inputs: the user name (*user*), current job's walltime estimate (*walltime*) and the DB table of already completed jobs (*completed_jobs*). In the first step, the last two runtimes of completed jobs of that user are found in the database (line 1). If the user has no completed jobs yet ($previous\_runtimes = \emptyset$), the user-provided *walltime* is used as the $soft\_walltime$ (line 3). Otherwise, the average is computed and the new $soft\_walltime$ is the minimum of this average value and the user-provided *walltime* (line 5).

This predictor was used for roughly three weeks and its performance was then analyzed. Although it reduced absolute estimate errors significantly, it had one major flaw which was obvious and critical — it generated too many predictions that were underestimated. This was a big problem because it meant that the scheduler had to perform many soft walltime prolongations (see Section 2.2) which implied that existing reservations previously computed by the scheduler (using overly optimistic soft walltimes) were not valid and top jobs were delayed. We performed detailed analysis and realized that *by adding a fixed reserve* to every generated soft walltime (+15 minutes) we should be able solve this problem in most cases. This simple modification was added to the predictor at the beginning of November 2021.

Although this modification reduced the number of underestimated soft walltimes, it only worked for users with fairly stable job runtimes. Sadly, there are users in our system with highly spread job runtimes. Notably, we are dealing with users that have highly varying runtimes within a single batch of jobs[2]. To demonstrate this problem we provide Figure 3 that shows every job submission of a given user within the first week of November 2021. Vertical axis represents job runtime while the horizontal axis is the job submission time.

It is clear, that with such highly spread runtime distribution it makes no sense to use simple arithmetic mean to "guess" next job's runtime. This finding motivated us to develop a more robust predictor. For this purpose, we have developed a simple event-driven simulator which emulated job submissions and completions in the system and allowed us to test the accuracy of various pre-

[2] In this context, job batch is the set of jobs submitted into the system by a given user in a short time frame, e.g., during few minutes.

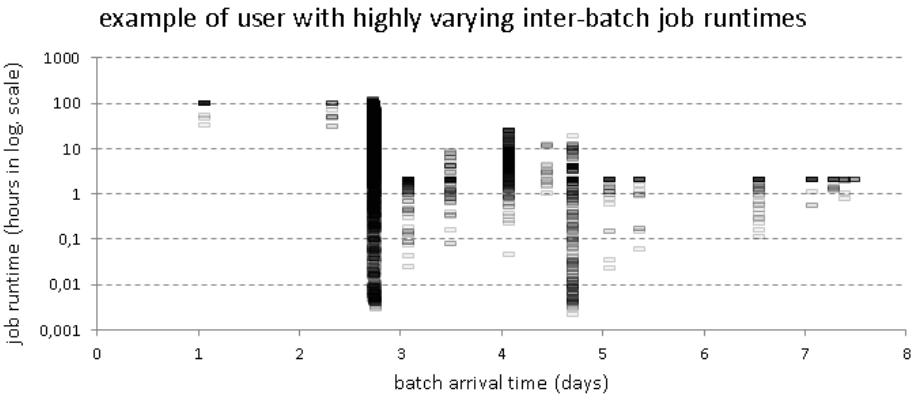

Fig. 3. Varying job runtimes of several job batches submitted during one week by a given user.

Algorithm 2 PREDICTOR_V3(*user*, *walltime*, *completed_jobs*)

1: *walltime_usages* := SELECT (j.runtime/j.job_walltime) as walltime_usage FROM *completed_jobs j* WHERE j.user_name=*user* ORDER BY j.job_end_time DESC limit 15;
2: **if** $walltime\_usages = \emptyset$ **then**
3: $soft\_walltime := walltime$;
4: **else**
5: $max\_walltime\_usage :=$ MAXIMUM($walltime\_usages$);
6: $predicted\_walltime := (max\_walltime\_usage \cdot walltime) + 900$;
7: $soft\_walltime :=$ MINIMUM($predicted\_walltime$, $walltime$);
8: **end if**
9: **return** $soft\_walltime$;

dictors. The final solution abandoned the original average runtime and instead focuses on the recent relative walltime usage. It is represented by Algorithm 2 and works as follows.

First, we compute so called *walltime_usage* ratio for the 15 most recently completed jobs. The *walltime_usage* is computed as the ratio of job runtime to its walltime ($j.runtime/j.job\_walltime$). The *walltime_usages* list is used to collect these ratios (line 1). Next, we choose the maximum of these values. The maximum relative walltime usage is then used to multiply the user-provided job walltime (line 6) producing the *predicted_walltime* (which includes 900 seconds corresponding to the 15 minutes-long reserve). It represents a conservative strategy, where the prediction is calculated using the known relative accuracy of user's recent estimates. By choosing the *max_walltime_usage* (i.e., by 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 soft walltime will be underestimated. At the same time, since the predictor only uses 15

recent jobs it reflect aging and orients itself more on the recent user's workload characteristics. With this approach we were able to reasonably cover the problems illustrated in Figure 3. The final $soft\_walltime$ is again computed as the minimum of the *predicted_walltime* and the user-provided *walltime* (line 7).

3 Comparison of Soft Walltime Predictors

In the following text we will summarize our findings that were collected during the development and deployment of our predictors in CERIT-SC system.

3.1 Initial Evaluation and Applied Modifications

We start with the results that were obtained by the event-driven simulator mentioned in the previous section. This simulator used real workload and realistically emulated job submissions and completions. Therefore, we were able to replay job arrivals and test all three variants of our predictor and compare their accuracy. Figure 4 illustrates the development of our predictor (v1, v2 and v3 variants). In this figure, we show absolute estimate errors (hours in log. scale) with respect to the used predictor. As a reference we also include errors of the original user-provided estimates (walltime). In order to distinguish between under- and overestimation, we first order errors in the increasing order for each used predictor. Next, we compute absolute values of these (ordered) errors and plot them using the log. scale. The resulting curves thus have a typical "V"-shape where the left part (decreasing) represents absolute values of negative errors (i.e., underestimated predictions) and the right part (increasing) corresponds to the overestimations (positive errors).

Clearly, original walltimes are never underestimated since they represent the upper bound of allowed runtime. From this experiment, which covers 67K jobs, we can see that the initial average-based predictor (predictor_v1) generated a lot of underestimated soft walltimes (over 40% of jobs were underestimated). The addition of 15 minute reserve (predictor_v2) reduces this unwanted situation, yet still nearly 31% of all jobs are underestimated. The predictor based on relative walltime usage (predictor_v3) produced the best results (less than 12% jobs is underestimated). While not so critical, the negative yet natural effect of predictor_v3 is the fact that jobs are typically quite overestimated with respect to predictor_v2 and v1.

3.2 Analysis of Soft Walltime Accuracy

Unlike the previous experiment which used identical workload and performed predictions using a simulator the following analysis is solely based on *results collected from the real system*. In the first part, we compare our three predictors as they were deployed during the time. While the underlying workload is not identical we can still observe how the different predictor variants (v1, v2 and v3) performed with respect to the original user-provided estimates.

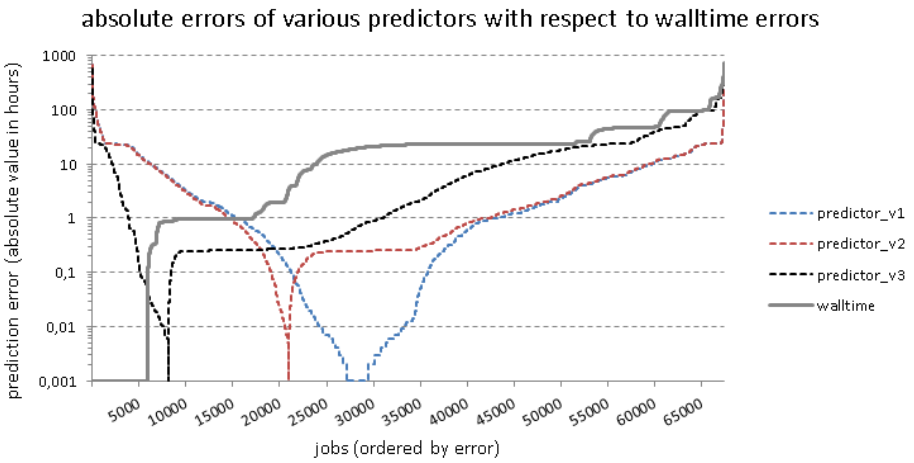

Fig. 4. The ratio and level of under- and overestimation with respect to different soft walltime predictors.

Figure 5 uses the actual data from the system to show the distribution of runtime estimate errors. It consists of 3 boxplots, each covering one time period — October 2021, November 2021 and December 2021–January 2022, respectively. In the first period (October 2021) predictor_v1 was used. It was replaced in November 2021 with the predictor_v2 which was replaced with predictor_v3 that has been used since December 2021. Each boxplot shows the distribution of errors both for the given predictor (v1, v2 or v3) and the original user-provided estimates. Moreover, the errors are divided into three groups according to the actual runtime of the job. The first group represents short jobs (runtime [0, 2] hours), the second group contains all medium jobs (runtime (2, 24] hours) while the third group consists of long-running jobs (runtime $\geq$ 24 hours).

Figure 5 shows how the aforementioned predictors improved the quality of walltime estimates with respect to those provided by users. Also, it shows how the quality of predictions was further improved with those two major modifications that were performed in November (Figure 5 top right) and December (Figure 5 bottom left). As we can see, predictor_v1 used in October 2021 produced a lot of underestimated soft walltimes (Figure 5 top left) and its performance was thus not acceptable. Clearly, the added 15 minute reserve applied in predictor_v2 significantly decreased the underestimation of soft walltimes as can be seen, e.g., from the reduced spread and better lower quartile value (long jobs). The second modification deployed in December 2021 (predictor_v3) further reduced the errors of generated soft walltimes. Version v3 clearly generates soft walltimes that are much more accurate than user-provided estimates (Figure 5 bottom left).

We have also analyzed the accuracy of soft walltimes on a per-user basis. These results are shown in Figure 6. This figure compares the average absolute

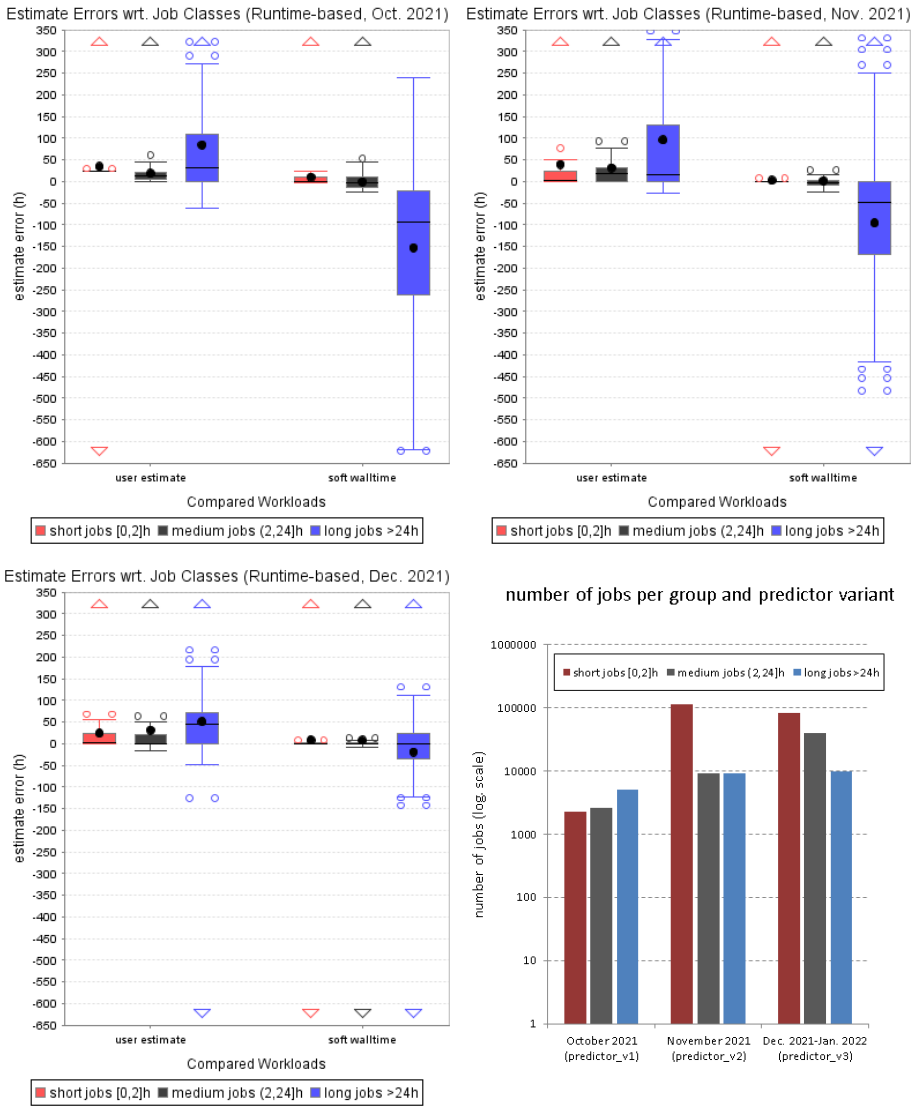

Fig. 5. The impact of two major upgrades of the soft walltime predictor with respect to the initial October version (top left). Both the November and the December upgrades (top right and bottom left) decreased errors significantly. All boxplots have the same scale of y-axis. Bottom right chart shows the number of jobs per job group and epoch.

errors of user-based walltimes and generated soft walltimes. Also, for each user we show the number of jobs they have submitted into the system. In general, soft walltime-based estimates were more accurate for 91% of users (on average). For the remaining 9% users, the average deterioration was rather small. The biggest

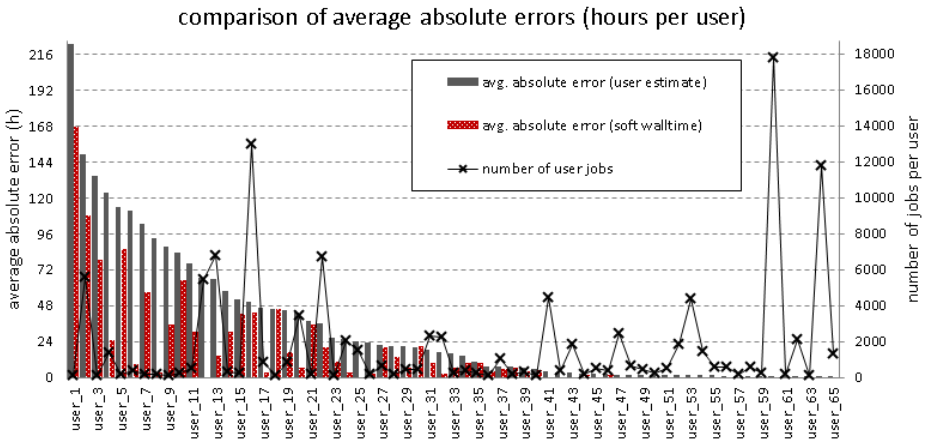

Fig. 6. Comparison of average absolute errors per user.

and smallest deteriorations we have observed were 26 minutes and 36 seconds, respectively. While the drawbacks of soft walltimes were minor, the improvements were very clear. For example, for 77% of users the average improvement in the accuracy was greater than 24 hours, i.e., the average error of user-based estimates was reduced by more than 24 hours thanks to the application of soft walltime predictor.

3.3 Soft Walltime Caveats

During the deployment of soft walltimes in the CERIT-SC system we have also analyzed the *impact of using soft walltimes*. During this process we have come up with a set of "caveats" that one should keep in mind when using soft walltimes. The first one is the danger that originates from underestimated soft walltimes and we have already discussed this caveat in Sections 2.2 and 3.1. The second caveat relates to the way jobs are routed into queues. In our system, user-provided walltime limit is one of the major factor that influences to which queue his or her job will be routed to. Simply put, the system has several queues with different maximum allowed walltime limits. "Short" queues can access larger pools of nodes while "long" have smaller pools of available nodes. These node pools are overlapping and the limits are used to increase the chance that short jobs will not be hugely delayed by long running jobs from long queues. However, jobs are always routed by user-provided walltimes and soft walltimes do not change this at all. It means that although the job may be considered as "short" by the scheduler it cannot be moved into a proper "short" queue and will remain in the "long" queue. Therefore, jobs having long walltimes and small soft walltimes cannot use those larger pools of nodes that are available to "short" queues. As a result, these jobs thus may experience larger wait times compared to jobs from "short" queues.

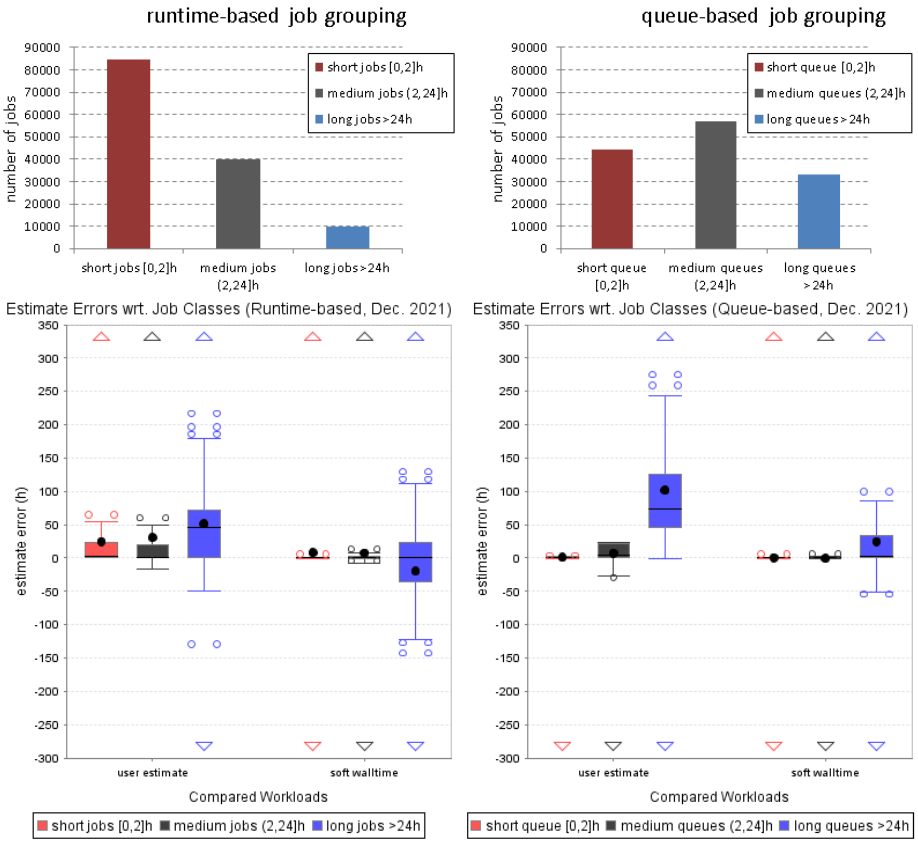

Fig. 7. Job (top) and error (bottom) distribution according to the runtime-based (left) and queue-based (right) job grouping. 3.4× more jobs fall into the "long" category when grouped according to their system queues (right). Both boxplots have the same scale of y-axis.

Figure 7 shows this caveat in practice. On the left side, we present job (top) and error (bottom) distribution when jobs are grouped according to their real runtime. We can see that short and medium jobs are the most common (63% and 30%) while long jobs are scarce (7%). On the other hand, when jobs are grouped according to their queue (right) the job-to-group distribution is much different (33%, 42% and 25%) since many jobs are overestimated by users, thus ending up in "wrong" queues.

This difference then also changes completely the distribution of estimate errors. In the former case (runtime-based grouping) soft walltimes are underestimated for 50% of long jobs while in the latter case (queue-based job grouping) soft walltimes are underestimated for only 25% of jobs from long queues. Similarly, user-provided estimates are much worse when considering long queues (right) instead of long jobs (left). The output from this caveat is that the system

administrator must closely monitor system performance on the per-queue basis as the introduction of soft walltimes will likely cause that resource-restricted queues (e.g., "long" queues in our case) will contain lots of jobs that may be more suitable for other system partitions (e.g., those used by "short" queues). Not only this may limit the system performance but it can also degrade the impact of using soft walltimes.

4 Comparison of System Performance

In the final part of our evaluation we compare how the system performed *prior* and *after* soft walltimes were introduced. We will compare two data sets from CERIT-SC system and analyze their differences. Since we are using real data from the system, this comparison has some inevitable shortcomings. Clearly, we are comparing two different workloads thus we cannot directly compare selected metrics and draw immediate conclusions from such comparison. Instead, we will provide side by side workload comparison and discuss observed trends in the processed workloads. In the future, we plan to extend this work by performing reasonably detailed simulations where we would test various system setups (with or without soft walltimes) using the same workload. This will provide additional and more reliable/comparable results.

Still, we believe that current comparison provides interesting data and we did our best to select two reasonably comparable epochs. First, we made sure to select two epochs where the system setup is identical, i.e., the number of nodes and queues (and their limits) is the same. The first trace where the scheduler uses user-provided estimates is called *CERIT-user-wall*. It comprises 935,724 jobs executed in the system during January–August 2021 period. The second trace where soft walltimes are used is called *CERIT-soft-wall*. It covers 4 months since October 2021 and contains 351,853 jobs. Out of these, 254,663 jobs have recorded soft walltime[3]. In the following comparison, only jobs having soft walltime are considered for comparison in the *CERIT-soft-wall* trace.

4.1 Comparison of Workload Characteristics

Let us first compare job runtime distributions in Figure 8. Clearly, in both workloads short jobs are dominant, followed by medium and long jobs. However, we need to look also on CPU requirements (job parallelism) and several other indicators. To achieve that, we analyze workload similarities in more detail using heatmaps. We are focusing on job sizes and job runtime/walltime distributions, as well as on the way how the total CPU load is distributed with respect to job sizes and durations among these two workloads. We use heatmaps where the "heat" intensity shows the percentage of jobs that fall within a given category of jobs. The y and x axes then characterize job category with respect to their

[3] The difference is caused by the fact that it takes some time before we collect enough data for each user to produce soft walltimes.

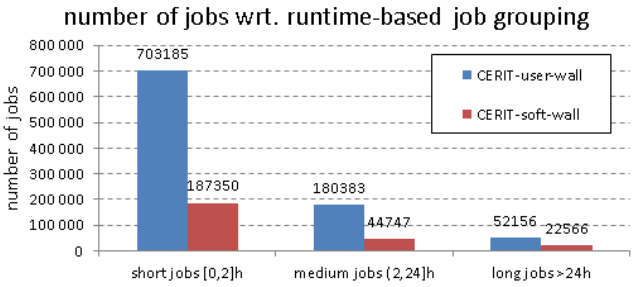

Fig. 8. Job distribution into the three common job runtime categories for CERIT-user-wall and CERIT-soft-wall workload traces.

CPU demands and runtime/walltime, i.e., y-axis denotes the required number of CPUs while x-axis divides jobs into categories according to their runtime or walltime, respectively.

Figure 9 (top) shows that both workloads are quite similar with respect to their CPU requirements and actual job runtime. In both cases, the majority of jobs requires at most 32 CPUs (99.7% and 99.8% of jobs, respectively). Also, the majority of jobs requires at most 1 day to execute (93.9% and 86.7% of jobs, respectively).

The middle row in Figure 9 shows, that both workloads have very coarse grained user-provided estimates. Many jobs simply use the maximum allowed queue runtime limit as their walltime[4]. When compared to the real runtimes (see top row in Figure 9), this heatmap clearly shows how the user-provided estimates are overestimated in both workloads. For example, according to user provided estimates in CERIT-soft-wall workload only 38.6% of jobs is expected to execute within 2 hours, while in reality 68.1% of all jobs run for less than 2 hours.

The last row in Figure 9 compares the use of CPU hours with respect to job classes. Here, the "heat" intensity shows the percentage of total used CPU hours that are consumed by a given category of jobs. It shows that also the overall CPU utilization is quite similar for *CERIT-user-wall* and *CERIT-soft-wall* workloads, i.e., the majority of CPU time is consumed by long-running jobs (*geq* 1 day) in both workloads (82.6% and 88.3% of CPU time, respectively).

Clearly, although the two workloads are not identical, they both have similar patterns, i.e., utilized CPU hours are mostly consumed by long-running jobs while at the same time, most jobs are rather short, requesting less than 40 CPUs. Based on these similarities, we now compare those two workloads by means of job wait time and bounded slowdown.

[4] In our case, those are 2, 4 and 24 hours, 2, 4 and 7 days and 2, 4 or >4 weeks.

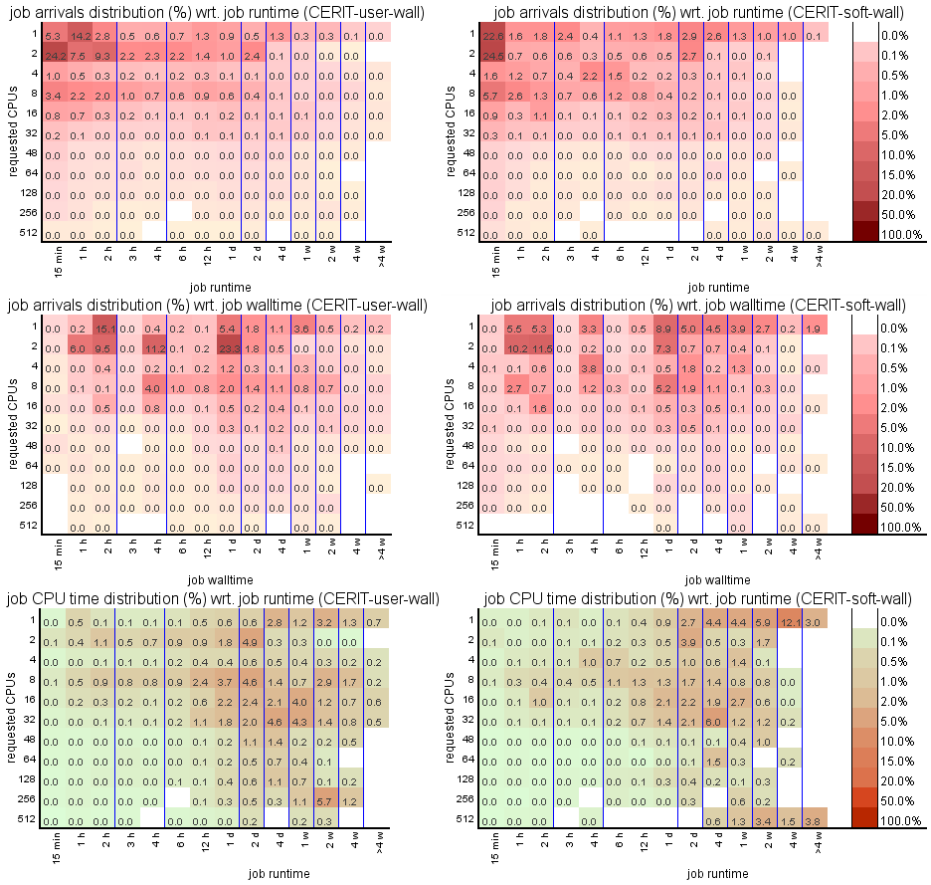

Fig. 9. Comparison of CERIT-user-wall (left) and CERIT-soft-wall (right) workload traces according to their job runtime distribution (top), job walltime distribution (middle) and utilized CPU hours (bottom).

4.2 Comparison of Wait Time and Bounded Slowdown

In this section, we use two optimization criteria to compare CERIT-user-wall and CERIT-soft-wall traces — job wait time and bounded slowdown [4]. Job slowdown is the ratio of the actual response time of the job to the response time if executed without any waiting. By definition, job slowdown is always ≥ 1. As pointed out by Feitelson et al. [4], slowdown reflects users' notion of system responsiveness through measuring if jobs are completed within the time proportional to the job length. In another words, it prefers completion of shorter jobs in a shorter time horizon in comparison with time consuming jobs where a longer waiting is acceptable. Since the whole idea behind soft walltime is to allow the scheduler to "find shorter jobs" and backfill them efficiently we have used these two metrics in our comparison.

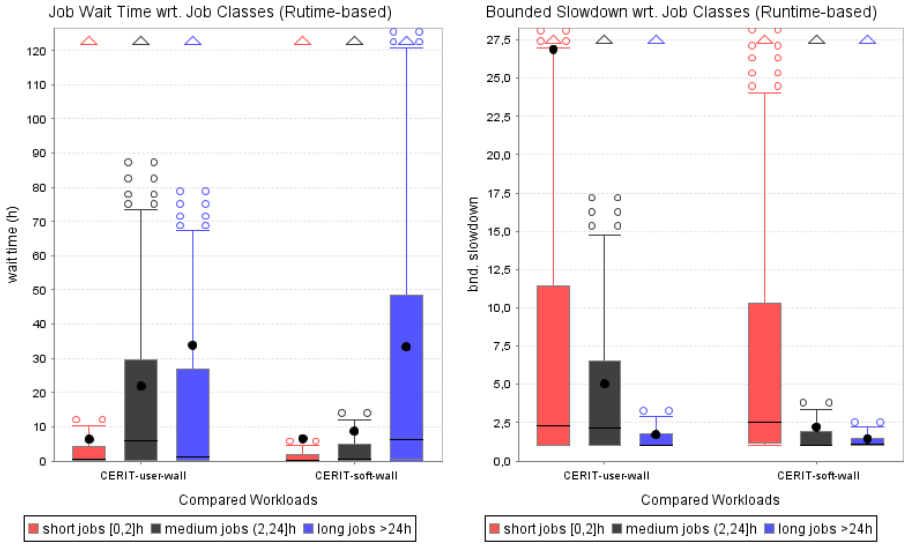

Fig. 10. Wait time (left) and slowdown (right) distributions for CERIT-user-wall and CERIT-soft-wall traces.

In nearly all existing traces (ours included), many jobs have very short runtime. This often represents jobs that ended prematurely right after their start due to some error. These short jobs then skew slowdown distribution (their slowdowns are huge). Therefore, we use so called *bounded slowdown* [3, 4], where the minimal job runtime is guaranteed to be greater than some predefined time constant, in our case 10 minutes[5].

Figure 10 shows the distribution of job wait times (left) and bounded slowdowns (right) for the CERIT-user-wall and CERIT-soft-wall traces. Let us start with the wait time comparison. We can see, that there is a significant difference in the shape of wait time distributions. Soft walltime-driven scheduler (CERIT-soft-wall) seems to distribute high wait times to the long-running jobs, which is natural for backfill-like scheduler. In the original walltime-driven workload (CERIT-user-wall), both short and medium jobs have significantly different wait time distribution, with medium jobs waiting the most. This is an indication that more diverse soft walltimes enable the scheduler to pick shorter jobs more easily and schedule them first. In other words, the wait time distribution in CERIT-soft-wall trace is satisfactory for us, as we expect the system to follow the trend of prioritizing shorter jobs[6]. Again, we must highlight the limitation of this com-

[5] Other values such as 10 seconds [3, 4] or 1 minute [18] are used as well in the literature. In CERIT-SC, 10 minutes is the recommended minimal runtime of regular job. Shorter jobs are not recommended due to excessive overhead related to their (frequent) processing.

[6] This is also coupled with fair-share based job ordering which we use to prioritize less active users over those who utilize the system heavily.

parison, i.e., we are comparing different workloads. As was already discussed in Section 4, instead of directly comparing the observed *values* we are only discussing "shapes" and "trends" here. Direct comparison of wait time values would be quite misleading as the underlying workloads are different.

The distribution of job slowdowns is shown in Figure 10 (right). Here, we see that the long wait times of medium jobs in CERIT-user-wall trace result in rather significant slowdowns as well. For example, the slowdown median for short and medium jobs is very close in CERIT-user-wall trace. On the other hand, CERIT-soft-wall trace has more "natural" slowdown distribution, where largest slowdowns are related to short jobs and decrease as job runtime increases (see the distribution of medium jobs and long jobs). Again, this indicates that the application of soft walltimes helps the scheduler to find suitable jobs for backfilling, which leads to lower wait times and thus reduces their slowdown.

5 Conclusion and Future Work

The aim of this work was to design and implement a soft walltime predictor for the PBS Pro workload management system. The predictor generates job soft walltimes using the knowledge obtained from historical workload data recorded in the database. The developed solution can be obtained from CESNET GitHub repository [5]. Moreover, we have presented our experience when first deploying this predictor in real system. We have analyzed the strengths and weaknesses of our current predictor which we summarize in the following list:

- **Strengths**
  - The predictor is easy to deploy using tools available in PBS Pro.
  - It generates soft walltimes that are much more accurate than user-provided estimates.
  - It has been used in practice for nearly 5 months now.
  - Current results indicate that the system is operating as intended and no negative feedback was observed (since the last upgraded predictor_v3 has been deployed).
- **Weaknesses**
  - Current predictor does not enable to generate different soft walltimes for different job classes of a single user (even if we know that the user has different job classes).
  - As a result, all jobs of a given user are treated as "identical" and their soft walltime is generated using the same scaling factor at the given moment (see *max_walltime_usage* in Algorithm 2).
  - Due to the current limitations in PBS Pro, there is limited possibility to update job soft walltime before job start (soft walltime can be modified only upon job arrival or when it starts but not while it is waiting in a queue). This means that the scheduler may work with rather "old" soft walltimes that are only updated when the job finally starts.

Also, we provide real workload trace that contains generated job soft walltimes[7]. Using this real-life data we have analyzed the accuracy and impact of our approach. We believe this is one of the first publicly available reports on practical application of soft walltimes.

In the future, we want to extend our existing work by detailed simulations that would increase our understanding of how the refined estimates influence the performance of the scheduler. Also, we would like to use such simulations to test new variants of soft walltime predictor that would focus on the current weaknesses (see above).

Acknowledgments. We kindly acknowledge the support and computational resources supplied by the project "e-Infrastruktura CZ" (e-INFRA LM2018140) provided within the program Projects of Large Research, Development and Innovations Infrastructures.

References

1. CERIT Scientific Cloud, February 2022. `http://www.cerit-sc.cz`.
2. 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.
3. D. G. Feitelson. Experimental analysis of the root causes of performance evaluation results: A backfilling case study. *IEEE Transactions on Parallel and Distributed Systems*, 16(2):175–182, 2005.
4. D. G. Feitelson, L. Rudolph, U. Schwiegelshohn, K. C. Sevcik, and P. Wong. Theory and practice in parallel job scheduling. In D. G. Feitelson and L. Rudolph, editors, *Job Scheduling Strategies for Parallel Processing*, volume 1291 of *LNCS*, pages 1–34. Springer Verlag, 1997.
5. Soft walltime predictor implementation, February 2022. `https://github.com/CESNET/softwalltime-predictor/`.
6. JSSPP workloads archive, February 2022. `https://jsspp.org/workload/`.
7. 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.
8. 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.
9. MetaCentrum, February 2022. `http://www.metacentrum.cz/`.
10. 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.

[7] This trace will be published in JSSPP workloads archive [6].

11. S. Seneviratne and S. Witharana. A survey on methodologies for runtime prediction on grid environments. In *7th International Conference on Information and Automation for Sustainability*, pages 1–6. IEEE, 2014.
12. 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.
13. Non-destructive walltime, February 2022. `https://community.openpbs.org/t/pp-482-non-destructive-walltime/587/4`.
14. Soft walltime documentation, February 2022. `https://pbspro.atlassian.net/wiki/spaces/PD/pages/42532871/PP-482+Soft+Walltime`.
15. M. Soysal, M. Bergho, and A. Streit. Analysis of job metadata for enhanced wall time prediction. In N. D. Dalibor Klusáček, Walfredo Cirne, editor, *Job Scheduling Strategies for Parallel Processing*, volume 11332 of *LNCS*, pages 1–14. Springer, 2018.
16. 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.
17. D. Tsafrir, Y. Etsion, and D. G. Feitelson. Backfilling using system-generated predictions rather than user runtime estimates. *IEEE Transactions on Parallel and Distributed Systems*, 18(6):789–803, 2007.
18. S. Vasupongayya and S.-H. Chiang. On job fairness in non-preemptive parallel job scheduling. In S. Q. Zheng, editor, *International Conference on Parallel and Distributed Computing Systems (PDCS 2005)*, pages 100–105. IASTED/ACTA Press, 2005.