Evaluating the Impact of Soft Walltimes on Job Scheduling Performance

Dalibor Klusáček and Václav Chlumský

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

Abstract. For two decades researchers have been analyzing the impact of inaccurate job walltime (runtime) estimates on the performance of job scheduling algorithms, especially in case of backfilling. Several studies analyzed the pros and cons of using accurate vs. inaccurate estimates. Some researchers focused on the ways users of the system can be motivated to provide more accurate runtime estimates. The recent addition of so-called "soft walltime" parameter in the widely used PBS Professional enables a system administrator to actually use some of these techniques to refine user-provided walltime estimates. The obvious question of a system administrator is whether such walltime predictions are useful and "safe" and what will be the impact on the overall system performance. In this work, we use several detailed simulations to analyze the actual impact of using soft walltimes in a job scheduler, discussing the scenarios when such "refined" estimates can be meaningfully used.

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

1 Introduction

In 1995, the seminal EASY backfilling [23] algorithm has been introduced and soon became defacto standard scheduling algorithm in all mainstream resource managers. Since then, many variants of the baseline backfilling have been proposed, e.g., backfilling with multiple job reservations [20], slack-based backfilling supporting priorities and bounded wait times [27] or conservative backfilling where each waiting job gets a reservation [18]. All variants of backfilling that use job reservation(s) have one thing in common. They rely on (inaccurate) job walltime estimates when (A) establishing job reservation(s), i.e., when determining the earliest expected start time for a queued job, and (B) when selecting "filler" jobs that must not collide with these existing reservation(s).

1.1 Walltime Estimates

In practice, these walltime estimates are typically very inaccurate and overestimated [10, 18]. Existing computing systems often use a user's walltime estimate as the upper bound of job's runtime and kill the job when it exceeds its estimated

walltime. 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. As a result, most jobs in the system use only few common estimates and therefore "look similar" to the backfilling algorithm.

Overestimation and limited walltime variability then impede the effectiveness of backfilling on many levels [29]. First, predicted start time(s) for waiting job(s) are very inaccurate, while available "holes" in the schedule appear to be too small for waiting jobs which decreases utilization and throughput. This well-known fact 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 [16, 4]. Although these efforts were significant, they remained mostly "academic" and most systems still face these problems due to several contributing factors.

We believe that there are two main reasons for this unfortunate situation that are surprisingly simple. First, it is just unrealistic to expect that users will thoroughly analyze their walltime requirements, updating them with each new job submission. Therefore, it is up to the resource manager or system administrator to develop and apply some automated technique. However, for many years mainstream resource managers were not prepared to provide *safe ways how to refine walltime estimates* without killing a job when a refined (smaller) estimate is exceeded. Fortunately, this situation has changed in 2017, when the mainstream PBS Professional delivered the concept of so-called *soft walltime* [26].

1.2 Soft Walltimes in PBS Professional

Soft walltimes 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. An important security feature is that soft walltimes cannot be specified or modified by users. Only the manager (system administrator) is allowed to setup them, 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 documentation [26].

1.3 Paper Contribution and Structure

The main contribution of this paper is an experimental analysis that uses simulation to demonstrate the effect of soft walltimes on the job scheduling performance. Our goal was not the development of "yet another runtime prediction technique". Instead, we use four very trivial walltime prediction techniques to

define soft walltimes and then show that even with such trivial techniques *the performance of the system can improve significantly.* The demonstration uses eight publicly available workload traces with different job characteristics and different estimate accuracies, showing the impact on average job wait times and slowdowns. Importantly, we provide detailed wait time analysis using *performance heatmaps* that show performance improvement or deterioration with respect to different job sizes (i.e., job lengths and CPU requirements).

We believe that this new soft walltime functionality available in the open-source PBS Professional together with our promising results — that are however based on very simple prediction techniques — can motivate a new round of practically oriented research on backfilling and runtime prediction techniques.

This paper is organized as follows. Section 2 briefly discusses the related work. Prediction techniques that were used to calculate soft walltimes in this paper are presented in Section 3. Experimental setup and simulation results are presented in Section 4. We conclude the paper in Section 5 and present possible future research directions related to soft walltimes.

2 Related Works

Many works have addressed the problem of inaccurate runtime estimates and the impact they have on the performance of backfilling. For quite some time, it was believed that the inaccuracy has little [11] or even positive effect on the performance of backfilling [4, 10]. However, Tsafrir [29] has demonstrated that accuracy is in fact favorable, similarly to the variability of estimates. The major problem in some of those older works was that they used unrealistic (deterministic/randomized) F-model [10] to synthetically generate (inaccurate) user runtime estimates. However, as pointed out by Tsafrir [30, 29], estimates generated by F-model provide too much information to the scheduler and do not correspond to the typical coarse-grained nature of user-provided estimates.

At the same time, researchers have considered ways how to obtain better job runtime estimates. One way is to try to motivate users by incentives. Authors of [16] have shown that even when users are motivated to improve the accuracy of their estimates — with the assurance that their jobs will not be killed if the improved estimates are too short — the accuracy of their new estimates was, on average, only slightly better than their original estimates [16].

In such situation, it is not surprising that several automated techniques have been proposed to establish more precise estimates. These techniques can be divided into several categories according to the applied estimation technique. The technique proposed in [5] uses repeated executions of the job to establish the estimate while other solutions work on the basis of compile-time analysis [21, 2] or using a historical information together with the statistical analysis [25] of previously executed jobs. Such approach has been applied in [24] where the authors use a template-based approach to categorize and then predict job execution times. This approach is based on the observation that similar applications are more likely to have similar runtimes than applications that have nothing

in common. The similarity is based on several parameters such as the type of the job, the owner of the job, the requested number of CPUs, etc. According to this information the jobs are divided into categories and the runtime estimate is computed using historical data [24]. Such categorization according to similarity has been used by many researchers [28, 15].

Often, predicted runtimes were used to address a closely related issue of estimating queue wait times [25, 19, 15]. For example, the mean queue delay predictions are derived by simulating the future behavior of the scheduler [25], or a uniform-log distribution is used to model the remaining lifetimes of jobs currently executing to predict when required machine(s) will become available and thus when the job waiting at the head of the queue will start [6]. Also, fully automatic methods for predicting bounds (with specific levels of certainty) on the amount of queue delay each individual job will experience have been developed [19]. Although these methods often use some form of job runtime prediction, they are out of the scope of this paper.

We kindly refer, e.g., to this survey [22] for more details concerning various runtime prediction techniques.

3 Runtime Prediction Techniques

In order to evaluate the suitability of soft walltimes we have used four different ways of computing such "refined" runtime estimate that we describe in this section. We did not use any of the aforementioned advanced techniques but used rather straightforward and easy-to-compute predictions.

Each technique 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]. Our first and most trivial solution uses the actual runtime of the last completed user's job as the new soft walltime for the newly arrived user's job (see Formula 1). The second solution depicted by Formula 2 keeps track of all runtimes of all completed user's jobs and uses the average runtime as the new soft walltime. Although these techniques are truly easy to implement, they are not very accurate. For example, if a given user is simultaneously using two different types of calculations (represented by short and long jobs), then the first technique will often either overestimate or underestimate the runtime significantly while the latter (average) will produce estimates that lie between those actual runtimes, i.e., such estimates will be always either over or underestimated.

$$soft_walltime(job_i) = runtime(job_{i-1}) \tag{1}$$

$$soft_walltime(job_i) = \frac{1}{i-1} \sum_{k=1}^{i-1} runtime(job_k) \tag{2}$$

[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.

In order to address this issue, the third and fourth solutions are somehow more complicated and do not use the actual job runtime directly. Instead, they measure the fraction of job's actual runtime and user's estimate (see $walltime_{usage}$ in Formula 3), i.e., they measure to what extent the estimated walltime was actually used. Since the user's estimate is the upper bound of job runtime[2], $walltime_{usage}$ 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. It is fair to mention, that similar approach has been used in [28]. Once the $walltime_{usage}$ is computed, it is used by our third and fourth prediction techniques to generate a soft walltime.

The third prediction technique first computes the average of $walltime_{usage}$ values, i.e., it computes the fractions of used walltimes of all previously completed jobs, and then multiplies the walltime estimate of a new job by the average of these numbers (see Formula 4). The result is then used as the new soft walltime. The fourth technique — instead of using the average — keeps track of five most recent completed jobs. For each such job it computes the $walltime_{usage}$ and then chooses the maximum and multiplies the job's walltime estimate by this maximum (see Formula 5). It represents a conservative strategy, where the new soft walltime is calculated using the known relative accuracy of user's recent estimates. By choosing the maximum $walltime_{usage}$ (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 soft 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 — which is not the case when the average-based method is used instead.

$$walltime_{usage}(job_i) = \frac{runtime(job_i)}{walltime(job_i)} \tag{3}$$

$$soft_walltime(job_i) = walltime(job_i) \cdot \frac{1}{i-1} \sum_{k=1}^{i-1} walltime_{usage}(job_k) \tag{4}$$

$$soft_walltime(job_i) = walltime(job_i) \cdot \max_{i-5 \leq k \leq i-1} walltime_{usage}(job_k) \tag{5}$$

During our initial experiments, we have soon realized that out of these four techniques, the worst results are typically obtained by the average-based techniques (second and third technique). Therefore, in the remaining part of this paper we only use the first (runtime of last completed job) and the fourth prediction technique, depicting them as *last runtime* and *min. diff.*, respectively.

4 Experimental Evaluation

This section describes the results of our evaluation, where we use aforementioned prediction techniques to generate soft walltime limits. Before we proceed

[2] The system is configured to kill a job if it exceeds user's walltime estimate.

to the results of our simulations, we describe the workload traces used in our experiments and the simulation methodology.

4.1 Workload Log Characteristics

In this work, we use eight different workloads coming from different systems with different parameters. Four workloads come from the Czech National distributed computing infrastructure. This infrastructure is managed by two major resource providers — CERIT-SC and MetaCentrum — each having its own job scheduler. We use three workload traces from CERIT-SC [3] system and one trace from MetaCentrum [17]. MetaCentrum_2013 trace includes 150K jobs, while CERIT-SC_2013, CERIT-SC_2015 and CERIT-SC_2017 contain 257K, 102K and 252K jobs, respectively. Remaining four workloads come from the Parallel Workloads Archive (PWA) [8]. We have used HPC2N, KTH SP2, CTC SP2 and SDSC SP2 traces that contain 202K, 28K, 77K and 59K jobs, respectively.

These workloads were selected because they represent very different systems. For example, both MetaCentrum and CERIT-SC are rather heterogeneous environments, providing access to several different clusters simultaneously. MetaCentrum_2013 workload trace comes from 14 clusters, CERIT-SC_2013 comes from 4 clusters while CERIT-SC_2015 and CERIT-SC_2017 come from 6 and 7 clusters, respectively. On the other hand, HPC2N, KTH SP2, CTC SP2 and SDSC SP2 traces each represent workloads coming from a single homogeneous cluster. Most importantly, these eight workloads exhibit very different *levels of accuracy* and different *variability* of users' runtime estimates. This is a very important factor which allows us to analyze how soft walltimes behave subject to either very imprecise or reasonably accurate estimates.

Figure 1 shows the cumulative distribution functions (CDF) of user estimates and actual runtimes for all eight data sets. Clearly, MetaCentrum_2013 and CERIT-SC_2013 traces have very poor estimates, where most users chose the default 24 hour estimate. The situation is slightly better in CERIT-SC_2015 and CERIT-SC_2017 workloads because by that time the default 24 hour estimate has been disabled by system administrators and users were forced to specify estimates upon job submissions. Still, the shape of the CDF resembles a staircase [29], which means that users preferred several common estimates, e.g., 2 hours, 4 hours, 24 hours, 2 days or 1 week.

On the other hand, all workloads from PWA show better precision and variability of walltime estimates. It is also worth mentioning, that with the exception of HPC2N, these workloads do not contain jobs requesting walltime greater than 24 hours, which is another major difference with respect to MetaCentrum and CERIT-SC traces.

4.2 Simulation Methodology

All experiments have been performed using Alea jobs scheduling simulator [13], with EASY backfilling as the scheduling algorithm [23]. The simulation code can

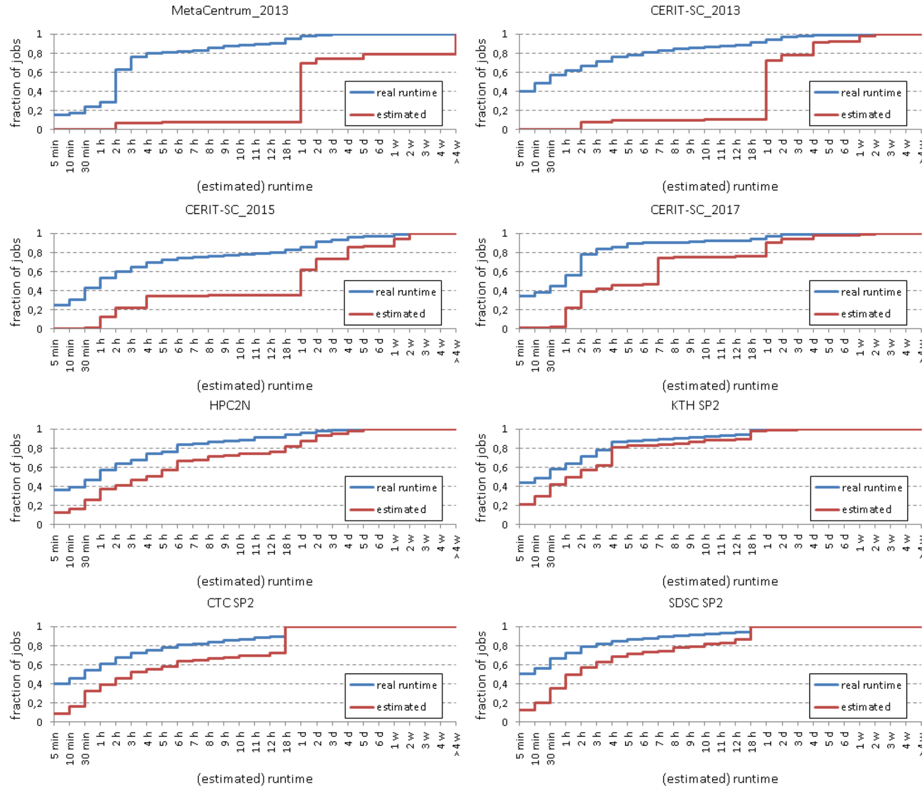

Fig. 1. Cumulative distribution functions (CDF) of actual and estimated job runtimes for all eight workloads.

be found at GitHub [1], while all traces can be obtained either from the Parallel Workloads Archive [8] or from the JSSPP's workloads archive [12].

Perfect Estimates, User-Provided Estimates and Soft Walltimes The experiments have been conducted as follows. First, we have simulated workload execution using perfect estimates, i.e., actual job runtimes were used by EASY backfilling to determine job reservations and backfilling opportunities. This setup represented our *baseline ideal solution*, where EASY is performing "correct decisions" based on accurate information. In the second step, we have used the user-provided estimates (without using soft walltimes). This setup represented the *standard solution*, i.e., the common situation which is normal in systems where no additional walltime refinement is used. Finally, we have run the same experiment while using soft walltimes in the EASY backfilling. As discussed in Section 3, only the *last runtime* and the *min. diff.* soft walltime prediction techniques have been used because average-based prediction techniques did not work very well.

Workload Replay and Dynamic Workload Adaptation It is a common practice to perform simulations by using job workloads in a static way. In such scenario, a given workload is "replayed" in the simulator using original job submission timestamps. Although such experiments allow for easy comparison of different simulation setups they are less likely to realistically "mimic" users interactions and behavior. As explained in [31], job submission times in a real system depend on how users react to the performance of previous jobs. Moreover, usually there are some logical structures of dependencies between jobs. It is therefore not very reasonable to use a workload "as is" with fixed (original) job submission timestamps, as the subsequent simulation may produce unrealistic scenarios with either too low or too high system loads, skewing the final results significantly.

Instead, dependency information and user behavior can be extracted from a workload trace, in terms of job batches, user sessions and think times between the completion of one batch and the submission of a subsequent batch. Then, each user's workload is divided into a sequence of dependent batches. During the simulation, these dependencies are preserved, and a new user's batch is submitted only when all its dependencies are satisfied (previous "parent" batches are completed). This creates the desired feedback effect, where job submission times are not dictated by the workload but are the result of the (simulated) scheduler-to-user interaction as users dynamically react to the actual performance of the system. At the same time, major characteristics of the workload including job properties or per-user job ordering are still preserved. More details can be found at [31, 13].

In order to get reasonable results we use a compromise simulation scenario, combining both static and dynamic workloads. We use the dynamic approach of Zakay and Feitelson [31] with our two most recent workloads (CERIT-SC_2017 and CERIT-SC_2015). These workloads are "fresh", representing realistically the system that we are trying to optimize in practice. For the six remaining workloads we use the standard simulation practice, i.e., we use them statically.

Result Analysis In case of both static and dynamic workloads we analyze the performance using two different approaches. First, we measure the overall impact of inaccurate walltimes and soft walltimes using the common average wait time [7] and the average slowdown [9] criteria. These results are discussed in Section 4.3. Next, we use detailed heatmaps [14] to better understand the impact of estimates and soft walltimes on jobs with respect to their CPU and runtime requirements. In a heatmap, a given metric is computed separately for each square of that heatmap. A square (or a bucket) on a heatmap is defined by its x and y coordinates and represents all jobs that fall into this category based on their CPU (y-axis) and runtime requirements (x-axis). Heatmaps are very useful visual aids allowing for quick and rather detailed result comparisons. These detailed results based on heatmaps are presented in Section 4.4. Furthermore, when the dynamic workload adaptation is used (CERIT-SC_2015/2017 workloads), we provide an additional analysis that measures the impact that

soft walltimes have on individual jobs and users[3]. These additional results are also presented in Section 4.4. Finally, we conclude our experiments with the discussion in Section 4.5.

4.3 Overall Results

We start our evaluation by focusing on the overall impact that inaccurate estimates and soft walltimes have on the average wait time and slowdown. As the *baseline experiment* we always use the results obtained when simulating EASY backfilling using perfect estimates (i.e., estimate = runtime). Next we measure the improvement or deterioration of average wait time and slowdown. The improvement or deterioration is expressed as *percentage* and is computed using Formula 6, where $metric_{baseline}$ denotes the avg. wait time/slowdown of the baseline solution (perfect estimates) and $metric_x$ is the value of avg. wait time/slowdown of the solution where perfect estimates were replaced either with the user-provided estimates (*estimated*) or soft walltimes (either *last runtime* or *min. diff.*).

$$percentage = \frac{metric_{baseline} - metric_x}{metric_{baseline}/100} \tag{6}$$

If a given metric is improved (i.e., avg. wait time/slowdown is decreased) then the resulting percentage is positive while a deterioration of a metric results in a negative percentage. It is worth noticing that positive percentage cannot exceed 100%, while negative percentage is not upper bounded[4]. The results of this experiment are shown in Figure 2 (avg. wait time) and Figure 3 (avg. slowdown), respectively. As discussed, for each workload trace we show the improvement/deterioration obtained with respect to the baseline solution[5].

Let us start with the average wait time. With the exception of HPC2N workload, the average wait times deteriorated when original user-provided estimates (*estimated*) were used compared to a solution computed using perfect estimates. Similarly, also the average slowdown of *estimated* deteriorated in all case, compared to the baseline scenario. As discussed, e.g., in Tsafrir's papers [29, 30], this

[3] Unlike in the static scenario, user-oriented analysis makes a great sense when the workload is dynamically adapted.

[4] For example, if the result is 25% it means that, e.g., the original wait time was decreased by 25%. On the other hand, if the result is -300%, it means that the original wait time was increased by 300%, i.e., four times.

[5] The actual average wait times of the baseline solution were as follows: CERIT-SC_2015 (6.5 hours), CERIT-SC_2017 (4.1 hours), MetaCentrum_2013 (3.0 hours), CERIT-SC_2013 (6.0 hours), HPC2N (4.2 hours), KTH SP2 (1.8 hours), CTC SP2 (3.8 hours) and SDSC SP2 (4.9 hours). The average slowdowns of the baseline solution were following: CERIT-SC_2015 (249.9), CERIT-SC_2017 (127.9), MetaCentrum_2013 (115.5), CERIT-SC_2013 (620.7), HPC2N (143.2), KTH SP2 (105.8), CTC SP2 (49.9) and SDSC SP2 (72.8).

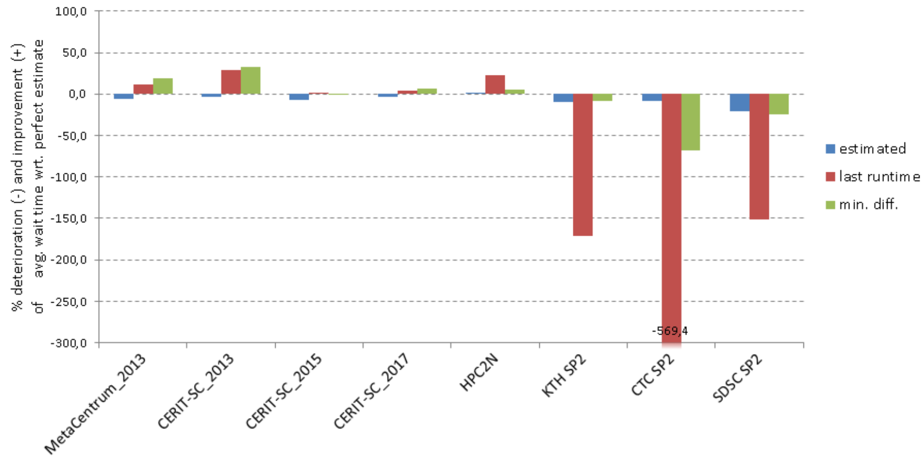

Fig. 2. Avg. wait time improvement (+) and deterioration (-) for all eight workloads.

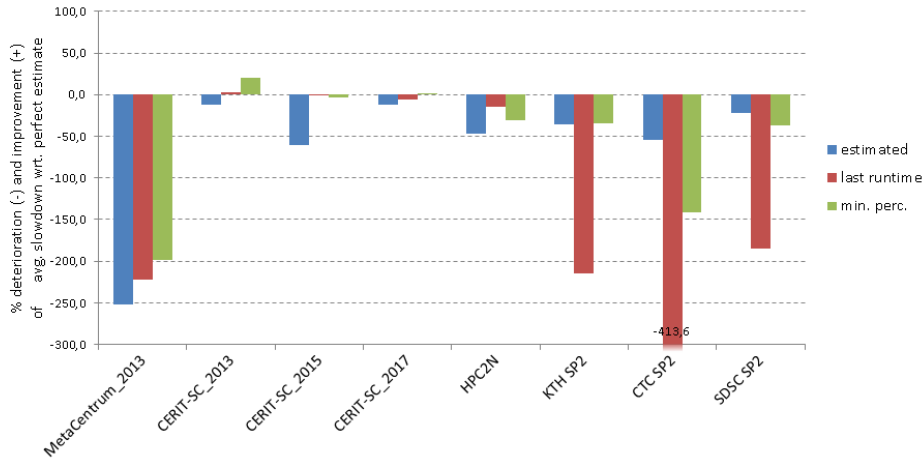

Fig. 3. Avg. slowdown improvement (+) and deterioration (-) for all eight workloads.

is not surprising since less accurate and less variable estimates may worsen the performance.

More interesting results are achieved by those two soft walltime-based prediction techniques (*last runtime* or *min. diff.*). In this case we are not only interested whether or not they improve/deteriorate the performance with respect to the baseline solution but more importantly we focus how good or bad are they with respect to (A) *estimated* solution and (B) each other. When the soft walltimes are constructed using *last runtime*, the results are better with respect to *estimated* in five workloads. In the remaining three workloads, using *last runtime* as the soft walltime prediction technique actually deteriorates the

performance, both from the point of view of the avg. wait time and slowdown. The *min. diff.* technique works better. It outperforms the *estimated* scenario in six workloads (both wait time and slowdown). Also, it outperforms the results of *last runtime* technique in six of eight workloads (both wait time and slowdown). Therefore, from now on we will only consider the *min. diff.* soft walltime prediction technique in the following experiments.

What these results show is that it certainly makes sense to consider soft walltimes in some cases. As can be seen, soft walltimes work best in the first five or six workloads (six, if only *min. diff.* is considered). We now try to answer the question why is it so. The main difference between the first five workloads and the remaining three (KTH SP2, CTC SP2 and SDSC SP2) is that all of them contain very long jobs that run/require more than 24 hours. Also, all CERIT-SC/MetaCentrum-based workloads have very poor user-provided runtime estimates, compared to the remaining four traces. Clearly, the combination of long jobs and poor original estimates increases the chance that even trivial prediction technique such as *min. diff.* will produce a reasonable "mixture" of varying estimates that are very useful when "filling the holes" in the schedule[6].

Although our initial experiments shed some light on the problem, we now proceed to a more detailed analysis. So far, we have only used the average wait time/slowdown criteria which can be easily skewed by the long-tail effects of the underlying job wait time and slowdown distributions. Therefore, in the following section we use performance heatmaps [14] to better demonstrate the improving effect of soft walltimes.

4.4 Detailed Performance Analysis using Heatmaps

This analysis uses two different types of heatmaps. The first type is used to show the distribution of jobs with respect to their actual runtime and CPU requirements, i.e., it shows which job classes (sizes) are the most common ones and which on the other hand are quite rare. The color scale of such heatmap represents the number of jobs belonging to a given "bucket".

The second type of heatmap is used to show the difference among average wait times of two different simulation setups. We compare the average wait time on a per "job bucket" basis. The color scale of such heatmap then shows the differences between the avg. wait times of the *baseline* solution and either *estimated* or *min. diff.* scenario. The result is either positive, negative or zero. A positive value (shades of red) represents an improvement with respect to the baseline solution (wait time of the baseline solution was higher), while a negative value represents a deterioration with respect to the baseline solution (shades of blue). A zero value represents no difference of average wait times (white color).

[6] With only few estimates used throughout the whole workload, backfilling has significantly decreased opportunity to fill these holes, because "most jobs look the same" and thus do not fit within available holes.

Dynamic Workloads We start the discussion with our two dynamic workloads — CERIT-SC_2015/2017. The heatmaps for these workloads are presented in Figures 4 and 5. Each such figure shows the job distribution heatmap (top), avg. job wait time difference heatmaps for *baseline* vs. *estimated* setup (middle) and *baseline* vs. *min. diff.*-based soft walltimes (bottom).

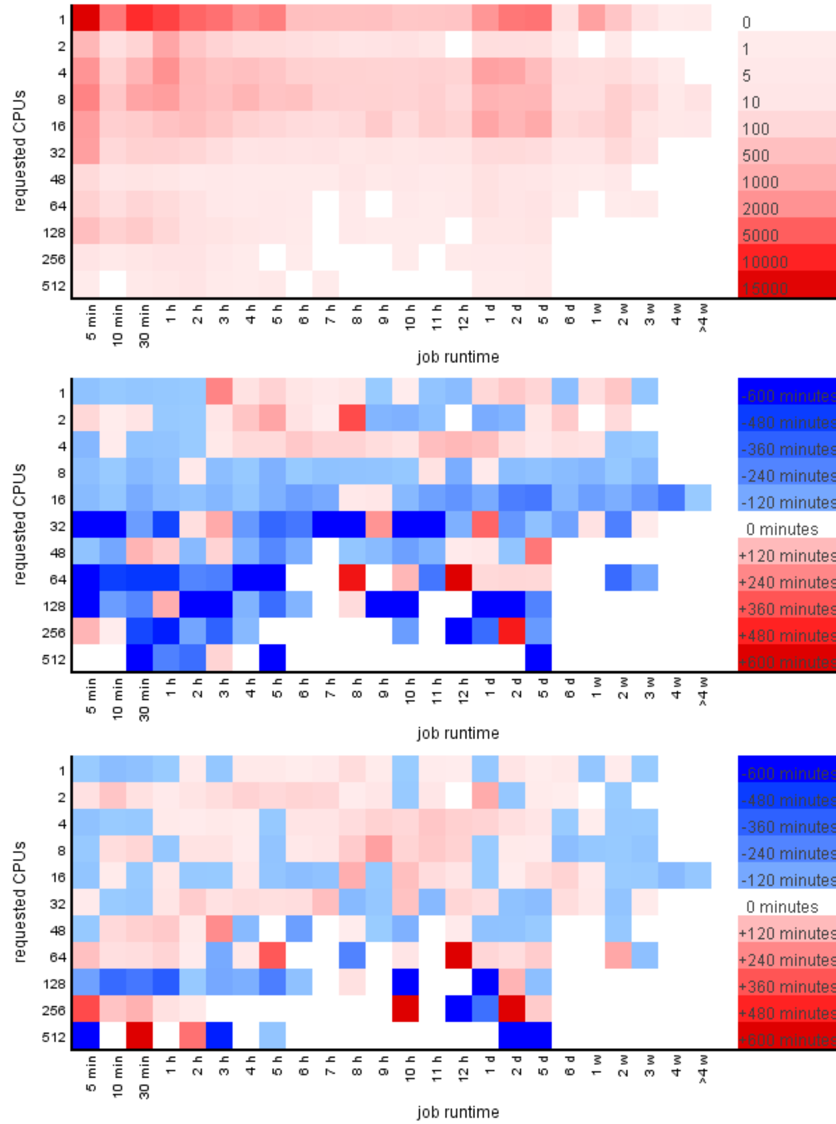

Fig. 4. CERIT-SC_2015. Job distribution heatmap (top), avg. job wait time difference heatmaps for *baseline* vs. *estimated* (middle) and *baseline* vs. *min. diff.* (bottom).

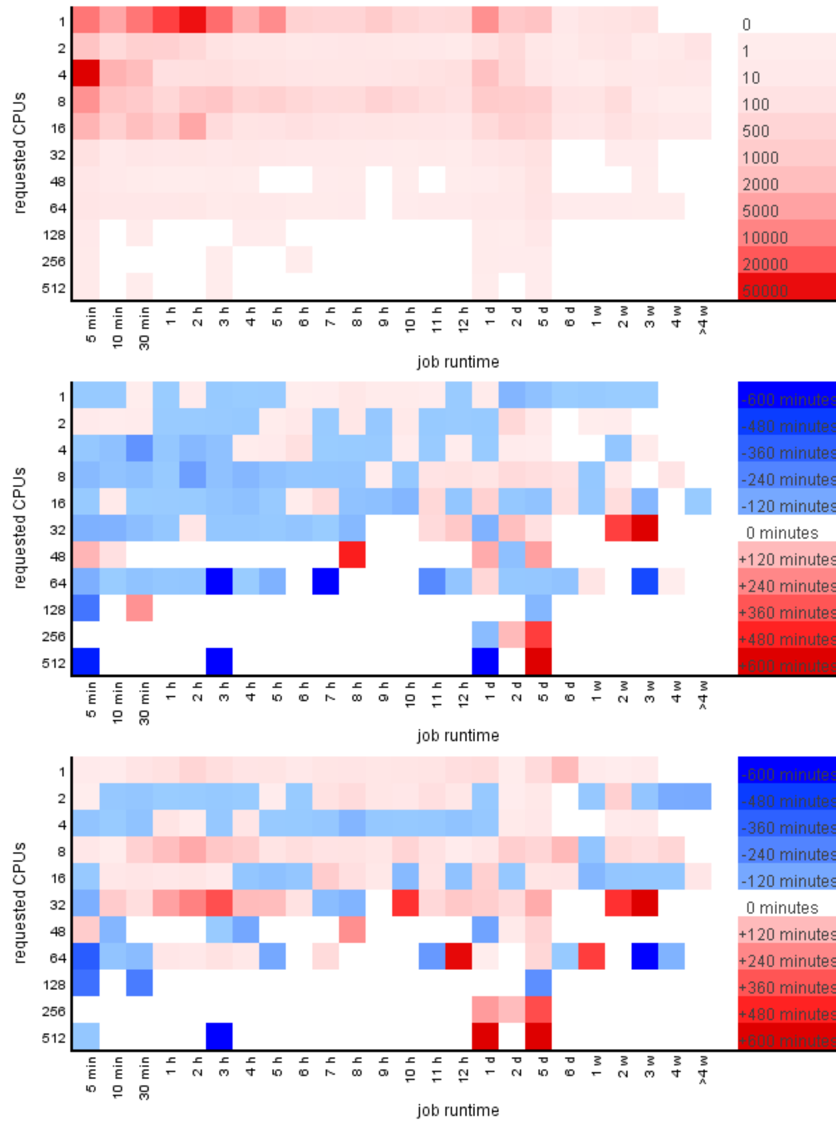

Fig. 5. CERIT-SC_2017. Job distribution heatmap (top), avg. job wait time difference heatmaps for *baseline* vs. *estimated* (middle) and *baseline* vs. *min. diff.* (bottom).

As we can observe, the promising results related to soft walltimes that were observed in Section 4.3 are confirmed by the (bottom) heatmaps in Figures 4–5. Here, the bottom heatmaps — showing the avg. wait time difference between baseline and soft walltime-based solution — are always "more red and less blue" than middle heatmaps, in which the difference between the baseline solution and the solution based solely on user-provided runtime estimates is shown. This

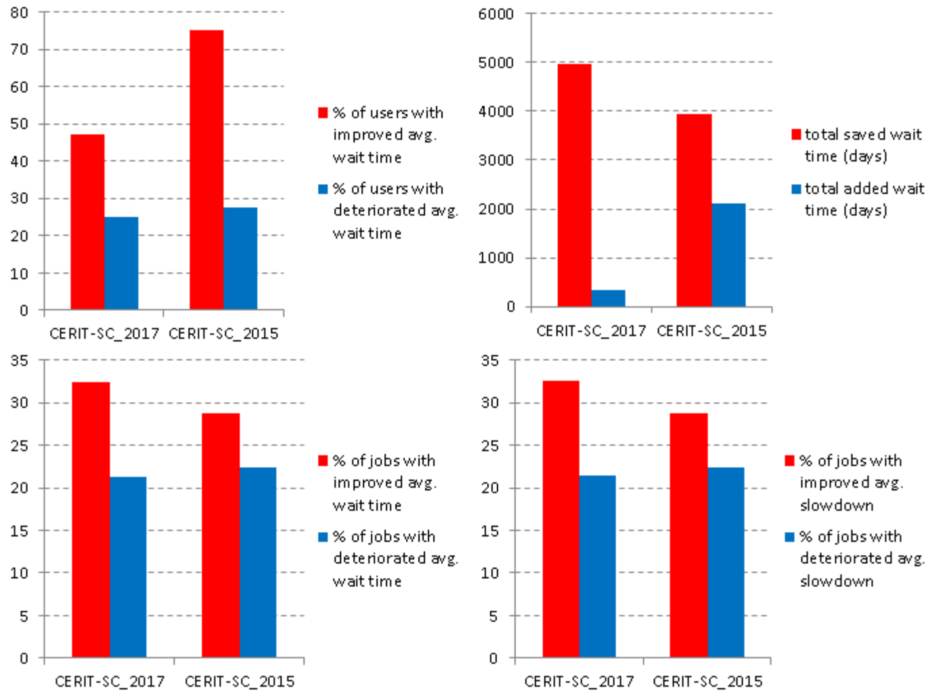

Fig. 6. Percentage of users with improved/deteriorated avg. wait times (top left), the total saved/added wait time of jobs (top right), % of jobs with improved/deteriorated avg. slowdown (bottom right) and % of jobs with improved/deteriorated avg. wait time (bottom left).

demonstrates that the use of *min. diff.*-based soft walltimes helps to improve the performance of EASY backfilling, reducing the average job wait time significantly for most job "sizes".

Accompanying results related to the two dynamic workloads are provided in Figure 6. Here we use several additional measurements (based on wait time and slowdown) to further illustrate the overall positive effect that soft walltimes create compared to the original user-provided estimates. Starting in the upper left corner and moving clockwise, the figure shows the percentage of users with improved/deteriorated average wait times, the total saved/added wait time of jobs in the system, the percentage of jobs with improved/deteriorated average slowdowns and finally the percentage of jobs with improved/deteriorated average wait times.

These results confirm, that soft walltimes improve the overall performance of the system. As the metrics indicate, many users now wait shorter and many jobs now have better wait times and slowdowns. As a result, the system minimizes overall waiting, as shown by the "total saved/added wait time" chart. Sure, every

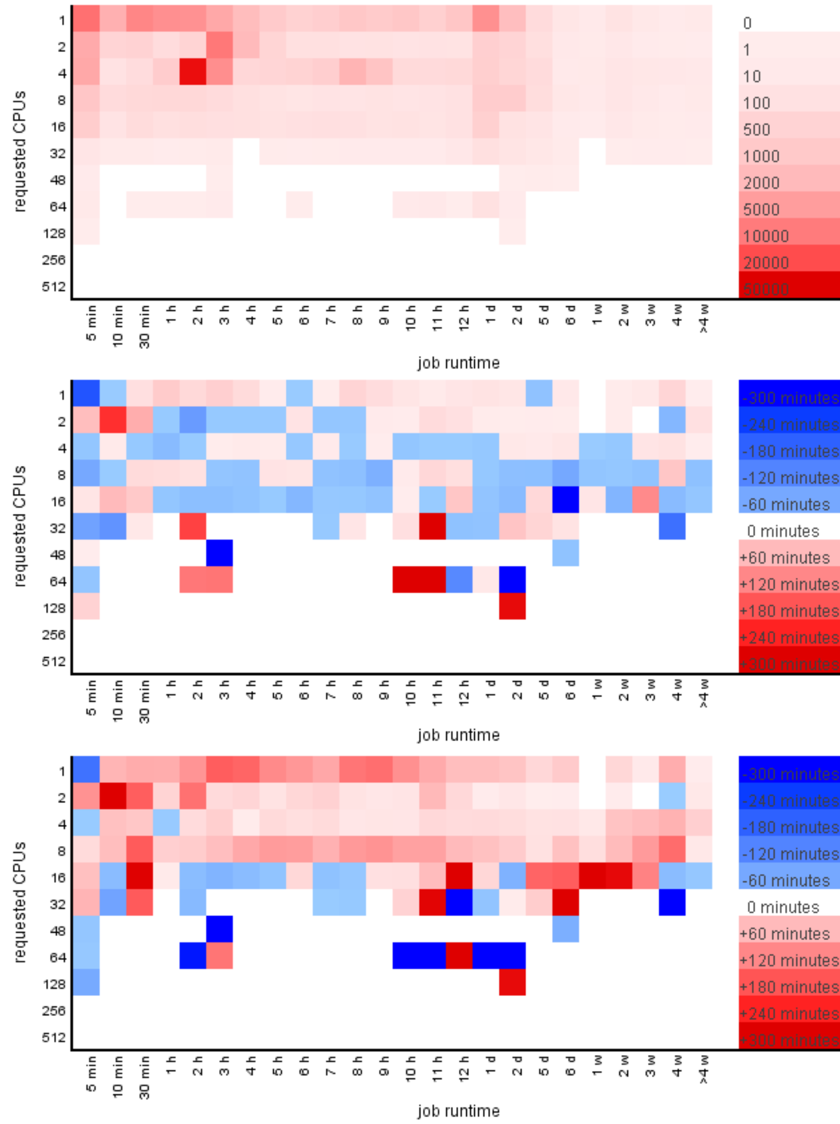

Fig. 7. MetaCentrum_2013. Job distribution heatmap (top), avg. job wait time difference heatmaps for *baseline* vs. *estimated* (middle) and *baseline* vs. *min. diff.* (bottom).

improvement comes with a price and we can see some performance deterioration for a fraction of jobs or users. However, the overall balance is always positive.

Static Workloads Heatmap-based results for the remaining six static workloads are shown in Figures 7–12. Again, the promising results of soft walltimes

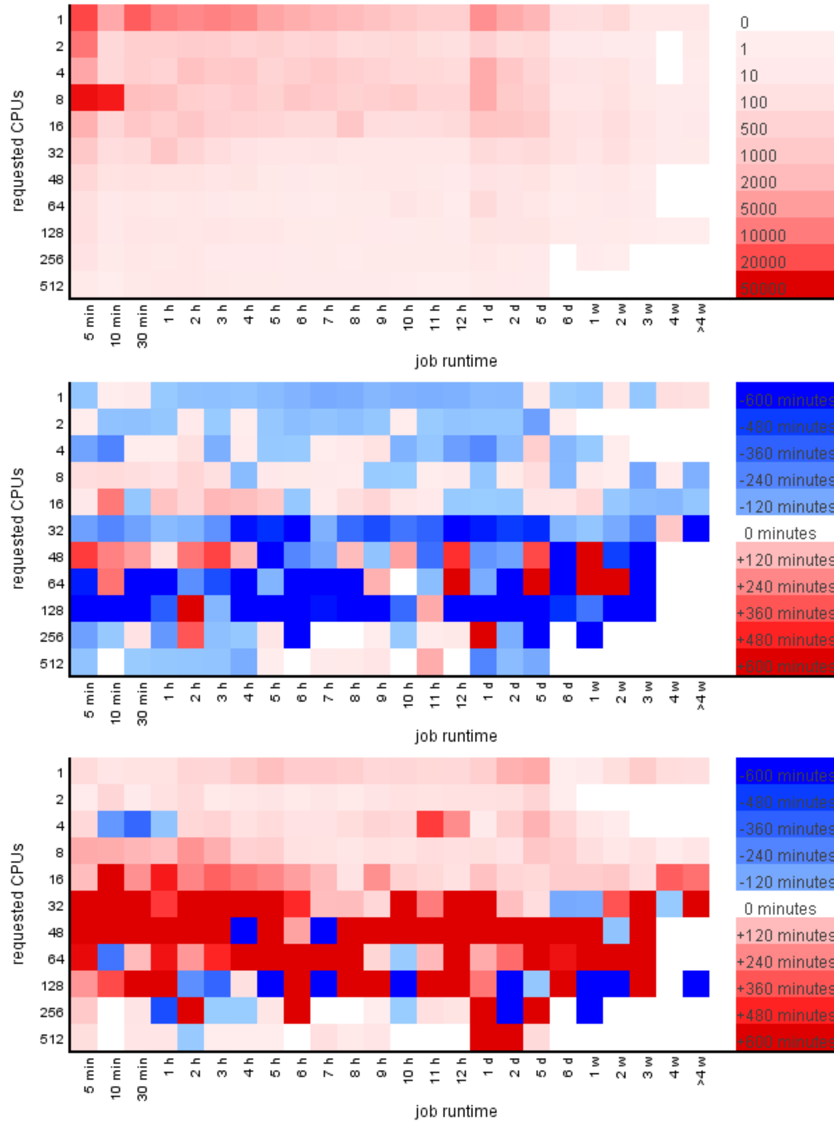

Fig. 8. CERIT-SC_2013. Job distribution heatmap (top), avg. job wait time difference heatmaps for *baseline* vs. *estimated* (middle) and *baseline* vs. *min. diff.* (bottom).

that were observed for MetaCentrum_2013 and CERIT-SC_2013 workloads in Section 4.3 are confirmed by the heatmaps in Figures 7 and 8. The improvement obtained by soft walltimes (bottom heatmaps) is significant compared to the middle heatmaps which show the difference between baseline (perfect) solution and solution based on the inaccurate user-provided runtime estimates.

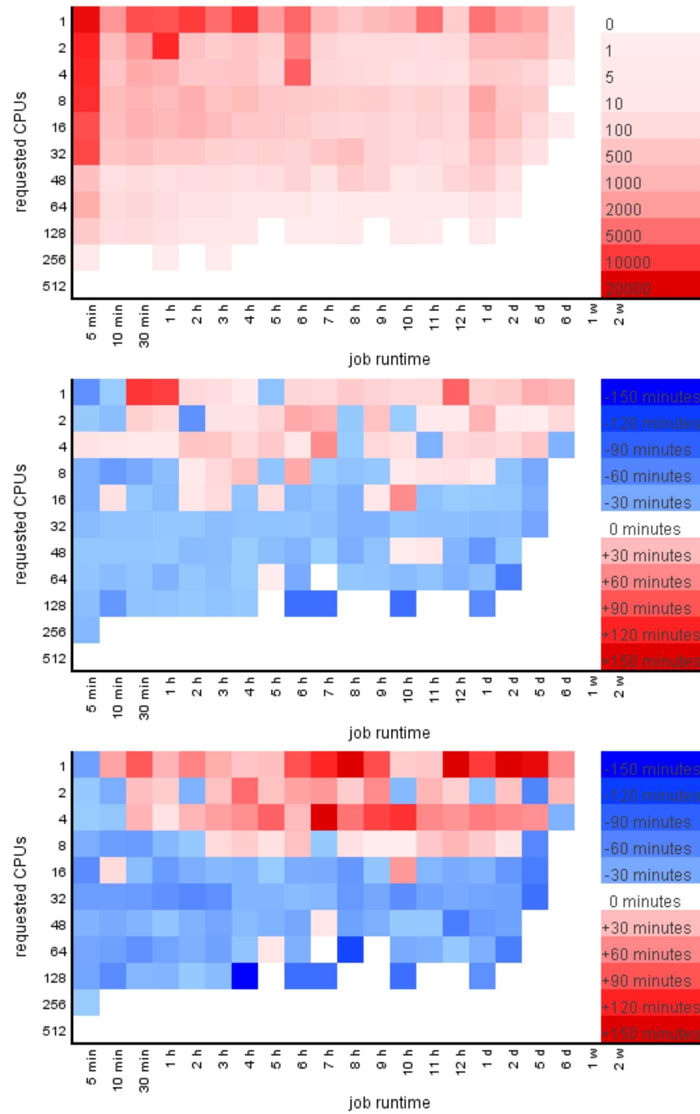

Fig. 9. HPC2N. Job distribution heatmap (top), avg. job wait time difference heatmaps for *baseline* vs. *estimated* (middle) and *baseline* vs. *min. diff.* (bottom).

A different situation is visible in case of HPC2N (Figure 9) and KTH SP2 (Figure 10) workloads. Here we see that soft walltime-based solutions (bottom heatmap) slightly improve average wait times for jobs requiring small amounts of CPUs. The effect is more visible for HPC2N. On the other hand, wait times of longer and more CPU demanding jobs are slightly increased (compared to the middle heatmap). As the top heatmaps reveal, both HPC2N and KTH SP2 have

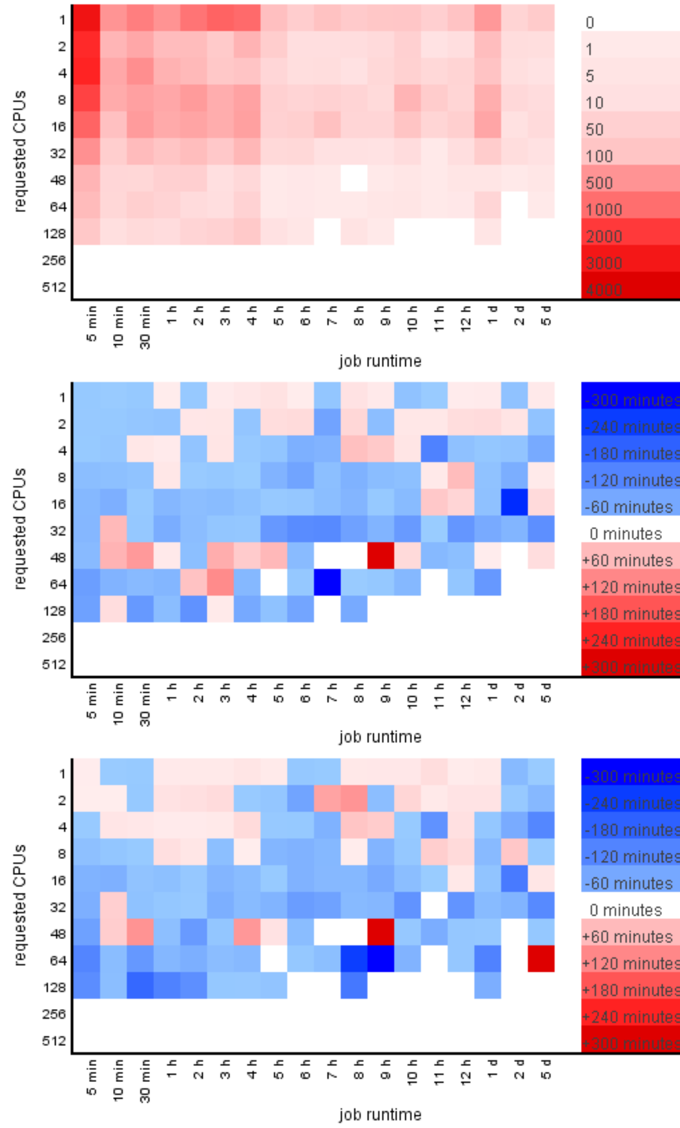

Fig. 10. KTH SP2. Job distribution heatmap (top), avg. job wait time difference heatmaps for *baseline* vs. *estimated* (middle) and *baseline* vs. *min. diff.* (bottom).

large job concentrations in the upper half of the heatmap, meaning that many jobs in the workload require only up to 32 CPUs. These are the same parts where the improvement can be observed. Together, it help us to understand the average wait time results seen in the Figure 2, where the setup involving soft walltimes (*min. diff.*) slightly outperformed the setup using user-provided estimates.

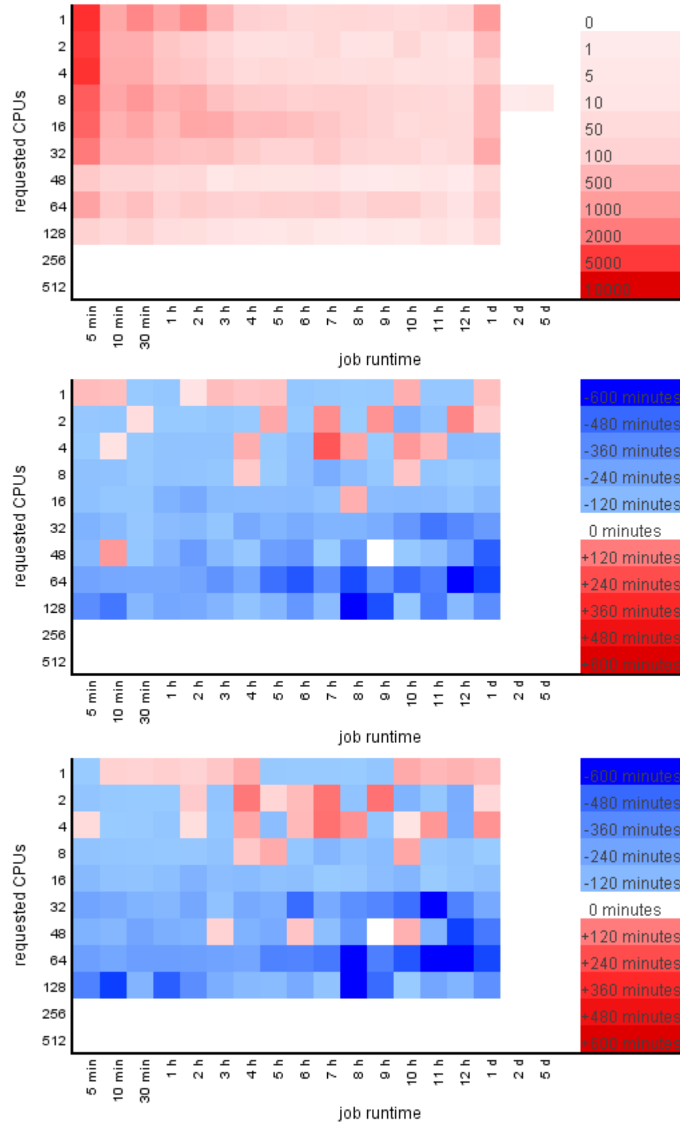

Fig. 11. SDSC SP2. Job distribution heatmap (top), avg. job wait time difference heatmaps for *baseline* vs. *estimated* (middle) and *baseline* vs. *min. diff.* (bottom).

Finally, Figures 11 and 12 (SDSC SP2 and CTC SP2 workloads) show the two situations in which soft walltimes based on *min. diff.* predictions do not work at all. In case of SDSC SP2, setup involving soft walltimes behaves similarly to the one relying on user-provided estimates, which corresponds to the observations in Figure 2. In case of CTC SP2, the use of soft walltimes deteriorates the

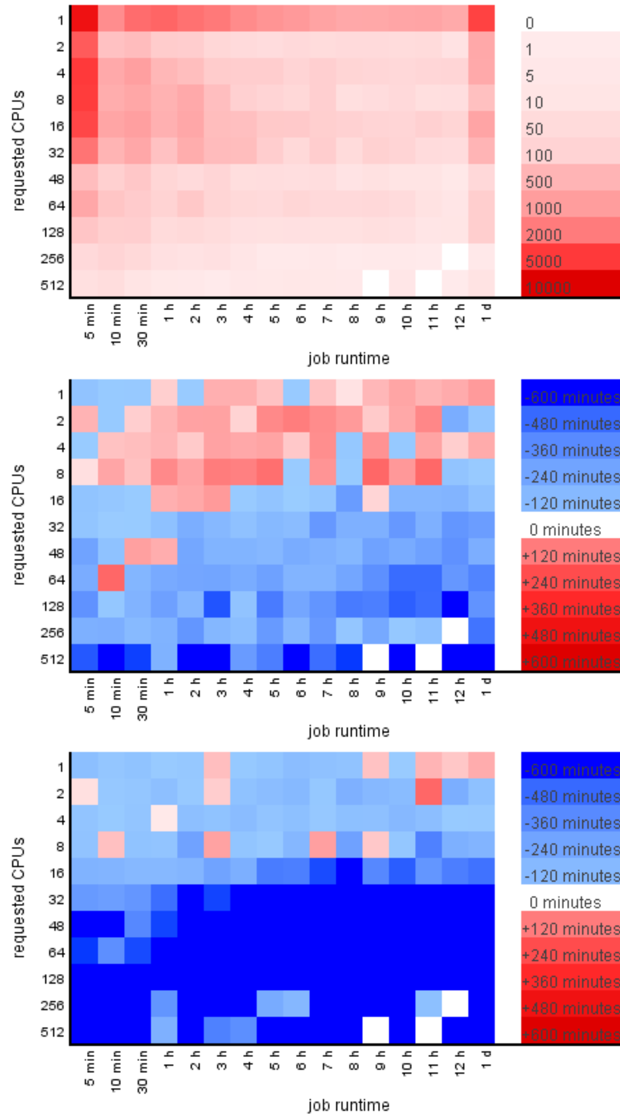

Fig. 12. CTC SP2. Job distribution heatmap (top), avg. job wait time difference heatmaps for *baseline* vs. *estimated* (middle) and *baseline* vs. *min. diff.* (bottom).

performance for nearly all job sizes, as demonstrates the bottom heatmap in Figure 12. Again, this heatmap corresponds to the bad average wait time result observed in Figure 2.

4.5 Discussion

The experiments presented in Sections 4.3 and 4.4 clearly demonstrated that for some workloads (and actual systems) even trivially constructed soft walltimes may represent an interesting option compared to the imprecise, coarse-grained user-provided estimates. On the other hand, it seem very likely that systems that (A) have reasonably variable user runtime estimates, and (B) do not contain long jobs, will not gain much improvement from presented techniques. This is not surprising, because such workloads already offer a lot of backfilling opportunities. Perhaps, it will be worth trying soft walltimes in such systems if some more accurate prediction method is available.

One of the interesting features of soft walltimes is that they do not directly rely on a user's (un)willingness to provide reasonable guesses. In a saturated system, where users compete for resources it would be very probable that some users would try to *cheat the scheduler*, by providing long walltime limits and (very) short soft walltimes. Without further "anti-cheating" mechanisms these users would be favored by the backfilling and would degrade both the accuracy and fairness of the system. Therefore, the use of soft walltimes provided by some "black box solution" will be more safe and fair than if users are allowed to directly specify both walltimes and soft walltimes. Sure, since prediction techniques usually use historic information concerning job runtimes, users can still "game" the prediction system in an indirect fashion, e.g., by submitting large numbers of very short jobs before submitting a very long computation. It is then up to the system administrator to develop some form of an anti-cheating policy.

5 Conclusion and Future Work

In this paper, we have analyzed the suitability and impact of using soft walltimes in parallel job scheduling. Soft walltimes clearly represent an interesting solution for systems where users are either unable or unwilling to provide reasonably heterogeneous and/or accurate estimates. In such systems even very simple runtime prediction techniques (like those used in this paper) can significantly improve the performance of the scheduler through increasing the portfolio of jobs suitable for backfilling.

On the other hand, our simulation results suggest that soft walltimes do not improve the performance in systems where the user-provided runtime estimates are already reasonably accurate or at least exhibit great variability. In such cases our simple prediction techniques fail to deliver better solutions.

In the future, it would be very interesting to analyze whether more advanced prediction techniques can deliver better performance. Here, it is important to define what "better perfomance" means. Using our own experience as well as the observations of other researchers it is highly likely that "better estimates" will not always produce, e.g., better average wait time or slowdown. This is obvious from the fact, that there are several examples (workloads) where the use of perfect estimates (estimate = runtime) leads to worse average wait time and/or

slowdown, compared to the setup where the user-provided inaccurate estimates are used instead. However, there are other important metrics that can benefit from improved estimates. For example, the reliability of predicted start times of jobs in a queue will improve if more accurate estimates are used. For some use cases this improved predictability may represent more important role than, e.g., the wait time.

Last but not least, we see another promising direction of using soft walltimes in systems with heterogeneous resources that have (significantly) different speeds of CPUs, GPUs and I/O devices. MetaCentrum and CERIT-SC represent such heterogeneous systems where different clusters are of different age and use different CPU/GPU architectures with big differences in processing and I/O speeds (local SSD scratch disks vs. local HDD vs. shared file system). In such systems, it would be interesting to adapt user-provided estimates and/or soft walltimes with respect to the actual machine being used. Our observations suggest that the runtime of CPU/GPU intensive applications significantly decreases on modern CPUs/GPUs while I/O intensive applications benefit from using fast local SSD scratch disks. It seems promising to use a prediction technique to adapt soft walltimes dynamically according to the performance characteristics of available resources and expected application's CPU/GPU and I/O demands.

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

References

1. Alea 4: Job scheduling simulator, February 2018. `https://github.com/aleasimulator`.
2. V. Balasundaram, G. Fox, K. Kennedy, and U. Kremer. A static performance estimator to guide data partitioning decisions. *ACM SIGPLAN Notices*, 26(7):213–223, 1991.
3. CERIT Scientific Cloud, February 2018. `http://www.cerit-sc.cz`.
4. 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.
5. M. V. Devarakonda and R. K. Iyer. Predictability of process resource usage: A measurement based study on UNIX. *IEEE Transactions on Software Engineering*, 15(12):1579–1586, 1989.
6. A. B. Downey. Predicting queue times on space-sharing parallel computers. In *11th International Parallel Processing Symposium*, pages 209–218, 1997.

7. C. Ernemann, V. Hamscher, and R. Yahyapour. Benefits of global Grid computing for job scheduling. In *GRID '04: Proceedings of the 5th IEEE/ACM International Workshop on Grid Computing*, pages 374–379. IEEE, 2004.
8. D. G. Feitelson. Parallel workloads archive, February 2018. `http://www.cs.huji.ac.il/labs/parallel/workload/`.
9. 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.
10. 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.
11. F. Guim, J. Corbalan, and J. Labarta. Prediction f based models for evaluating backfilling scheduling policies. In *Eighth International Conference on Parallel and Distributed Computing, Applications and Technologies (PDCAT 2007)*, pages 9–17. IEEE, 2007.
12. D. Klusáček. Workload traces from metacentrum and CERIT Scientific Cloud, February 2018. `http://jsspp.org/workload/`.
13. 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.
14. D. Krakov and D. G. Feitelson. Comparing performance heatmaps. In N. Desai and W. Cirne, editors, *Job Scheduling Strategies for Parallel Processing*, volume 8429 of *LNCS*, pages 42–61. Springer-Verlag, 2013.
15. R. Kumar and S. Vadhiyar. Prediction of queue waiting times for metascheduling on parallel batch systems. In W. Cirne and N. Desai, editors, *Job Scheduling Strategies for Parallel Processing*, pages 108–128. Springer-Verlag, 2014. Lect. Notes Comput. Sci. vol. 8828.
16. 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.
17. MetaCentrum, February 2018. `http://www.metacentrum.cz/`.
18. 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.
19. 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.
20. PBS Works. *PBS Professional 14.2, Administrator's Guide*, February 2018. `http://www.pbsworks.com`.
21. V. Sarkar. Determining average program execution times and their variance. In *ACM SIGPLAN Conference on Programming Language Design and Implementation*, pages 298–312, 1989.
22. 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.
23. J. Skovira, W. Chan, H. Zhou, and D. Lifka. The EASY – LoadLeveler API project. In D. G. Feitelson and L. Rudolph, editors, *Job Scheduling Strategies for Parallel Processing*, volume 1162 of *LNCS*, pages 41–47. Springer, 1996.

24. W. Smith, I. Foster, and V. Taylor. Predicting application run times using historical information. In D. G. Feitelson and L. Rudolph, editors, *Job Scheduling Strategies for Parallel Processing*, volume 1459 of *LNCS*, pages 122–142. Springer, 1998.
25. 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, 1999.
26. Soft walltime documentation, February 2018. `https://pbspro.atlassian.net/wiki/spaces/PD/pages/42532871/PP-482+Soft+Walltime`.
27. D. Talby and D. G. Feitelson. Supporting priorities and improving utilization of the IBM SP scheduler using slack-based backfilling. In *IPPS '99/SPDP '99: Proceedings of the 13th International Symposium on Parallel Processing and the 10th Symposium on Parallel and Distributed Processing*, pages 513–517. IEEE Computer Society, 1999.
28. 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.
29. 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.
30. D. Tsafrir, Y. Etsion, and D. G. Feitelson. Modeling user runtime estimates. In D. G. Feitelson, E. Frachtenberg, L. Rudolph, and U. Schwiegelshohn, editors, *Job Scheduling Strategies for Parallel Processing*, volume 3834 of *LNCS*, pages 1–35. Springer Verlag, 2005.
31. N. Zakay and D. G. Feitelson. Preserving user behavior characteristics in trace-based simulation of parallel job scheduling. In *22nd Modeling, Anal. & Simulation of Comput. & Telecomm. Syst. (MASCOTS)*, pages 51–60, 2014.