Optimization Metrics for the Evaluation of Batch Schedulers in HPC

Robin Boëzennec[1], Fanny Dufossé[2], and Guillaume Pallez[3]

[1] Inria, Labri, University of Bordeaux, France
[2] Univ. Grenoble Alpes, Inria, CNRS, Grenoble INP, LIG, 38000 Grenoble, France
[3] Inria, Université de Rennes, France

Abstract. Machine Learning techniques are taking a prominent position in the design of system softwares. In HPC, many work are proposing to use such techniques (specifically Reinforcement Learning) to improve the performance of batch schedulers.
Their main limitation is the lack of transparency of their decision. This underlines the importance of choosing correctly the optimization criteria when evaluating these solutions. In this work, we discuss bias and limitations of the most frequent optimization metrics in the literature. We provide elements on how to evaluate performance when studying HPC batch scheduling. We also propose a new metric: the standard deviation of the utilization, which we believe can be used when the utilization reaches its limits.
We then experimentally evaluate these limitations by focusing on the use-case of runtime estimates. One of the information that HPC batch schedulers use to schedule jobs on the available resources is user runtime estimates: an estimation provided by the user of how long their job will run on the machine. These estimates are known to be inaccurate, hence many work have focused on improving runtime prediction.

Keywords: Metric · Online Scheduling · High Performance Computing · Runtime estimates · EASY

1 Introduction

In recent years, most of industrial and academic large experiments have been replaced or simultaneously simulated with large scale applications. This development corresponds to the increase of massively parallel computing platforms. The High Performance Computing (HPC) platforms are typically developed to execute such massive computing jobs. The largest platform, Frontier [2] has recently reached the exascale level, that was envisioned by parallel computing research community for decades.

Large HPC platforms encounter a high rate of submission of new jobs that lead to a percentage of utilization over 90%. This implies a capacity to allocates these jobs fast enough to avoid congestion. The scheduling techniques were studied for decades and mainly lead to a characteristic of the most efficient scheduling techniques: their simplicity. The most sophisticated methods using machine

learning approach [5, 9] can barely match the performance of nowadays classical scheduling methods such as Easy-BF.

Easy-BF mainly consists in a FCFS approach completed by a backfilling step. Jobs are considered in their order of submission and assigned as soon as the requested number of servers are available. This method creates some utilization *gaps* (i.e. idle nodes). To avoid such idle time, in a second step, the queue of waiting jobs is considered to find one or more jobs that can fill the gap and being executed without delaying the first come job.

Machine Learning techniques are taking a prominent position in the design of system softwares. In HPC, many work are proposing to use such techniques (specifically Reinforcement Learning) to improve the performance of batch schedulers [23, 24]. To motivate the importance of these strategies, the authors often focus on a selected optimization criteria such as the mean bounded slowdown, without looking *under the hood* at what these black box algorithms do.

The question of scheduling performance is ambiguous. For each job, the objective is obviously its finish time, but the performance has to be evaluated globally. Two kind of criteria are mainly considered. The percentage of utilization or the maximum arrival rate of jobs are platform oriented objectives. The users oriented objectives use average or weighted average values of a formula that combine the waiting time and the runtime or the walltime of each job. The selection of the criteria is a critical issue in scheduling, as some criteria are associated to a priority order of jobs based on their duration or their number of nodes. For example, the objective of average completion time is minimized with ordering jobs by shortest runtime first. This is not consistent with platforms dimensioned for jobs requesting a large number of servers for a long duration.

In this work, we discuss more thoroughly the use of quantitative optimization criteria for the design of HPC Resource and Job Management Software (aka Batch Schedulers). Specifically, we precise how some criteria hide effects that could be seen as negative from an HPC perspective.

Our main contributions are the following:

1. We give a qualitative analysis of many optimization criteria used in the literature. We stress how most of the recent work that have claimed to improve the performance of Easy-BF have done so on criteria that we show to be extremely biased towards very small jobs. This is the case of many recent work that have proposed to use deep-learning for resource management and that have not proposed a qualitative analysis of their results.
2. We demonstrate the statements from this analysis by studying the classical Easy-BF algorithm on two workloads (Mira and Theta) with two runtime estimate functions: a very precise one, and the actual runtime estimate provided by users. This section confirms that without performing a qualitative analysis and by simply looking at specific performance, one cannot conclude a study.

In Section 2, we discuss the objective criteria that are considered in scheduling framework. We focus on their limitations for HPC systems, and provide alternatives to improve them. We then work on demonstrating experimentally

our statements. Section 3.1 details the methodology of the different evaluations before presenting and analyzing the results in Section 3.2. Section 4 discusses related work. Finally, Section 5 concludes the work.

2 Evaluating the quality of a schedule

Several optimization criteria are used to evaluate the performance of a Resource and Job Management Software. In this Section, we discuss more in depth those objectives, particularly in the context of High-Performance Computing. We explain their limitations in this context.

The analysis presented in this work is targeted for High-Performance Computing: building a machine able to perform ExaFlops targets the execution of large scale applications mostly and the validation of the performance of a solution should reflect this. Extreme-scale platforms have a high operating cost and are expected to be utilized as much as possible.

Analysis of HPC system traces showed that *Users are now submitting medium-sized jobs because the wait times for larger sizes tend to be longer* [17]. To execute medium-size jobs, it is probably more efficient (cost-wise) to have multiple smaller clusters than an HPC machine with a dense interconnect.

In order to define the objective we use the following notations for job J_i:

r_i	The release time of job J_i (also called submission time)
C_i	The completion time of job J_i
t_i^{real}	The real length of job J_i (also called execution time)
$t_i^{\text{wait}} = C_i - r_i - t_i^{\text{real}}$	The waiting time of job J_i

2.1 Mean (bounded) slowdown

The average slowdown (also called mean flow) is the main optimization criteria in many recent works on improving resource management in HPC [14, 4, 24]. Its goal is to provide a measure of fairness over applications.

The slowdown S_i of job J_i (also called the flow of the job) corresponds to the ratio of the time it spent in the system over its real execution time. Formally, it is defined as

$$S_i = \frac{t_i^{\text{real}} + t_i^{\text{wait}}}{t_i^{\text{real}}} = \frac{C_i - r_i}{t_i^{\text{real}}}$$

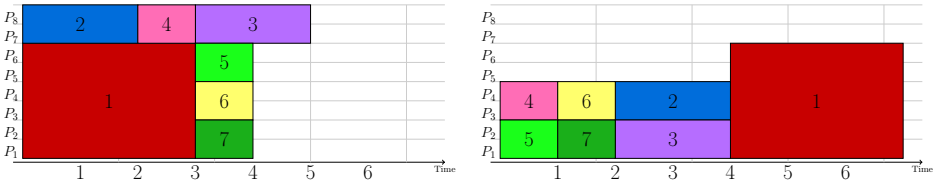
Note that in practice many jobs are extremely small (few seconds). In these cases their slowdown could be arbitrarily high even if their wait time is ridiculously small (a five minutes wait time for a job that dies instantly (one second) would result in a slowdown of 300).

The solution that is often used is to consider a variant of the slowdown called the *bounded slowdown*:

$$S_i^b = \max\left(\frac{C_i - r_i}{\max\left(t_i^{\text{real}}, \tau\right)}, 1\right) \tag{1}$$

where τ is a constant that prevents the slowdown of smaller jobs from surging. Then the average bounded slowdown $\bar{S}$ is:

$$\bar{S^b} = \frac{1}{n}\sum_i S_i^b, \quad \text{where } n \text{ is the number of jobs}$$



(a) Response time ≈ 3.6, slowdown ≈ 2.8 (b) Response time $= 3$, slowdown ≈ 1.8

Fig. 1: In this example, all jobs are released at $t = 0$. Despite what appears to be a more efficient strategy, the left schedule has worse average response time and slowdown than the right schedule.

Limits for HPC workloads By improving the quality of service to the small jobs, one can considerably improve this objective. This is often what is actually measured when work study this objective, and is the opposite of what a system administrator of an HPC machine is looking for. This is illustrated in Figure 1.

In Section 3.2, we also show that this is subject to a high variability, and hence the performance is highly dependent on the input data.

Alternative approach To understand the actual behavior of the system, some work [7] consider the bounded slowdown as a function of the size of the job. In this case, this objective is not one to optimize anymore, but more a qualitative way to measure and understand the performance of a solution. Another approach is to use a weighted version of the average slowdown where large jobs are given more weight than smaller jobs.

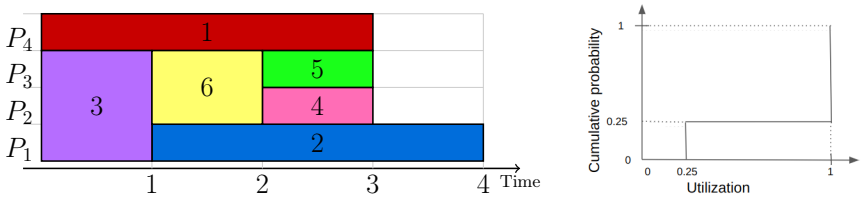
2.2 Utilization

This optimization criteria measures how fully the platform is occupied. It is a particularly important objective for an HPC platform that costs multiple-million of dollars yearly to operate. This is the main objective studied in [10–12].

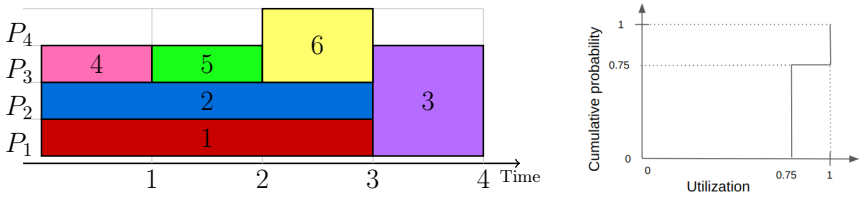
If $W(t_1, t_2)$ is the total amount of work done between t_1 and t_2 on a platform with N nodes, the utilization $U(t_1, t_2)$ on the interval $[t_1, t_2]$ is measured as:

$$U(t_1, t_2) = \frac{W(t_1, t_2)}{N \cdot (t_2 - t_1)}. \tag{2}$$

Note that when jobs fail to complete fully (for instance because their walltime is underestimated), it is interesting to measure the "useful utilization", i.e. the volume of computation that lead to a successful execution [7] .



(a) Schedule example and its corresponding cumulative density function



(b) Schedule example and its corresponding cumulative density function

Fig. 2: Even though the global utilization is the same between the two schedules (13/16), the distribution of their utilization differ significantly.

Limits for HPC workloads One of the main limitation concerns machines with lower submission rate (i.e. that are not "packed"), then any scheduling solution has the same (low) utilization since it corresponds to executing almost all jobs during the whole window. Utilization by itself does not allow to discriminate between different schedule qualities (Figure 2).

Another one is the fact that it is more a system administrator target: how to maximize the yield of my machine. It does not give a sense of the quality of the schedule: an easy way to maximize utilization would be to have a large queue of jobs waiting to be executed and find the one that works best at all time (often favoring smaller jobs that can fill a hole).

Alternative approach Our observations show that in some scenarios if the utilization of an HPC platform is lower than 93%, the "quality" of a scheduler has no impact on the average utilization of the schedule.

There are settings where the workload has different "modes" (such as intensive in the day; low on requests in the night), in this case it may be interesting to study utilization of these workloads separately. Having a good understanding of one's workload is important.

We found that a way to measure this is to study the density function of the utilization (see Figure 2). Indeed intuitively, for two identical job submission

schemes a "better" scheduling algorithm will have more phases at very high utilization (and hence more at lower utilization). Indeed, it can pack jobs as soon as they are available, whereas a poorer scheduling quality will delay jobs from phases of time with intensive job counts to phases with less intensive job counts.

When two schedules have an almost identical utilization (because all jobs are scheduled in the same time window), we propose to measure the variance of the utilization as a way to differentiate the quality of a schedule: the "best" algorithm from a utilization perspective should have a higher variance (more time-windows with very high occupation and more time-windows with low occupation). For example the schedule in Figure 2a is better at using all available resources at the same time, leading to a variance of utilization 9 times greater than the schedule in Figure 2b.

Some remarks on using the variance:

1. The standard deviation of utilization is not a new metric independent of the utilization. It can be used to compare schedulers when the system is under-utilized (and then utilization can not discriminate algorithms), to give information about which algorithm would be able to reach the higher utilization once the system becomes fully utilized.
2. This metric allows to qualify whether one schedule is better than another one from a utilization perspective, but it lacks interpretability: what does having a variance x times greater than another one means overall? This is an open question for us.
3. It is important to note that the variance is only relevant to compare schedules with a similar utilization. If it is not the case, one can just tell which schedule is better by looking at the utilization.

2.3 Response time (and Wait Time)

Mean response time (or mean wait time) is a metric often used in the literature [10, 12, 21, 16, 19, 23]. The response time $\mathcal{RT}_i$ of a job J_i is the duration between the submission of the job and its completion, or equivalently its wait time and its length.

$$\mathcal{RT}_i = t_i^{\text{wait}} + t_i^{\text{real}}$$

The mean response time is equivalent to the mean wait time since the difference is the mean runtime which depends on the workloads but not on the schedule. In the following, we only address the response time, but our reasoning identically apply to wait time.

Limits for HPC workloads Using this objective gives equal importance to all jobs, independently of the work they represent. In an HPC workload, this gives an advantage to the numerous "small" jobs, even if they only represent a very small portion of the workload. In Figure 1 we can see that the scheduling on the top intuitively looks more efficient than the second, and yet it has a worst mean

response time. This is because the scheduling on the bottom favors small jobs despite being less effective at densely packing jobs.

Similarly to the mean bounded slowdown, we show in Section 3.2 that when using a workload from a big compute center (with a high jobs sizes variability), the mean response time is mainly going to be influenced by the proportion of very small jobs which are backfilled (and then have a very low response time). In addition to not corresponding to what we want to optimize, it is also subject to a lot of variability and is quite random which is going to disturb the evaluation of performance. This is something that was confirmed by our experiments and which is covered more in depth in Section 3.2.

In the end, this is a limit for the response time objective because simply improving it does not necessarily mean improving the quality of the overall schedule (from an HPC perspective).

Alternative approach Goponenko et al. [13] have used the weighted mean response time, where one weights the response time by a priority (such as the total amount of work of a job, or the number of nodes that a processor uses). We argue that when computing the average response time in the context of an HPC job scheduler, one should give a higher weight to bigger jobs. For example giving a weight proportional to the area of the job would allow to transform the mean response time in a system administrator metric: as Goponenko et al. [13] underlined, using this weight would mean swapping a job with two smaller jobs of the same duration but half the resources would not change the weighted response time. This way neither small nor big jobs are favored, and what is measured is the ability of the scheduler to densely pack jobs.

Alternatively, Gainaru et al. [12] have proposed to measure only the response time of non-backfilled jobs.

2.4 Additional comments

As we said, many objectives when optimized have negative side effect for large scale platform (such as improving the performance of small jobs at the detriment of large jobs).

Yet many work, particularly recent work that discuss improving batch-scheduling techniques using machine learning still optimize these objectives. As an example, recent research directions have focused on using RL-based scheduling in batch schedulers [23, 24]. They show that by using RL into the batch scheduling, one can improve considerably the response time and bounded slowdown at a small cost in utilization.

By simply looking at the objective function, their analysis lacks quality elements that could show the limits of their performance as discussed in the previous section. Specifically it is very likely that their significant improvement in response time or slowdown are mostly used by the improvement of the slowdown/response time of small jobs, which may be done at the detriment of those of larger jobs.

Work by Carastan-Santos et al. [4] where the ML algorithm provides a priority function confirms this intuition and the fact that learning-based batch schedulers with the objective of bounded slowdown simply give higher priority to small jobs. Similarly, Legrand et al. [14] have realized the importance of small jobs for bounded slowdown and focus on having an oracle which guesses which job is small and which is large. This is sufficient for significant performance gains for this objective.

3 Use-case: the impact of runtime estimates

HPC Resource and Job Management Systems rely on user-submitted runtime estimate functions. These estimates are known to be inaccurate. Many work [3, 17] have focused on improving runtime prediction.

To demonstrate the risk of evaluating quantitatively a schedule, we propose to evaluate the performance of two notable runtime estimate functions: a perfect estimate, and the estimate provided by the users (see Section 3.1). The results on metrics detailed in Section 2 are detailed in Section 3.2.

3.1 Evaluation Methodology

We simulate the execution of Easy-BF using the batch simulator Batsim on the workloads of platforms Mira and Theta.

Batsim Simulations are run in Batsim [8] (version 4.1.0), a simulator to analyze batch schedulers with the Easy-BF version of the algorithm easy_bf_fast from Batsched (version 1.4.0). easy_bf_fast is an online scheduler. Batsched is a set of Batsim-compatible algorithms implemented in C++.

Our most intense simulations (compute-wise) execute 10 000 jobs on 49 152 nodes, which corresponds to Mira's characteristics. A single simulation with this setup takes about 10 minutes to complete on a laptop with a processor intel i5-8350U. It has 4 cores, 8 threads, a max frequency of 3.6GHz, and 6MB of cache.

Workloads We used traces from computers Mira and Theta [1] of the Argonne National Laboratory. The Mira supercomputer was launched in 2012 at the 3rd place of TOP500 [2] HPC centers. It ran 49 152 nodes (and 16 times more cores) and was maintained until 2019. The available trace covers years 2014 to 2018. It contains a total of 330k jobs.

The Theta platform was launched in 2016 and runs 4 392 nodes (and 64 times more cores) with traces from 2017 to 2022. We have not used the first year of Theta because the number of cores used was varying. Without this year, the trace contains about 420k jobs.

In both cases, system admins were giving incentives to users to request a number of nodes which is an integer power of two, that is nearly always the case [17].

For the evaluations, we create a total of 70 inputs by partitioning the traces in sets of 10k consecutive jobs (30 for Mira, 40 for Theta): we sorted traces by submission time, and we used the jobs from index 1 to 300 000 (by slice of 10 000) for Mira, and from index 1 to 400 000 (by slice of 10 000) for Theta.

These samples provide a wide variability of workloads: on Mira they span from 12 days of consecutive submissions to 110 days, with a mean duration of 59 days, while on Theta they span from 15 to 61 days with a mean of 40 days.

The workloads are then constructed as follows. Consider the jobs sorted by their release times:

1. We study the workload starting from t_{1001}, the release time of its 1001^{st} job;
2. The 1000 first jobs are used to have a non-empty queue at the beginning of the analysis, and hence all their release times are set to t_{1001}.

Measuring performance Since we simulate subset of the traces, we need to prune the traces for the performance evaluation in order to remove possible side-effects that may not be representative.

Utilization related objectives The utilization and its standard deviation are measured on a given time window as presented in Equation (2). If only a part of a job is inside the window, we ignore the part of the job that is outside the window. To remove side-effects, we crop the borders of the execution window to measure performances when the scheduler is in its steady state. This is consistent with the literature [21]. The measure window is defined as

$$[t_1, t_2] = [0.15 \cdot \text{max\_submission\_time}, 0.85 \cdot \text{max\_submission\_time}].$$

Bounded slowdown and Response time When computing these objectives, we do not include the performance of the first and last 15% jobs to measure the performance of the steady state (For details : the 10% jobs in the initial queue are included in these 15%, which mean that at the start we crop the 10% in the initial queue plus the first 5% scheduled jobs.). We use $\tau = 10$s for the bounded slowdown (Equation (1)), following the literature [21].

Relative improvement for a given metric In the evaluation we discuss the *relative improvement of* Exact *over* User-walltime *for an objective* O (which we abbreviate sometimes as *Relative Improvement*). This relative improvement $\text{RI}(O)$ is measured as:

– If O is a maximization objective (e.g. utilization, utilization standard deviation), then

$$\text{RI}(O) = \frac{O_{\text{Exact}} - O_{\text{User-walltime}}}{O_{\text{User-walltime}}} \tag{3}$$

– If O is a minimization objective (e.g. response time, bounded slowdown), then

$$\text{RI}(O) = \frac{O_{\text{User-walltime}} - O_{\text{Exact}}}{O_{\text{User-walltime}}} \tag{4}$$

This difference allows to see clearly that when $\mathrm{RI}(O) > 0$ then Exact performs better than User-walltime by a factor $\mathrm{RI}(O)$ on objective O, while when $\mathrm{RI}(O) < 0$, then Exact performs worse than User-walltime by a factor $-\mathrm{RI}(O)$ on objective O.

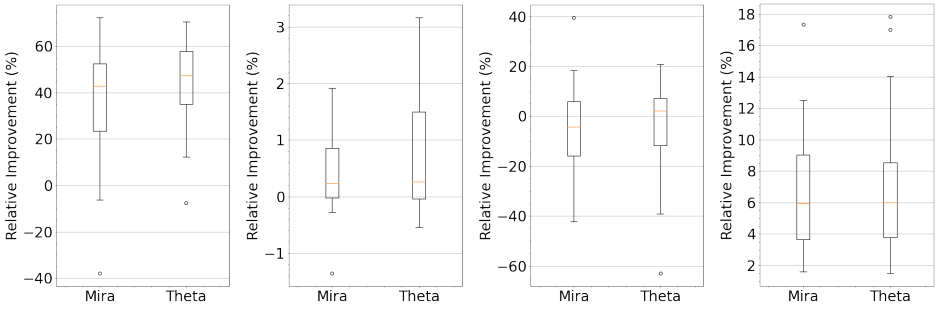
Runtime estimate functions The goal of this work is to evaluate the impact of the precision of runtime estimates. Hence we define several walltime functions. Given r the runtime of a job (in seconds):

- Exact : $r \mapsto r + 1\text{second}$. This simulates an almost perfect estimate (except for extremely small jobs, but this simplifies the interaction with Batsim).
- User-walltime, it corresponds to the walltime provided by the users.

3.2 Experimental evaluation

In this section we simulate a study of the impact of walltime accuracy in the performance of Resource and Job Management Software in order to discuss various metrics.

Coarse-grain analysis The two runtime estimate functions are evaluated in Figure 3 over the various criteria discussed in Section 2: Bounded Slowdown (Section 2.1), Utilization (Section 2.2), Response Time and Weighted Response time (Section 2.3). In these figures, to discuss the performance difference between the two runtime estimates functions, we use the relative improvement of Exact over User-walltime as computed by Equations (3) and (4).



(a) Mean Bounded Slowdown (b) Utilizaton (c) Mean Response Time (d) Mean Weighted Response Time

Fig. 3: Relative improvement of Exact over User-walltime for different metrics. The subfigures have different scales. Utilization is a maximization objective while the others are minimization objectives.

The quantitative results in Figure 3 demonstrate the importance of the selection of the objective function. If we study the bounded slowdown, having

a perfect estimate of the walltime seems to improve the performance of the Ressources and Job Management Systems (RJMS) by almost 50% which seems remarkable. Utilization-wise, the performance only sees marginal performance improvement (approximately 0.5%). On the contrary, it seems that knowing in advance precisely the runtime of an application can be detrimental to the response time of the machine (approx. 7% decrease of performance for Mira), but it does not hold if we measure the weighted response time.

Going deeper into these figures, we want to talk about the variability of the performance. Indeed, this is particularly important: discussing the mean of an objective when the variability is highly variable is meaningless: it means that the performance is highly influenced by the workload used as input. This is the case for bounded slowdown (Fig 3a), as well as the response time (Fig 3c). We believe that this is because these objectives are highly influenced by the number of *small* jobs available in the input workload. **Hence, the mean Bounded slowdown and mean Response Time cannot be used as performance objectives for the evaluation of RJMS.**

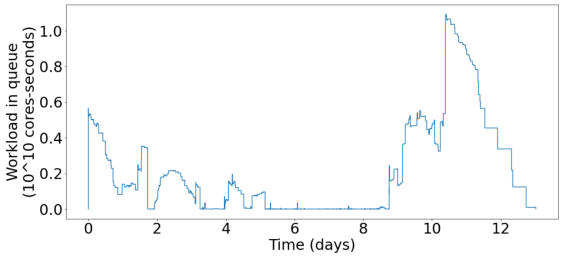
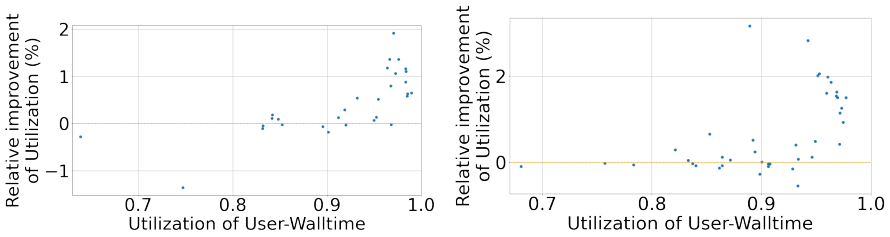


Fig. 4: Available work in the waiting queue as a function of time for one of workload sample from Mira when scheduled with the Exact walltime function. This shows that the availability of jobs is not regular throughout the execution.

With respect to the utilization 3b, it seems that the improvement is extremely small (about 0.5%). As we explained in Section 2.2, this may not be surprising and is an artifact of the non-constant arrival rate (see Figure 4). When there is a low utilization (and low arrival-rate), all jobs end-up being executed within the measured time-window even with poor packing quality. To demonstrate this, we plot in Figure 5 the relative improvement of the utilization as a function of the Utilization of User-walltime.

The measured presented in Figure 5 confirms our intuition: when the utilization is below 93% there is almost no utilization improvement, while for high-utilization periods (above 95% utilization), Exact improves the system utilization by 1-2%. Of course above 95% utilization the gap available for improvement is extremely small, and it is hard to use this improvement to quantitatively compare several solutions (different algorithms or in this case the impact of better



(a) Data from Mira (b) Data from Theta

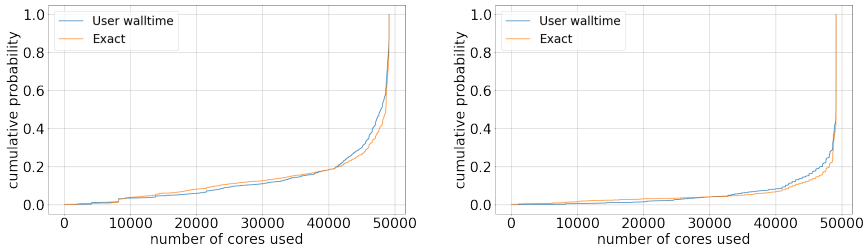
Fig. 5: Relative improvement of the Utilization of EXACT over USER-WALLTIME as a function of the Utilization

runtime estimates). This shows the limits of the utilization as an objective to compare two solutions.

These elements underline the fact that simply looking at an objective function without further analysis is risky. We showed that one should not use the average bounded slowdown or average response time as an optimization objective to measure the performance of an HPC batch scheduling algorithm.

Towards better objective

Qualitative analysis of the utilization performance As discussed in Section 2.2, the fact that two algorithms have the same utilization does not mean that they are equivalent but it is more an artefact of the workload. In Figure 6, we show the density distribution of EXACT and USER-WALLTIME for two workloads where the relative difference in utilization is almost null.



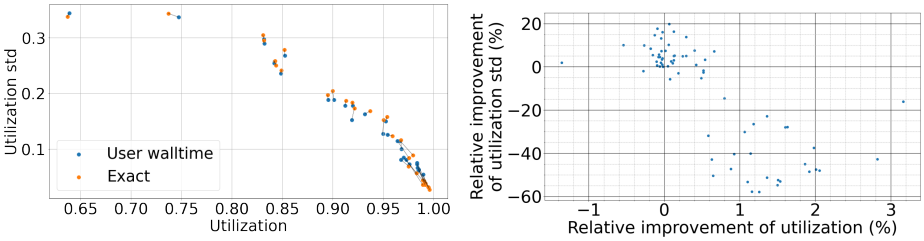
(a) Utilization is 0.92 (RI: 0.29%). (b) Utilization is 0.95 (RI: 0.07%).

Fig. 6: Cumulative distribution functions of the utilization for two selected scenarios from Mira where the difference in utilization between EXACT and USER-WALLTIME is close to 0.

We observe from Figure 6 that the solution that uses perfect estimation of walltime functions has more scenarios with extremely low utilization and more

with higher utilization. We interpret it as a better management of peaks of submissions, hence that the solution generally performs better job packing. This would be consistent with the fact that the algorithm using Exact performs better in periods of very dense utilization ($> 95\%$).

As a quantitative objective to be able to compare various algorithms, we propose to measure the standard deviation (std) of utilization. For two algorithm with identical utilization, high standard deviation imply large variation of utilization, that we correlate with more periods of high utilization and more periods of low utilization. We study the standard deviation of utilization in Figure 7.



(a) Utilization and its standard deviation of each scenario for both algorithms (Mira Data).

(b) Relative improvement of the standard deviation of the utilization as a function of the relative improvement of the utilization (Mira and Theta data).

Fig. 7: Comparison of utilization and standard deviation of the utilization of Exact and User-walltime

On Figure 7, we have plotted the utilization of all workload inputs and their associated standard deviation, even if the ones that are interesting are the ones when the relative improvement of utilization is almost zero. In Figure 7b we can see that when the relative improvement utilization difference is between -0.5% and 0.5%, the relative difference of the standard deviation of utilization can be up to 20%! This confirms the strong impact of using correct runtime estimates on the utilization of the platform and leaves a lot of room for other solutions.

Qualitative analysis of the response time performance We concluded earlier that the mean response time and mean average bounded slowdown are poor objectives to qualify a schedule because of their high variability. They can however be used to understand the quality of a schedule with respect to fairness to individual jobs: how can we analyze the impact of having a better runtime estimate?

To start this discussion, we remind that when looking at the *mean response time* as an optimization objective, the mean improvement of using a precise runtime estimate was almost null, and that it had high variability. While one could have concluded that improving the runtime estimate led to almost no change, we show how one could study this qualitatively here.

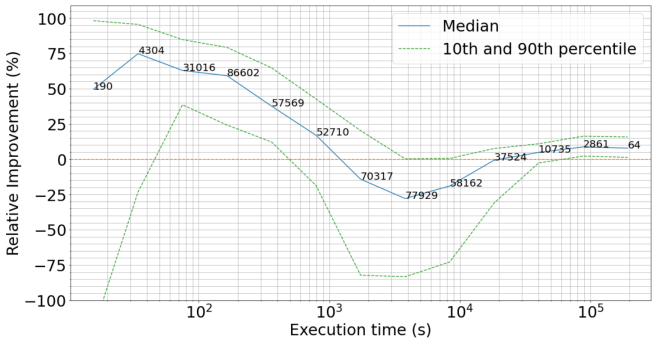


Fig. 8: Median Relative Improvement of the response time for Mira and Theta jobs as a function of t^{real}. The numbers on the blue line are the numbers of jobs in each group.

Figures 8, 9a and 9b plot the median relative improvement of response time, function of some job parameters: their execution time, their number of nodes, and the ratio Execution time over user-predicted Walltime (t_i^{estimate}). To create groups of jobs with a similar value for a parameter A, we take the interval $[\min(A), \max(A)]$ and divide it in 18 subgroups of equal length after scaling, i.e.: when a geometric (resp. linear) scale is used for the x axis, it means that the interval $[\min(A), \max(A)]$ was divided in 18 groups of same size on a geometric (resp. linear) scale. We then only plot results of groups with more than 50 elements.

For example, the first figure splits the set of jobs into 18 groups of jobs with similar execution time (only 16 are displayed because 2 groups contain less than 50 jobs). The blue line corresponds to the median relative improvement of the mean response time over the 70 samples (30 from Mira and 40 from Theta).

The values printed on the blue line are the number of jobs in each group. The green dotted lines correspond to the first and last decile. When computing the median and the deciles, we did not take in account the samples of Mira and Theta for which the group was empty.

In Figure 8 we observe that the impact of having a correct prediction is highly correlated to the job execution time. Specifically we observe three groups: short jobs, medium-length jobs and long jobs. When switching from User-walltime to Exact:

- small jobs have a lower average response time when the precision of their runtime estimate improves Exact;
- medium-length jobs have an increased average response time;
- longer jobs benefit of a little improvement of their response time.

Again, studying these objectives, one should be concerned about the extremely high variance of the performance.

With these precisions, one then can try to understand the impact of the solution. In this case we suppose that this is because of backfilling: medium-

length jobs are less backfilled than they were when the runtime estimate of longer jobs is really wrong. On the contrary, small jobs are easier to backfill in small schedule gaps. Finally long jobs (which are almost never backfilled) benefit from the fact that fewer medium-lengthed take priority over them. We can verify this intuition by measuring the number of jobs that are indeed backfilled.

	All	$Rt < 1e3$	$1e3 < Rt < 2e4$	$2e4 < Rt$
Backfilled with User-walltime	209230 (75%)	115998 (82%)	90357 (71%)	2845 (26%)
Backfilled with Exact	199200 (71%)	117652 (83%)	80040 (63%)	1479 (14%)
Number of jobs	279990	141903	127218	10828

Table 1: Jobs backfilled with User-walltime and Exact walltime functions in function of their runtime (Theta data). We removed the first and last 15% jobs as they are not used to compute response time.

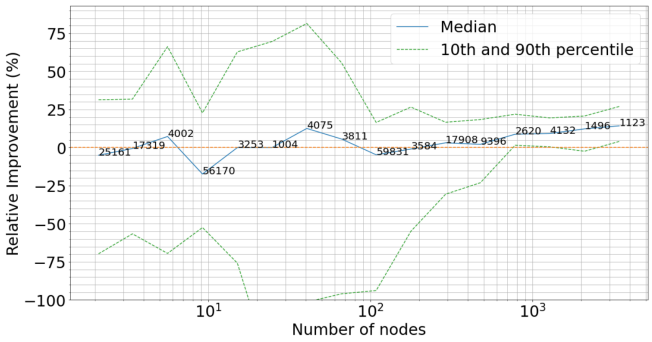
Table 1 investigates the correlation between jobs runtime and response time by comparing the number of backfilled jobs in the three categories identified on Figure 8. We consider that a job J is backfilled if and only if there is another job B which has a lower submission time than J and starts after J on at least one of the nodes J used. The impact of Exact walltime function is to reduce the number of medium and long backfilled jobs. The ratio of backfilled short jobs is almost the same, but the removal of the largest jobs permit the large improvement in response time observed in Figure 8. Medium runtime jobs are thus negatively impacted by Exact walltime function. Finally, even if they are twice less backfilled, the response time of long jobs is slightly decreased due to a better packing of Exact.

Note, we believe that backfilling could also explain the high variance in response time improvement for job response-time: we expect that the 10^{th} percentile of small jobs with $RI < -100\%$ corresponds to jobs that are backfilled with User-walltime but not backfilled with Exact. The thorough study of this behavior becomes out of the scope of this work: we try to focus on *how to* analyzing performance via the study of optimization objectives, however, if one was studying the impact of runtime estimates, this variability should be analyzed.

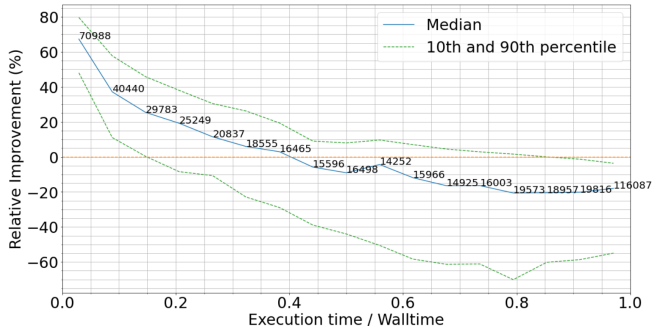
We have showed that response time could be used to understand qualitatively the performance of a solution. Again, one should be careful about unexplained large variance in performance.

In Figure 9, we evaluate the relative improvement of the response time as a function of the number of nodes of the jobs and as a function of the ratio execution time over runtime estimate (a ratio of one means that the users have predicted perfectly the length of their jobs, while a ratio of 0 means that they have done an extremely poor prediction).

Again, this figure shows how one can use the response time as a tool for qualitative analysis: in this case it seems that the relative improvement is not



(a) Evolution of response time for jobs as a function of the Number of nodes (only Theta data)



(b) Evolution of response time for jobs as a function of the ratio $t^{\text{real}}/t^{\text{estimate}}$

Fig. 9: Relative Improvement of EXACT over USER-WALLTIME of the response time (Mira and Theta data), as a function of some job parameters. The numbers on the blue line are the numbers of jobs in each group.

correlated to the number of nodes used by the jobs, except for very large jobs. The variability also decreases with larger jobs. However, unsurprisingly we can see that the worse the prediction of the user, the better the relative improvement.

This qualitative analysis confirms the theoretical limitations that we discussed in Section 2. Response time can be used for qualitative analysis by studying it on specific job parameters. We have also demonstrated how one can use the standard deviation of the utilization to differentiate to solutions that seems to have similar performances.

4 Related work

The question of metrics in HPC Batch schedulers is addressed by Goponenko et al. [13]. They focus on the question of packing efficiency and fairness. They consider the metrics of unweighted average bounded slowdown, unweighted average

response time, weighted average response time, and a last metric weighted by number of requested nodes that increases with the waiting time. Utilization is not considered as a metric but as a global objective for selecting a good metric. They conclude in a poor interest of average bounded slowdown and unweighted average response time in terms of efficiency and fairness.

The choice of one or more metrics is driven by some general abstract objective, as quality of packing or fairness between users. Verma et al. [22] compares 4 metrics designed for packing efficiency including utilization. The other metrics are Hole filling, that count the number of unitary jobs that could have been added in holes of the schedule, workload inflation that increases the size of the workload until the limit of pending jobs is reached, and cluster compaction that reduces the number of nodes until the same limit. These metrics are compared for different criteria including accuracy and time for computation of the metric (the two last metrics imply the computation of multiple schedules).

Some other metrics have been used to measure the packing capacity of an algorithm. The loss of capacity is the name of two different metrics used to evaluate idle time while jobs are waiting. Leung et al. [15] use the loss of capacity to measure the capacity of improvement of the utilization, that is the average minimum between the number of nodes requested by pending jobs and the number of available nodes. Zhang et al. [25], use it to measure the average fraction of idle nodes when there are waiting jobs. Some authors [18, 6] consider the mean response time and bounded slowdown for different categories of jobs based on their duration and number of requested nodes.

5 Conclusion

Evaluating correctly the performance of resource and job management systems is a major question that relies on many dimensions. With the generalization of black-box recommendation systems, being confident in the evaluation is a key research problem.

Tsafrir et al. [21, 20] have discussed how one should generate workloads in order to evaluate correctly the impact of runtime estimates. In the line of their work, we discuss the objectives that one should consider in order to evaluate correctly the impact of runtime estimates on job schedulers.

In this work we were able to show that certain average objectives such as the mean bounded slowdown or mean response time should not be used as they are subject to too much variability depending on the input. We have discussed the limitation of the utilization for a given workload to compare different traces. Finally, we introduced a new optimization metric which can help inform about the quality of a schedule: the standard deviation of the utilization.

We used a specific use-case, runtime predictions, to show experimentally the limits of these objectives.

Acknowledgment

This work was supported in part by the French National Research Agency (ANR) in the frame of DASH (ANR-17-CE25-0004) and in part by the Inria Exploratory project REPAS.

References

1. ALCF Public Data. https://reports.alcf.anl.gov/data/. This data was generated from resources of the Argonne Leadership Computing Facility, which is a DOE Office of Science User Facility supported under Contract DE-AC02-06CH11357, Accessed: 2022-12-08.
2. Top500. https://www.top500.org/.
3. Bailey Lee, C., Schwartzman, Y., Hardy, J., and Snavely, A. Are user runtime estimates inherently inaccurate? In *Workshop on Job Scheduling Strategies for Parallel Processing* (2004), Springer, pp. 253–263.
4. Carastan-Santos, D., and De Camargo, R. Y. Obtaining dynamic scheduling policies with simulation and machine learning. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis* (2017), pp. 1–13.
5. Carastan-Santos, D., De Camargo, R. Y., Trystram, D., and Zrigui, S. One can only gain by replacing easy backfilling: A simple scheduling policies case study. In *2019 19th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGRID)* (2019), IEEE, pp. 1–10.
6. Chiang, S.-H., Arpaci-Dusseau, A., and Vernon, M. K. The impact of more accurate requested runtimes on production job scheduling performance. In *Job Scheduling Strategies for Parallel Processing: 8th International Workshop, JSSPP 2002 Edinburgh, Scotland, UK, July 24, 2002 Revised Papers 8* (2002), Springer, pp. 103–127.
7. Du, Y., Marchal, L., Pallez, G., and Robert, Y. Doing better for jobs that failed: node stealing from a batch scheduler's perspective.
8. Dutot, P.-F., Mercier, M., Poquet, M., and Richard, O. Batsim: a Realistic Language-Independent Resources and Jobs Management Systems Simulator. In *20th Workshop on Job Scheduling Strategies for Parallel Processing* (Chicago, United States, May 2016).
9. Fan, Y., Li, B., Favorite, D., Singh, N., Childers, T., Rich, P., Allcock, W., Papka, M. E., and Lan, Z. Dras: Deep reinforcement learning for cluster scheduling in high performance computing. *IEEE Transactions on Parallel and Distributed Systems 33*, 12 (2022), 4903–4917.
10. Fan, Y., Rich, P., Allcock, W. E., Papka, M. E., and Lan, Z. Trade-off between prediction accuracy and underestimation rate in job runtime estimates. In *2017 IEEE International Conference on Cluster Computing (CLUSTER)* (2017), IEEE, pp. 530–540.
11. Gainaru, A., and Pallez, G. Making speculative scheduling robust to incomplete data. In *2019 IEEE/ACM 10th Workshop on Latest Advances in Scalable Algorithms for Large-Scale Systems (ScalA)* (2019), IEEE, pp. 62–71.
12. Gainaru, A., Pallez, G., Sun, H., and Raghavan, P. Speculative scheduling for stochastic HPC applications. In *Proceedings of the 48th International Conference on Parallel Processing* (2019), pp. 1–10.

13. Goponenko, A. V., Lamar, K., Peterson, C., Allan, B. A., Brandt, J. M., and Dechev, D. Metrics for packing efficiency and fairness of HPC cluster batch job scheduling. In *2022 IEEE 34th International Symposium on Computer Architecture and High Performance Computing (SBAC-PAD)* (2022), pp. 241–252.
14. Legrand, A., Trystram, D., and Zrigui, S. Adapting batch scheduling to workload characteristics: What can we expect from online learning? In *2019 IEEE International Parallel and Distributed Processing Symposium (IPDPS)* (2019), IEEE, pp. 686–695.
15. Leung, V. J., Sabin, G., and Sadayappan, P. Parallel job scheduling policies to improve fairness: A case study. In *2010 39th International Conference on Parallel Processing Workshops* (2010), IEEE, pp. 346–353.
16. Mu'alem, A. W., and Feitelson, D. G. Utilization, predictability, workloads, and user runtime estimates in scheduling the IBM SP2 with backfilling. *IEEE transactions on parallel and distributed systems 12*, 6 (2001), 529–543.
17. Patel, T., Liu, Z., Kettimuthu, R., Rich, P., Allcock, W., and Tiwari, D. Job characteristics on large-scale systems: long-term analysis, quantification, and implications. In *SC20: International conference for high performance computing, networking, storage and analysis* (2020), IEEE, pp. 1–17.
18. Perkovic, D., and Keleher, P. J. Randomization, speculation, and adaptation in batch schedulers. In *SC'00: Proceedings of the 2000 ACM/IEEE Conference on Supercomputing* (2000), IEEE, pp. 7–7.
19. Tang, W., Lan, Z., Desai, N., and Buettner, D. Fault-aware, utility-based job scheduling on blue, gene/p systems. In *2009 IEEE International Conference on Cluster Computing and Workshops* (2009), IEEE, pp. 1–10.
20. Tsafrir, D. Using inaccurate estimates accurately. In *Job Scheduling Strategies for Parallel Processing: 15th International Workshop, JSSPP 2010, Atlanta, GA, USA, April 23, 2010, Revised Selected Papers 15* (2010), Springer, pp. 208–221.
21. Tsafrir, D., Etsion, Y., and Feitelson, D. G. Backfilling using system-generated predictions rather than user runtime estimates. *IEEE Transactions on Parallel and Distributed Systems 18*, 6 (2007), 789–803.
22. Verma, A., Korupolu, M., and Wilkes, J. Evaluating job packing in warehouse-scale computing. In *2014 IEEE International Conference on Cluster Computing (CLUSTER)* (2014), IEEE, pp. 48–56.
23. Zhang, D., Dai, D., He, Y., Bao, F. S., and Xie, B. Rlscheduler: an automated HPC batch job scheduler using reinforcement learning. In *SC20: International Conference for High Performance Computing, Networking, Storage and Analysis* (2020), IEEE, pp. 1–15.
24. Zhang, D., Dai, D., and Xie, B. Schedinspector: A batch job scheduling inspector using reinforcement learning. In *Proceedings of the 31st International Symposium on High-Performance Parallel and Distributed Computing* (New York, NY, USA, 2022), HPDC '22, Association for Computing Machinery, p. 97–109.
25. Zhang, Y., Franke, H., Moreira, J. E., and Sivasubramaniam, A. Improving parallel job scheduling by combining gang scheduling and backfilling techniques. In *Proceedings 14th International Parallel and Distributed Processing Symposium. IPDPS 2000* (2000), IEEE, pp. 133–142.