Qualitatively Analyzing Optimization Objectives in the Design of HPC Resource Manager

Robin Boëzennec, Fanny Dufossé, Guillaume Pallez

Analyzing Qualitatively Optimization Objectives in the Design of HPC Resource Manager

Robin Boëzennec , Fanny Dufossé , Guillaume Pallez

***Abstract*—A correct evaluation of scheduling algorithms and a good understanding of their optimization criterias are key components of resource management in HPC. In this work, we discuss bias and limitations of the most frequent optimization metrics from the literature. We provide elements on how to evaluate performance when studying HPC batch scheduling.**

We experimentally demonstrate these limitations by focusing on two use-cases: a study on the impact of runtime estimates on scheduling performance, and the reproduction of a recent high-impact work that designed an HPC batch scheduler based on a network trained with reinforcement learning. We demonstrate that focusing on quantitative optimization criterion ("our work improve the literature by X%") may hide extremely important caveat, to the point that the results obtained are opposed to the actual goals of the authors.

Key findings show that mean bounded slowdown and mean response time are irrelevant objectives in the context of HPC. Despite some limitations, mean utilization appears to be a good objective. We propose to complement it with its standard deviation in some pathologic cases. Finally, we argue for a larger use of area-weighted response time, that we find to be a very relevant objective.

***Index Terms*—State of the Practice, Methodology, Resource Management, Machine Learning, Metric, Batch Scheduling, High Performance Computing, Runtime estimates**

I. Introduction

With the development of machine learning solutions, resource Management of large scale systems is evolving. We are seeing an increasing number of learning-based algorithms to map applications to resources or to optimize their use. Such techniques often consist of two steps: a learning phase that learns how to optimize a given objective, and an exploitation phase.

Compared to historical *classical* resource management techniques, their main limitation is the lack of transparency of their decision: what have they learned? What criteria are they putting first when taking such and such decision?

As an example, consider the classical job-packing problem: how do you schedule parallel jobs on an homogeneous parallel platform. For this problem, classical scheduling heuristics such as First Come First Served (FCFS), where jobs are sorted by increasing arrival date before being scheduled, or Shortest Area First (SAF), where jobs are scheduled by increasing volume of work before being scheduled, are well understood and have been thoroughly studied. Their simplicity permit a clear comprehension of their behavior. The most well known of the job packing heuristics is EASY-BF that combines the FCFS approach with a backfilling technique to reduce idle time. The backfilling step consists in filling the idle periods of nodes with small waiting jobs that can be allocated without delaying previously scheduled jobs. Backfilling permits a better packing capacity without impacting the fairness.

A second category of scheduling algorithms consist of neural networks trained with reinforcement learning (RL) techniques [11], [15], [27]. RL algorithms perform their explorative learning phase by evaluating the performance of a schedule on a given metric, and updating the different parameters of their networks based on these performances. In the end, the performance for these metrics often work out well, but, the actual behavior of such schedulers is generally opaque. In addition, this behavior highly depend on the metric used for optimization. Thus, the analysis of such algorithms can not be based solely on the targeted metrics. Generally, a scheduling analysis can not only be quantitative and requests to consider qualitative criteria as packing efficiency, fairness or transparency.

Understanding the bias and limitations of metrics for HPC resource management becomes even more important to help explain the behavior of such algorithms.

In this work which extends considerably our preliminary discussion [4] we discuss several methodological elements to qualitatively study optimization criteria and the result of an analysis. This work is illustrated using the job packing problem, but our methodology should be applied to other resource management problems (such as I/O, memory). Our main contributions are the following:

- We give a qualitative analysis of several optimization criteria used in the literature. We argue that the mean bounded slowdown and mean response time are irrelevant objectives in the context of HPC. On the contrary, we show the relevance of the area weighted response time to measure the packing efficiency of HPC scheduling algorithms even in context where the system utilization does not allow to discriminate between algorithms.
- Through two experimental use-cases, we confirm our findings:

    1) 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

Manuscrit received ... ; revised ... . Robin Boëzennec is with Inria, Labri, University of Bordeaux, and University of Rennes. Guillaume Pallez is with Inria and Université de Rennes. Fanny Dufossé is with Univ. Grenoble Alpes, Inria, CNRS, Grenoble INP, LIG.

and by simply looking at specific performances, one cannot conclude a study.

2) We strengthen our points with a qualitative analysis of the performances of RLScheduler [27]. RLScheduler is a neural-network batch scheduler for HPC, trained with reinforcement learning. This use-case also serves us to discuss the importance of some good practices in performances evaluation. We show that compared to the state of the art, RLScheduler is not up to the task, yet.

The rest of the paper is constructed as follows: In Section II, 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 III-A details the methodology of our first use-case before presenting and analyzing the results in Section III-B. Section IV analyzes the behavior of RLScheduler and uses it to discuss several metrics. Section V discusses related work. Finally, Section VI concludes the work.

II. 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* [20]. 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.

To define objectives, we use the following notations for job J_i (represented visually in Figure 1):

t_i^{sub}	The release time of job J_i (aka submission time)
t_i^{start}	The starting time of job J_i
t_i^{fin}	The completion time of job J_i
t_i^{run}	The length of job J_i (aka execution time/runtime)[1]
t_i^{wait}	The waiting time of job J_i ($t_i^{\text{wait}} = t_i^{\text{start}} - t_i^{\text{sub}}$)
N_i^{cores}	The number of cores used by job J_i

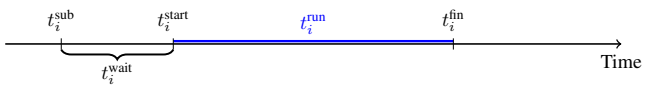
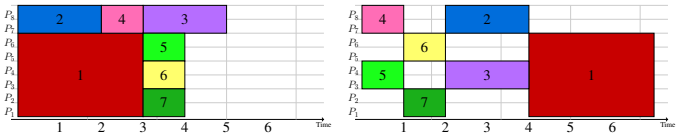


Fig. 1: A visual representation of the various notations

[1]This is different from the requested/estimated time t_i^{estimate} which we discuss in Section III.



(a) Mean Response time ≈ 3.6; Mean Slowdown ≈ 2.8; WRT $= 3.3$

(b) Mean Response time $= 3$; Mean Slowdown ≈ 1.8; WRT $= 5$

Fig. 2: In this example, all jobs are released at $t = 0$. Despite what appears to be a more efficient strategy, the left schedule has worse mean response time and slowdown than the right schedule.

A. Mean (bounded) slowdown

The average bounded slowdown (also called mean flow) is an optimization criteria extensively used in the literature [6], [7], [17], [25], [27], [28]. 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

$$S_i = \frac{t_i^{\text{run}} + t_i^{\text{wait}}}{t_i^{\text{run}}} = \frac{t_i^{\text{fin}} - t_i^{\text{sub}}}{t_i^{\text{run}}}$$

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*:

$$S_i^b = \max\left(\frac{t_i^{\text{fin}} - t_i^{\text{sub}}}{\max\left(t_i^{\text{run}}, \tau\right)}, 1\right) \quad (1)$$

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

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

1) 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 studies this objective, and is the opposite of what a system administrator of an HPC machine is looking for. This is illustrated in Figure 2.

Work by Carastan-Santos et al. [6] 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. [17] 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 substantial performance gains for this objective.

In Section III-B1, we also show that this is subject to a high variability, and very influenced by the behavior of small jobs representing an insignificant part of the workload. It makes it an unfit metric to evaluate the performance of RJMS in HPC.

2) Alternative approach: To understand the actual behavior of the system, Du et al. [9] 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 a more 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.

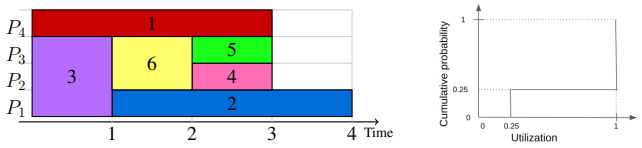
B. Utilization

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 [12]–[14].

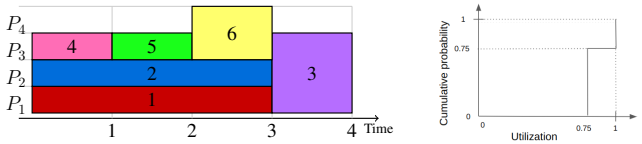
If $W(t_1, t_2)$ is the total amount of work done between t_1 and t_2 on a platform with N nodes, the utilization $U(t_1, t_2)$ on the interval $[t_1, t_2]$ is measured as:

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

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 [9].



(a) Schedule example and its corresponding cumulative distribution function



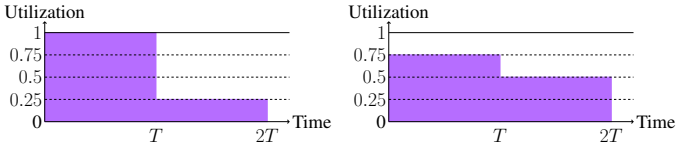
(b) Schedule example and its corresponding cumulative distribution function

Fig. 3: Even though the global utilization is the same between the two schedules (13/16), the distribution of their utilization differ significantly.

1) 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 discriminating between different schedule qualities (Figure 3).

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).

2) 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.



(a) Mean utilization: 0.625
Utilization std: 0.5625

(b) Mean utilization: 0.625
Utilization std: 0.125

Fig. 4: Different scheduling configurations with their mean utilization and utilization std

There are settings for which 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 3). 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. Hence, an alternative approach is to look at the standard deviation of the utilization instead of its mean. Remember that this only works when the system is under-utilized.

When a system is under-utilized, two schedules have an almost identical utilization, so we propose to measure the standard deviation of the utilization as a way to differentiate the quality of a schedule: the "best" algorithm from a utilization perspective should have a higher standard deviation (more time-windows with very high occupation and more time-windows with low occupation). The idea behind is that when there are bursts of incoming workload, the better the scheduler, the sooner the workload is scheduled (hence with peak of utilization vs a more balanced utilization). This is shown graphically in Figure 4 where the first schedule is better at using all available resources at the same time, leading to a standard deviation of utilization greater than the second schedule.

Some remarks on using the standard deviation:

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 thus utilization can not discriminate algorithms), to give information about which algorithm would be able to reach the higher utilization once the system actually becomes fully utilized.
2) Thus, it is important to note that the standard deviation 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.
3) 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 standard deviation x times greater than another one means over-

all? In the Appendix [5, Section 1] we give an answer for a particular example, but we have no general answer.

C. Response time (and Wait Time)

Mean response time (or mean wait time) is a metric often used in the literature [12], [14], [19], [23], [25], [27]. 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.

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

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.

1) 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 2 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. 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).

Some authors also study the maximum response time, as a mean to qualify the performance that a user may expect. However, this metric does not differentiate between a 1-hour job that waits for 1 minute, and a 1-minute job that waits for 1 hour.

Finally, similarly to the mean bounded slowdown, we show in Section III-B1 that depending on the workload this objective is subject to a lot of variability.

2) Alternative metric: Goponenko et al. [16] have argued for the use of the AWF, where one weights the response time by a priority proportional to the quantity of work (cores$\cdot$time) of each job. In the following we call this metric *WRT* for area-Weighted Response Time.

$$\text{WRT} = \sum_i W_i \cdot \mathcal{RT}_i \tag{3}$$

This metric is interesting for the following properties:

Proposition 1. *Given a schedule:*

1) *Performing work earlier improves the WRT metric;*
2) *Permuting any amount of work without changing the utilization profile*[2] *of the schedule keeps the WRT metric unchanged;*

As long as the rigid job model is preserved (i.e. each job keep the same runtime and number of cores).

[2]The utilization profile of a schedule is the function $t \mapsto u(t)$ where $u(t)$ is the instantaneous utilization at time t. Then $U(t_1, t_2) = \frac{\int_{t_1}^{t_2} u(t)dt}{t_2 - t_1}$

While the second result was mentioned [16], we did not find a formal proof of this result in the literature and have provided it in the Supplemental Material [5, Section 2].

Interestingly, Proposition 1 highlights the fact that WRT is a metric that measures the quality of the Utilization Profile, and hence evaluates the packing efficiency of algorithms. Compared to the utilization, WRT is able to give information on the mean response time. It also keeps its relevance at low utilization, even when the workload submission profile varies. Indeed, it is able to compare two schedules with similar average utilization if one schedule performs work earlier than the other schedule.

WRT can therefore also be seen as an interesting alternative to the utilization metric as well as an alternative metric to Mean Response Time.

D. Additional comments

We have presented several limits that one faces when considering quantitatively optimization metrics. There are other important considerations that one should consider.

1) Performance Gain: As a community we often value large gains over previous algorithms. However, let us show with a simple example how these gains can be deceptive. Define the opposite of the utilization U as idle occupation $I = 1 - U$ which is an objective that one wants to minimize. Given an algorithm with an utilization of $U_1 = 95\%$ (this corresponds to current HPC utilization [20]). If another algorithm improves this utilization by 1%, this corresponds to an improvement of the idle time of 20% . So is a 1% gain a good performance or not?

2) Measurement: How to measure correctly the performance of a solution is also a complicated issue because of non-steady state phases where behavior can be different. We discuss this in more depth in Sections III-A4 and IV-B3.

3) Summary: As presented many objectives when optimized have negative side effect for the scheduling of large jobs on large scale platforms.

Yet many works, particularly recent works 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 [27], [28]. 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. We demonstrate the limits of these analysis in Section IV by correctly analyzing the results.

In the next Sections, we show on specific use-cases what happens when one does not stop at the quantitative performance, but goes in depth of using a metric.

III. 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], [20] 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 estimates functions: a perfect estimate, and the estimate provided by the users (see Section III-A3). The results on metrics detailed in Section II are detailed in Section III-B.

A. Evaluation Methodology

We simulate the execution of EASY-BF using the batch simulator Batsim on the workloads of platforms Mira and Theta. Our code is available at https://gitlab.inria.fr/rboezenn/hpc_metrics_code.

1) Batsim: Simulations are run in Batsim [10] (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.

2) 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 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 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 [20].

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 submission times:

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

3) 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 : $r \mapsto r + 1$ 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.

4) 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.

a) 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, in consistence with the literature [25]. The measurement window $[t_1, t_2]$ is defined s.t.:

$$\begin{aligned} t_1 =& 0.15 \left(\max_i t_i^{\text{sub}} - \min_i t_i^{\text{sub}} \right) \\ t_2 =& 0.85 \left(\max_i t_i^{\text{sub}} - \min_i t_i^{\text{sub}} \right). \end{aligned}$$

b) 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 [25].

c) 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 sometimes abbreviate as *Relative Improvement* or RI). This relative improvement $\text{RI}(O)$ is measured as:

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

$$\text{RI}(O) = \frac{O_{\text{EXACT}} - O_{\text{USER-WALLTIME}}}{O_{\text{USER-WALLTIME}}} \tag{4}$$

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

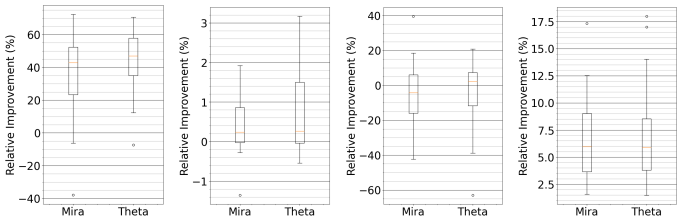
$$\text{RI}(O) = \frac{O_{\text{USER-WALLTIME}} - O_{\text{EXACT}}}{O_{\text{USER-WALLTIME}}} \tag{5}$$

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

B. Result analysis

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

The two runtime estimate functions are evaluated in Figure 5 over the various criteria discussed in Section II: Bounded Slowdown (Section II-A), Utilization (Section II-B), Response Time and WRT (Section II-C). In these figures, to discuss the performance difference between the two runtime estimate



(a) Mean Bounded Slowdown (b) Utilization (c) Mean Response Time (d) WRT

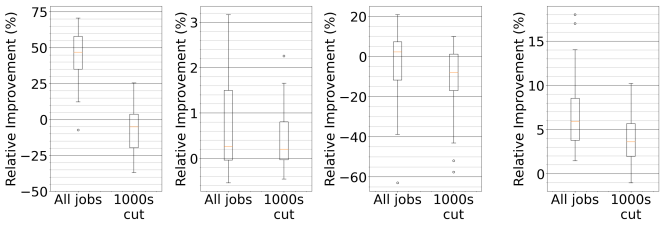
Fig. 5: Relative improvement of EXACT over USER-WALLTIME for different metrics. The subfigures have different scales. Utilization is a maximization objective while the others are minimization objectives.

functions, we use the relative improvement of EXACT over USER-WALLTIME as computed by Equations (4) and (5).

The quantitative results in Figure 5 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 Resources 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. 4% decrease of performance for Mira), but it does not hold if we measure the area-weighted response time.

Looking at mean values is not enough. In the next section, we analyze in depth these results.

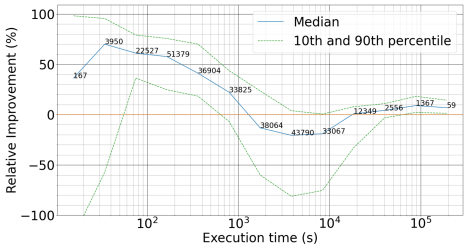
1) Mean-Bounded Slowdown and Response Time: An observation of Figure 5a (on bounded slowdown metric) and 5c (on mean response time) is the important variability of the performance. Discussing the mean of an objective with high variability is meaningless: the performance is highly influenced by the workload



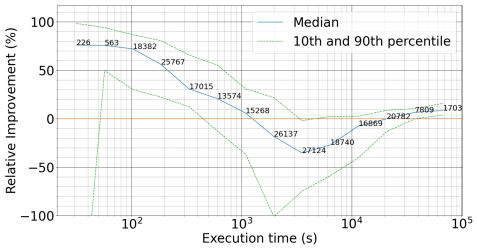
(a) Average bounded slowdown (b) Utilization (c) Mean response time (d) WRT

Fig. 6: Evolution of the Relative Improvement of EXACT over USER-WALLTIME with several metrics when deleting the jobs that last less than 1000s (Theta data). y-axis are not the same.

Another way to confirm this is to slightly change the workload and see the impact on the objective. In Figure 6, we compare the performance of the two algorithms on the



(a) Theta data



(b) Mira data

Fig. 7: Median Relative Improvement of the RT for Theta (up) and Mira (down) jobs as a function of t^{run}. The numbers on the blue line are the numbers of jobs in each group.

workload that contains all jobs, and on the workload where we have removed all jobs that last less than 1000 seconds. *Interestingly, after this transformation, only 50% of the jobs remain, while 99% of the work remains.*

After this transformation, for both objectives, the algorithm that seemed to perform better now performs worse! It is not the case for the two other metrics (Utilization and WRT). In addition, one can remark that these objectives still have a very high variability which means again that by selecting the right input workloads, the results could be completely different.

These results show that mean bounded slowdown and mean response time should not be used as quantitative optimization criteria to evaluate the performance of a scheduler. Their high sensitivity to small jobs and their high variability make then unreliable objectives.

Qualitative analysis: These two metrics are nevertheless useful to a qualitative analysis. The following analysis is based on the mean response time with awareness of its bias.

On Figure 7, we plot the relative improvement of response time as a function of job execution time.

Details on Figure generation: To group jobs, we divided the interval $[\min_i t_i^{\text{run}}, \max_i t_i^{\text{run}}]$ in 18 groups of same size on a geometric 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. The values printed on the blue line are the numbers of jobs in each group. The green dotted lines correspond to the first and last decile.

Figure 7 shows a correlation between the job execution time and its response time improvement by using EXACT. Specifically we observe three groups: short jobs, medium-

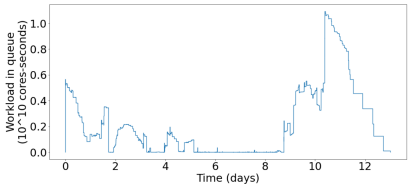


Fig. 8: Available work in the waiting queue as a function of time for one of workload sample from Mira when scheduled with the EXACT walltime function. This shows that the availability of jobs is not regular throughout the execution.

length jobs and long jobs. When switching from USER-WALLTIME to EXACT:

- EXACT improves considerably the response time of small jobs;
- Meanwhile, medium-length jobs see an increased average response time;
- Finally, longer jobs benefit from a little improvement of their response time.

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

A more qualitative analysis like this one may help understand the behavior of an algorithm over another one. Similarly, one could study the impact of other parameters (for instance the number of nodes, or the ratio $t_i^{\text{run}}/t_i^{\text{estimate}}$).

Particularly, in this case one could hypothesize that when using better runtime estimates medium-length jobs are less backfilled than they were when the runtime estimate of longer jobs is really wrong. Long jobs (which are rarely backfilled) benefit from fewer medium-lengthed jobs taking priority over them.

Slightly out of scope of this study (but for the sake of arguments), one can verify this intuition by measuring the number of jobs that are indeed backfilled (see Table I, to read it: 82% of the 142k jobs that are smaller than 1000s are backfilled with USER-WALLTIME).

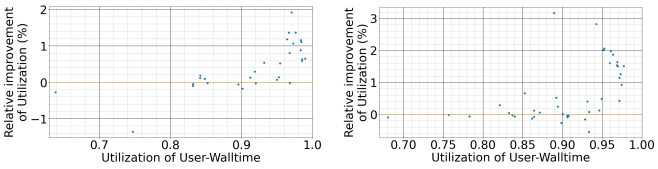
	All	$t_i^{\text{run}} < 1000s$	$1000s < t_i^{\text{run}} < 20ks$	$20ks < Rt$
Backfilled with USER-WALLTIME	209k (75%)	116k (82%)	90.4k (71%)	2.84k (26%)
Backfilled with EXACT	199k (71%)	118k (83%)	80k (63%)	1.48k (14%)
Total Number of jobs	280k	142k	127k	10.8k

TABLE I: Number of Jobs backfilled with USER-WALLTIME and EXACT in function of their runtime (Theta data). We removed the first and last 15% jobs of each sample as they are not used to compute response time.

To conclude, we can make the following recommendations on mean response time and mean bounded slowdown:

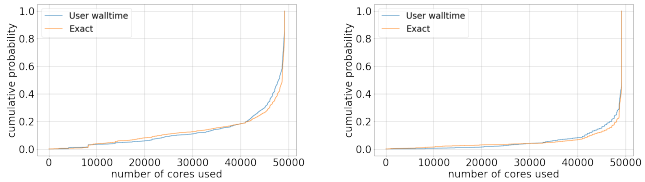
1) In general, they should not be used to evaluate quantitatively a solution;
2) They can help understand *qualitatively* the performance of a solution. Again, one should be careful about unexplained large variance in performance.

2) Utilization:



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

Fig. 9: Relative improvement of the Utilization of EXACT over USER-WALLTIME as a function of the Utilization.



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

Fig. 10: Cumulative distribution functions of the utilization for two selected scenarios from Mira where utilization difference between EXACT and USER-WALLTIME is close to 0.

a) Qualitative analysis: With respect to the utilization 5b, it seems that the improvement is extremely small (about 0.5%). As we explained in Section II-B, this may not be surprising and is an artifact of the non-constant arrival rate (see Figure 8). 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 9 the relative improvement of the utilization as a function of the Utilization of USER-WALLTIME.

The measure presented in Figure 9 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 runtime estimates). This shows the limits of the utilization as an objective to compare two solutions. Indeed, it is only a relevant objective when higher than a certain threshold.

In Figure 10, we show the density distribution of EXACT and USER-WALLTIME for two workloads where the relative difference in utilization is almost null. We observe 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 ($> 95\%$).

b) Toward better objectives: As a quantitative objective to be able to compare various algorithms, we propose to measure the standard deviation (std) of utilization. For two

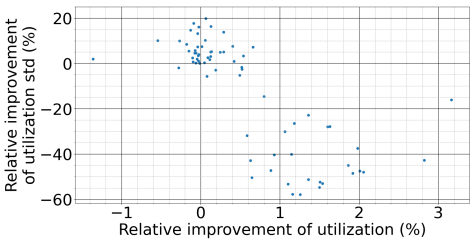


Fig. 11: Relative improvement of the standard deviation of the utilization as a function of the relative improvement of the utilization (Mira and Theta data).

algorithms 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.

In Figure 11, we plotted the increase in standard deviation as a function of the increase in utilization. Two clusters are visible:

- A cluster with a relatively high increase in utilization, but a drop in standard deviation.
- Another with a relative improvement of utilization close to 0, but an increase in standard deviation

The first cluster corresponds to the data points of the figure 9 which have a clear improvement in utilization (note that the y-axis of 9 is the x-axis of 11), while the second corresponds to the data points where there is no clear increase in utilization.

In the first case, as the utilization is already better with EXACT than with USER-WALLTIME, one can conclude that EXACT is better without having to look at the standard deviation. However, for the second cluster, it is not possible to conclude that an algorithm is better than another just by looking at the utilization: indeed, the utilization is quite close to 0. So, one needs to look at the standard deviation (which is up to 20% better with EXACT) to conclude that EXACT is better than USER-WALLTIME.

Our conclusion is two-fold:

1) First, our experiments confirm that when a platform is under-utilized, the utilization can be an irrelevant objective.
2) In this particular case, the standard deviation allows to compare various algorithms in order to determine which one would be able to reach the highest utilization once the platform becomes fully-utilized.

IV. USE-CASE: REINFORCEMENT-LEARNING FOR RESOURCE MANAGEMENT

In the previous Section, we have claimed that using the mean bounded slowdown was a problem when used to evaluate quantitatively a solution. We have claimed in Section II-D that this was particularly a problem for what we called *black-box* algorithms, i.e. scheduling algorithm that take decisions that are not explainable.

Explainability of algorithms When scheduling jobs using the First-Come-First-Served strategy, one can explain the decisions taken by the schedule (the oldest job gets priority). Similarly, in the F_1 algorithm provided by Carastan-Santos et al. [6], even if one does not have the details on how the priority function is obtained, the scheduling strategy is interpretable (a mix of size of the job and release time of the job). We call these algorithms *explainable*: the system administrator can explain the algorithm to the users. By opposition, a recent line of work such as the work by Zhang et al. [27] propose to train solutions via various learning strategies. The scheduler then takes what it believes to be the best solution. In this case one cannot explain what made the scheduler take a decision over another one. This is one we call a *black-box algorithm.*

In this Section, we demonstrate our claim by reproducing a recent result by Zhang et al. [27] and by providing a different analysis of the performance. We selected this work for several reasons:

- It is one of the first work that provides a RL-based solution for resource management, was published at SC'20 (a very visible conference in HPC) and has already been cited more than 50 times which shows an engagement by the community.
- It claims that *"the learned model perform stably, even when applied to unseen workloads, making them practical for production use."*
- The main benefits observable from their solution is when applied to the Mean Bounded Slowdown objective.

A. Methodological framework

For this section, we have used the code made available by the authors at https://github.com/DIR-LAB/deep-batch-scheduler. We used the last commit available (cd433e3) pushed in May 2021. In addition to the RLScheduler code, the authors provide their input traces, their trained models and other baseline schedulers. This is what we used for the analysis of this section. The code of our analysis is available at https://gitlab.inria.fr/rboezenn/hpc_metrics_code.

1) Scheduling algorithms: RLScheduler has one network model with several versions depending on the training: the authors trained a separated version for each pair of trace (SDSC-SP2, HPC2N, and Lublin-1 and Lublin-2) and optimization metric (mean bounded slowdown, mean response time and utilization).

In our analysis, we focused on the trace Lublin-1 (called Lublin256 in the code) because it is the one with the highest utilization, and hence where the importance of the scheduler is likely to be the most notable (note that the trace is synthetic).

Let RLSCHED-MBSD-LUB1 be the model trained with mean bounded slowdown on Lublin-1 traces, and RLSCHED-UTIL-LUB1 the model trained with utilization on Lublin-1 traces. We studied the algorithms RLSCHED-MBSD-LUB1 and RLSCHED-UTIL-LUB1 with backfilling as well as the algorithm FCFS-BF (first-come-first-served with backfilling) provided in the github repository.

2) Traces: The traces included have a total of 10 000 jobs. For their evaluations, Zhang et al. [27] performed ten independent tests, scheduling 1024 randomly sampled consecutive jobs of the trace. The seed for their tests were available. As a sanity check, we verified that we could reproduce the main

results they obtained on the Lublin-1 trace (specifically those presented in Table II).

B. Analysis of results

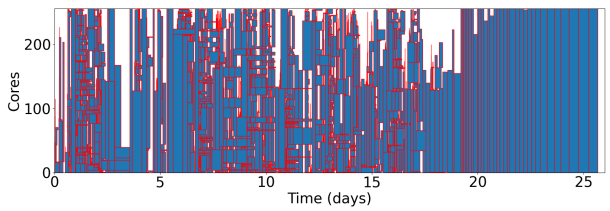
When compared to FCFS-BF, Zhang et al. [27] reported the following performance of the RLScheduler mechanism with backfilling:

- It strongly and consistently improves the Mean Bounded Slowdown [27, Table V, Table VII].
- It sometimes slightly improves utilization, and sometimes slightly hurts it [27, Table VI].
- It strongly improves the Maximum Bounded Slowdown [27, Table VIII].

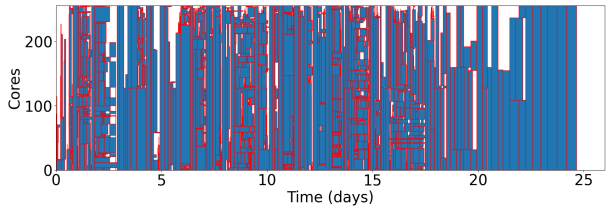
For completeness, these results are reported in Table II.

In what follows, we re-discuss these results with a qualitative analysis and show results opposite to what the authors observe. We conclude on the importance of a good methodological framework.

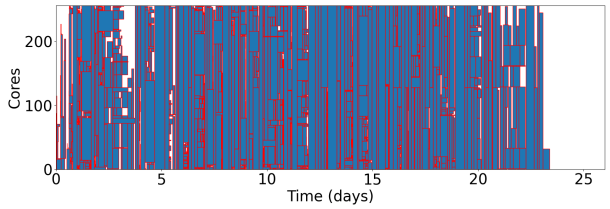
1) Visual representation: Before discussing qualitatively all results, we present in Figure 12 the Gantt chart of three schedules (with RL model and without). All these schedules are computed on the same job data set.



(a) RLSCHED-MBSD-LUB1



(b) RLSCHED-UTIL-LUB1



(c) FCFS-BF

Fig. 12: Gantt chart for various scheduling strategy on the same workload (Lublin-1, start at index 5000, 2048 jobs). The jobs are in blue and the red lines represent the edges.

Note that for this data set, if the first job is released at time 0, then the last job is released at 10pm on day 17.

Already, one can clearly see on these examples a behavior of RLScheduler that schedules many *large* jobs at the end of the schedule. This is extremely concerning as it intuits starvation. In addition, an observation is that the utilization seems extremely unbalanced throughout the execution. All this is of course circumstantial, but coincides with the various observations that we made in the previous Section of this work. In the rest of this Section we demonstrate that this is actually a trend.

2) Mean Bounded Slowdown: We have showed in Sections II-A and III-B1 the limits of studying the mean bounded slowdown: in terms of input dependency and high variability, and with the fact that having an important improvement on the mean bounded slowdown may mean unbalancing the workload and having many small jobs executed first. This seemed particularly noticeable on Figure 12.

We confirm these various results here by studying the wait-time as a function of the work of each job ($W_i = N_i^{\text{cores}} t_i^{\text{run}}$) in Figure 13, as well as recomputing the mean bounded slowdown when the smallest jobs are removed from the trace.

Methodology To generate Figure 13, we used the Lublin-1 dataset. We run the entirety of the 10 000 jobs for each of the scheduling algorithm. We then divided the interval $[\min_i W_i, \max_i W_i]$ in 9 groups of same size. Then, we box-plotted the wait time of the jobs of each group.

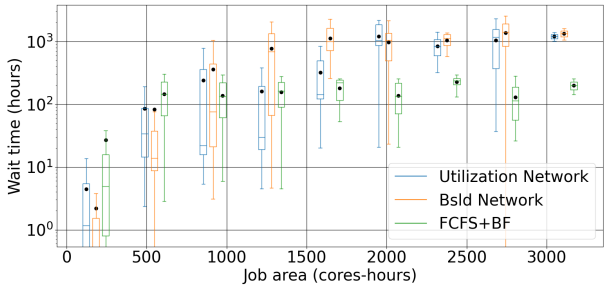


Fig. 13: Wait time as a function a job work for several schedulers.

The data observed on Figure 13 confirms the fact that what the RL model does is to actually schedule small jobs as soon as possible while delaying large jobs. This is particularly true for the network trained on the bounded slowdown objective.

Finally, Table III, confirms our previous results in therm of dependency of the mean bounded slowdown metric toward small jobs. It compares the mean bounded slowdown results obtained on the entirety of the Lublin-1 traces, and on the same trace once we removed jobs with an execution time lower than 130 seconds. The remaining jobs represent 50.6% of the jobs of the trace but 99.7% of the quantity of work. The network was not re-trained on the new trace with deleted jobs.

This section confirms again the fact that the mean bounded slowdown should not be used as an optimization objective.

3) Utilization: On the utilization performance, the first point to be made is that all scenarios have quite low utilization: the highest utilization observed by the authors is Lublin-1 with 87% (see Table II). This contradicts many recent results about utilization in HPC centers (for instance on Mira the average utilization is 95% [20]).

This could be explained if the trace had a low submission rate, However, in this case, on average the system has a load

Trace	Mean BSD		Utilization		Max BSD	
	FCFS-BF	RLSCHED-MBSD-#	FCFS-BF	RLSCHED-UTIL-#	FCFS-BF	RLSCHED-MBSD-#
Lublin-1	235.82	58.64	0.868	0.850	-	-
SDSC-SP2	1595.1	397.82	0.682	0.707	7257	4116
HPC2N	127.38	86.14	0.639	0.642	2058	1147
Lublin-2	247.61	118.79	0.587	0.593	-	-

TABLE II: Various performance difference between FCFS-BF, and RLSched (trained on the corresponding trace) [27].

	FCFS+BF	RL (Mbsd)	Ratio of performances
Lublin-1	461	54	8.5
Lublin-1 with cut	34	11	3.1

TABLE III: Evolution of the mean bounded slowdown metric when deleting short jobs.

much higher that what it can deal with. This is visible on Figure 12 where we do not consider any jobs submitted after day 18, but where the execution of the trace lasts until after day 25. Indeed, the mean arrival rate of Lublin-1 is 272 cores-seconds per second. As the platform has 256 cores, it would therefore need extra cores to process all the jobs in time, even with a utilization of 100%!

The explanation comes from an important methodological error that was made in the evaluation of the utilization. When measuring utilization, Zhang et al. [27] measure the utilization of the whole execution of their small traces. In practice, the beginning of the trace (which we can call "initialization phase") and the end of the execution trace (which we can call "clean-up phase") should not be used to measure the utilization since their behavior would certainly change if the trace increased in time.

This is particularly true for the RL algorithm where it seems that the large jobs are delayed with each new small job, and where, in practice they could never be executed.

In Figure 14, we show the difference of utilization on a single sample, as a function of its size (the number of jobs in the sample), using two measurement strategies, one that does not use measurement bounds, and one that measures utilization on the interval $[0.15\left(\max_i t_i^{\text{sub}} - \min_i t_i^{\text{sub}}\right); 0.85\left(\max_i t_i^{\text{sub}} - \min_i t_i^{\text{sub}}\right)]$ as proposed in Section III-A4.

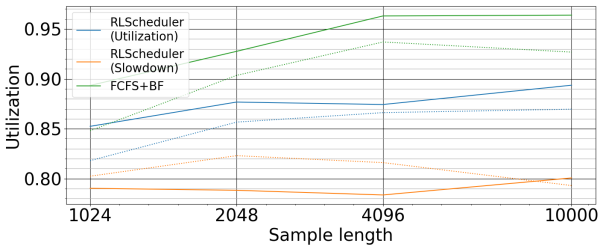


Fig. 14: The dotted lines give the utilization when using no measurement bound. The samples of length 1024, 2048 and 4096 start at index 5000, and the sample of size 10 000 starts at index 0.

From Figure 14 we can make the following observations:

- When considering the full trace, the utilization of FCFS is 96%. This is more coherent with what we know from HPC centers [20] and far from the 87% claimed by Zhang et al.
- The RLScheduler models have extremely poor utilization performance, with a degradation up to 17% when considering the full trace for RLSCHED-MBSD-LUB1. This result contradicts significantly the statements made by the authors.

The lesson learned is that for utilization (i) it is important to have a trace long enough and not a series of small traces; (ii) one should be careful about initialization and clean-up phases. In addition to a correct choice of evaluation criteria, one should be extremely careful about the methodological evaluation. Indeed, the methodology provided by Zhang et al. [27] seemed to imply that the difference in utilization was extremely low, while it is actually quite important when we consider a much longer trace.

4) Max Bounded Slowdown: Given the observations from the two previous sections, one may wonder why the authors do not observe starvation and how they can claim an improvement in Max Bounded Slowdown (see Table II). Note that the authors did not measure it for Lublin-1, but experimental evaluation confirm their quantitative observations (see Figure 15).

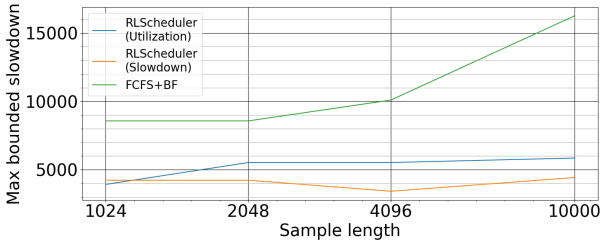


Fig. 15: Max Bounded Slowdown of FCFS+Backfilling, and RLScheduler+Backfilling when τ = 10 seconds.

The first element that should be noted and that should serve as a warning is the value of this bounded slowdown: 15 000 for FCFS+BF. It essentially says that it can happen for a job to wait for 15k times its size. If this was a one-hour job, then it would mean 250 days. This semi-qualitative analysis tells us that this number does not make sense.

It however makes sense if we consider the smallest jobs and the bound $\tau = 10$ seconds used for the computing the bounded slowdown (Eq. (1)). In this case, a bounded slowdown of 15 000 corresponds to a wait time of 42h which corresponds to the fact that the system is over-utilized and that the more we move forward in time, the larger the queue. We can verify this by increasing the size of τ (Figure 16), in this case the max bounded slowdown loses several orders of magnitude, and the max bounded slowdown of RLSCHED-MBSD-LUB1 becomes 2 times worse than that of FCFS+BF!

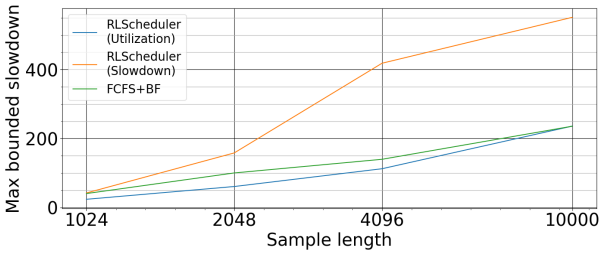
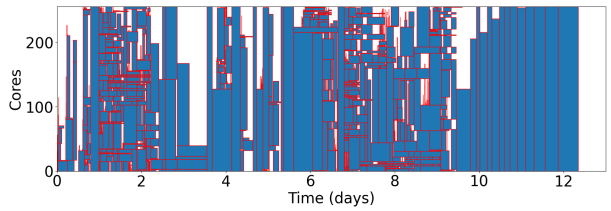


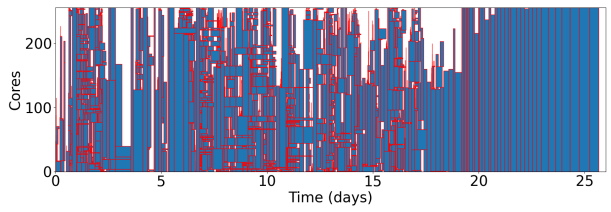
Fig. 16: Max Bounded Slowdown of FCFS+Backfilling, and RLScheduler+Backfilling when $\tau = 1$ hour.

5) Final comments: Throughout this section, we were able to demonstrate again the limits of the objective functions we discussed before. In addition, we showed the importance on how to measure them correctly (e.g. utilization and max bounded slowdown). Interestingly, a simple critical look at the results should have been enough to notice the methodological errors (utilization of 87% is inconsistent with what we know, same for a max bounded slowdown of 15 000).

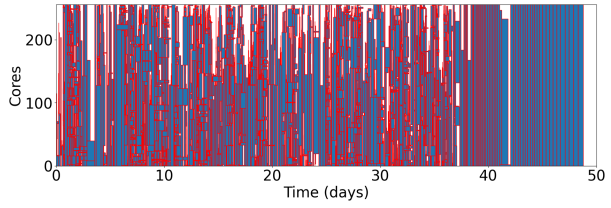
Again, this strengthens our point that simple quantitative analysis are extremely deceptive and are not sufficient. In this case, it hides the fact that the Reinforcement Learning Strategy delays large job indefinitely. This is less visible in the paper because the authors only consider sets of 1024 jobs, but becomes more apparent if we increase the number of jobs (see Figure 17).



(a) 1024 jobs



(b) 2048 jobs



(c) 4096 jobs

Fig. 17: Gantt chart obtained with RLScheduler-MBSD depending on the number of jobs used (start at index 5000).

Correctly studying optimization metrics is of tremendous importance to our field, particularly when studying *black-box* algorithms (such as schedules that are computed by Reinforcement Learning algorithms).

V. Related work

The question of metrics in HPC Batch schedulers is addressed by Goponenko et al. [16]. They focus on the question of packing efficiency and fairness. They consider the metrics of mean bounded slowdown, mean response time, WRT, 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 mean bounded slowdown and mean 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. [26] compare four metrics designed for packing efficiency including utilization. The other metrics are Hole filling, that counts 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. [18] 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. [29], use it to measure the average fraction of idle nodes when there are waiting jobs. Some authors [8], [21] consider the mean response time and bounded slowdown for different categories of jobs based on their duration and number of requested nodes.

Inioluwa Deborah Raji et al. [22] criticize the fact that AI solutions are often deployed while not working. They claim that their functionality is often overlooked and should not be taken as granted.

VI. 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. [24], [25] 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. We also provide elements on a correct evaluation framework.

Specifically we showed the following results:

- We underlined the importance of some critical but often overlooked practices in performance measurement. It

includes using measurement bounds, looking qualitatively at the results... The most important of which being that one should observe the variability of one's result and should not use an objective if the variability is too high. These aspects become even more critical when dealing with black box algorithms.

- Certain average objectives such as the mean bounded slowdown or mean response time should not be used as optimization criteria in HPC. They heavily favor small jobs which is irrelevant for the domain. In addition, this also leads to a very strong reliance on small jobs which causes them to be subject to too much variability depending on the input.
- On the contrary, we believe as others before us that the area-Weighted Response Time (WRT) may be a more robust objective for the analysis of HPC Resource Management solutions. It works as both an administrator and a user metric. In addition, it does not share the flaws of the other user-centric metrics we have discussed, and contrary to utilization it stays a relevant metric when the platform is partially under utilized.
- We have discussed the limitation of the utilization for a given workload to compare different algorithms, and underline the importance of the methodological framework to study it. In the case where the utilization does not allow differentiating between various algorithms, we introduced a new optimization metric which can help inform about the quality of a schedule: the standard deviation of the utilization.

A side result of our analysis is that Easy-Backfilling is actually extremely efficient for HPC machines, even when walltime estimates are bad, and that we should be extremely wary of work that claim high gains over this algorithm.

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] Boëzennec, R., Dufossé, F., and Pallez, G. Optimization metrics for the evaluation of batch schedulers in hpc. In *JSSPP 2023-26th edition of the workshop on Job Scheduling Strategies for Parallel Processing* (2023), pp. 1–19.

[5] Boëzennec, R., Dufossé, F., and Pallez, G. Supplemental material for "qualitative analysis of hpc optimization objectives".

[6] 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.

[7] 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.

[8] 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.

[9] Du, Y., Marchal, L., Pallez, G., and Robert, Y. Doing better for jobs that failed: node stealing from a batch scheduler's perspective.

[10] 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).

[11] 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.

[12] 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.

[13] 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.

[14] 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.

[15] Gaussier, E., Lelong, J., Reis, V., and Trystram, D. Online tuning of easy-backfilling using queue reordering policies. *IEEE Transactions on Parallel and Distributed Systems 29*, 10 (2018), 2304–2316.

[16] 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.

[17] 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.

[18] 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.

[19] 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.

[20] 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.

[21] 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.

[22] Raji, I. D., Kumar, I. E., Horowitz, A., and Selbst, A. The fallacy of ai functionality. In *2022 ACM Conference on Fairness, Accountability, and Transparency* (2022), pp. 959–972.

[23] 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.

[24] 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.

[25] 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.

[26] 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.

[27] 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.

[28] 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.

[29] 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.

BIOGRAPHIES

Robin Boezennec is a PhD Student at INRIA Bordeaux – Sud-Ouest. He received master degree in applied in mathematics and computer science from École des Ponts ParisTech, with a double degree at ENS Paris-Saclay. His main research topic is scheduling, with a strong emphasis on theory.

Fanny Dufossé is tenured researcher at Inria Grenoble, France since 2017. Her research interests include scheduling, green computing and combinatorial scientific computing. She has been in the program committee in different international conferences including SC 2023, Cluster 2023 and Europar 2021.

Guillaume Pallez is a tenured researcher at Inria Rennes. His research interests include algorithm design and scheduling techniques for parallel and distributed platforms (data-aware scheduling, stochastic scheduling etc). Among other, he served as the Technical Program vice-chair for SC'17, Technical program chair for SC'24, and cogeneral chair for ICPP'22. He was a recipient of the 2019 IEEE TCHPC Early Career researcher award. See http://people.bordeaux.inria.fr/gaupy/ for further information.