Qualitatively Analyzing Optimization Objectives in the Design of HPC Resource Manager

ROBIN BOËZENNEC, INRIA, Rennes, France, Université of Rennes, Rennes, France, CNRS, Rennes, France, and IRISA, IRISA, France
FANNY DUFOSSÉ, Université Grenoble Alpes, Grenoble, France, INRIA, Grenoble, France, CNRS, Grenoble, France, and Grenoble INP UGA, Grenoble, France
GUILLAUME PALLEZ, INRIA, Rennes, France

A correct evaluation of scheduling algorithms and a good understanding of their optimization criteria 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 improves 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 hazardous for a purely quantitative analysis in the context of HPC. Despite some limitations, utilization appears to be a good objective. We propose to complement it with the standard deviation of the throughput in some pathological cases. Finally, we argue for a larger use of area-weighted response time, that we find to be a very relevant objective.

CCS Concepts: • **Computing methodologies** → **Simulation evaluation**; **Model development and analysis;**

Additional Key Words and Phrases: State of the practice, high performance computing, runtime estimates, mean bounded slowdown, response time, utilization

ACM Reference Format:
Robin Boëzennec, Fanny Dufossé, and Guillaume Pallez. 2024. Qualitatively Analyzing Optimization Objectives in the Design of HPC Resource Manager. *ACM Trans. Model. Perform. Eval. Comput. Syst.* 9, 4, Article 15 (December 2024), 28 pages. https://doi.org/10.1145/3701986

Authors' Contact Information: Robin Boëzennec, INRIA, Rennes, Bretagne, France, Université of Rennes, Rennes, Bretagne, France, CNRS, Rennes, Bretagne, France, and IRISA, IRISA, Bretagne, France; e-mail: robin.boezennec@inria.fr; Fanny Dufossé, Université Grenoble Alpes, Grenoble, Auvergne-Rhône-Alpes, France, INRIA, Grenoble, Auvergne-Rhône-Alpes, France, CNRS, Grenoble, Auvergne-Rhône-Alpes, France, and Grenoble INP UGA, Grenoble, Auvergne-Rhône-Alpes, France; e-mail: fanny.dufosse@inria.fr; Guillaume Pallez, INRIA, Rennes, Bretagne, France; e-mail: guillaume.pallez@inria.fr.

ACM 2376-3639/2024/12-ART15
https://doi.org/10.1145/3701986

1 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 with 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 making such a decision?

As an example, consider the classical job-packing problem: how to schedule parallel jobs on a homogeneous parallel platform. Classical scheduling heuristics exist 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. They 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 consists of neural networks trained with **reinforcement learning** (**RL**) techniques [9, 16, 30]. 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 depends on the metric used for optimization. Thus, the analysis of such algorithms cannot be based solely on the targeted metrics. It should consider qualitative criteria as packing efficiency, fairness or transparency.

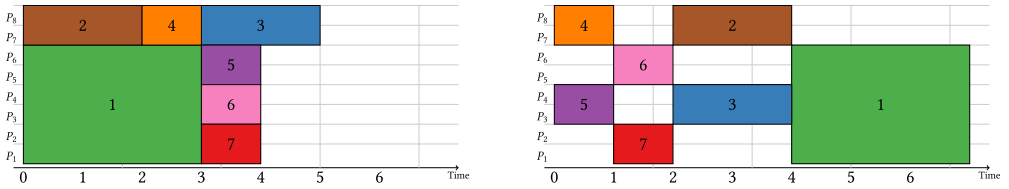
Understanding the bias and limitations of metrics for HPC resource management becomes even more important to help explain the behavior of such algorithms. This is the core problematic of this work.

Quantitative vs Qualitative analysis. Evaluation metrics can be used in several ways. Quantitatively, i.e., the typical optimization problem where one focuses on a number to optimize, the *performance*, or qualitatively. Qualitative analysis [22] comes from social science and is usually applied to individual or subset of individuals, typically through interviews, or by analyzing their specific behaviors. In resource management, we can extend such techniques by focusing on specific patterns, or behavior of specific jobs.

The goal of qualitative analysis is to provide a larger understanding of the phenomenon studied, increasing the trust in the study, and possibly discovering bias in the study. In this work, we argue that this type of analysis is becoming fundamental when considering black box algorithms.

Motivational examples. There are some famous examples about the limits of quantitative metrics. The response time is a metric that measures the length of time a job stayed in the system, however it does not allow to discriminate between a 1 min job which waited 1 hour to be executed (response time: 61 min), and a 1 hour job which waited 1 min to be executed (same response time). Faced to this, many authors have advocated the use of the slowdown [4, 30], i.e., the response time normalized by the execution time of a job.

However, looking at this metric is not satisfying either: in Figure 1, we show two possible schedules for a same instance problem with 7 jobs (released at time 0) and 8 nodes. Despite what appears



(a) Schedule A (b) Schedule B

Fig. 1. This example describes two schedules of the same set of jobs. All jobs are released at $t = 0$. Despite what appears to be a more efficient strategy, Schedule A has worse mean slowdown than Schedule B. This example is explained more thoroughly in Section 2.

to be a less interesting schedule (from an HPC perspective), from a mean slowdown perspective, Schedule B performs much better than Schedule A (see Section 2). From a qualitative perspective, this can be explained since we observe that the slowdown of (the numerous) small jobs is much smaller in Schedule B, hence impacting considerably the mean slowdown performance.

In this case, **looking simply at a quantitative objective hides critical information**, underlying the importance of a qualitative evaluation.

The final example against quantitative evaluation is how we value scientific contributions. As a community, we often value large gains over previous algorithms. Let U be the utilization (i.e., how well the machine is used, see Section 2 for a more thorough definition), an objective that one wants to maximize. By opposition, call I the idle occupation, $I = 1 - U$ which is an objective that one wants to minimize. Given an algorithm with a utilization of $U_1 = 95\%$ (this corresponds to current HPC utilization [23]). 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? How about 20%? This can be more subtle. Boito et al. [3] have provided I/O scheduling strategy that allow to improve the global system utilization of 2.5% (which can be considered low). To do this, their solution improve the I/O performance by 50% (which can be considered as high). Which is the correct metric to evaluate the performance of an I/O scheduler? In the context of I/O, this really stress out that presenting a single metric would not be sufficient, and that a qualitative analysis needs to present the full picture.

In this work, which extends considerably our preliminary discussion [2], we discuss several methodological elements to qualitatively study optimization criteria and the result of an analysis and argue for a more qualitative evaluation of resource management system. 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. For each objective, we show constructively how to improve or use them. We argue that the mean bounded slowdown and mean response time should not be used as quantitative objectives for evaluating resource management in HPC, particularly for RL-based scheduling. On the contrary, we show the relevance of the area weighted response time to measure the packing efficiency of HPC scheduling algorithms even in contexts 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

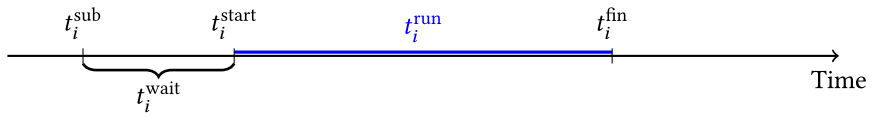


Fig. 2. A visual representation of the various notations.

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 performances, one cannot conclude a study.

(2) We strengthen our points with a qualitative analysis of the performances of RLScheduler [30]. RLScheduler is a neural-network batch scheduler for HPC, trained with RL. This use-case also serves us to discuss the importance of some good practices in performance evaluation. Our qualitative analysis contradicts the authors quantitative-based conclusions.

The rest of the article is constructed as follows: In Section 2, we discuss the objective criteria that are considered in the 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 our first use-case before presenting and analyzing the results in Section 3.2. Section 4 analyzes the behavior of RLScheduler and uses it to discuss several metrics. Section 5 discusses related work. Finally, Section 6 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. Indeed, extreme-scale platforms have a high operating cost and are expected to be utilized as much as possible. This includes the interconnect network which is one of the energy consuming part [18]. However, recent 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* [23], which questions the importance of having a dense interconnect, and ultimately of having HPC machines.

To define objectives, we use the following notations for job J_i (represented visually in Figure 2):

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 ($t_i^{wait} = t_i^{start} - t_i^{sub}$)
N_i^{cores}	The number of cores used by job J_i

2.1 Mean (Bounded) Slowdown

The mean bounded slowdown (also called mean flow) is an optimization criteria extensively used in the literature [4, 5, 19, 28, 30, 31]. Its goal is to provide a measure of fairness over applications.

[1]This is different from the requested/estimated time $t_i^{estimate}$ which we discuss in Section 3.

Table 1. Mean Slowdown of the Schedules in Figure 1

Metric	Schedule A	Schedule B
Mean slowdown (min is better)	≈ 2.8	≈ 1.8

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

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

One of the known limits of the slowdown is that in HPC traces many jobs are extremely small (few seconds). The slowdown of a job can be arbitrarily high even if its wait time is ridiculously small (a five minutes wait time for a job that dies instantly (one second) results in a slowdown of 300). The classical solution is to consider a variant of the slowdown called the *bounded slowdown*:

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

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

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

2.1.1 Limits for HPC Workloads. Improving the quality of service of small jobs considerably improves this objective. This is often what is actually measured when work studies this objective, and negatively impacts qualitative criteria as fairness and packing efficiency. This is illustrated in Figure 1. Indeed, Schedule B performs better in terms of mean bounded slowdown (Table 1) despite an apparent worse packing strategy than Schedule A.

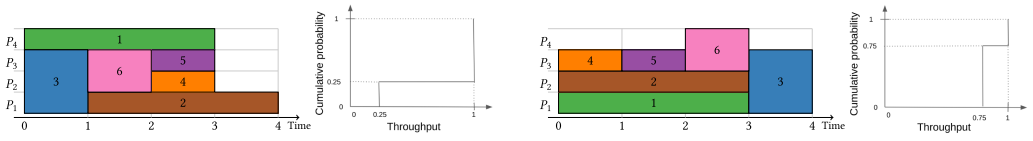
Work by Carastan-Santos et al. [4] where the ML algorithm provides a priority function confirms this intuition and the fact that learning-based batch schedulers with the objective of bounded slowdown simply give higher priority to small jobs. Similarly, Legrand et al. [19] have studied the importance 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. Other works [11, 12, 33] investigate the intrinsic bias of the slowdown to small jobs, even with the bounded variant.

In Section 3.2.1, 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 computing load. It makes it an unfit metric to evaluate the performance of RJMS in HPC.

2.1.2 Alternative Approach. To understand the actual behavior of the system, Du et al. [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 a more qualitative way to measure and understand the performance of a solution. Another approach is to use a weighted version of the mean 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, 14, 15].



(a) Schedule A (b) Schedule B

Fig. 3. Two schedules of the same tasks with their respective throughput cumulative distribution function. Even though the global utilization is the same between the two schedules (13/16), the distribution of their throughput differs significantly.

We first define the *throughput* ρ of the system. $\rho(t)$ is the amount of work being processed at time t, i.e., the number of nodes which are running at time t. We have also seen this notion called *instantaneous utilization* (when it is normalized by the number of nodes). The utilization $U(t_1, t_2)$ on the time interval $[t_1, t_2]$ is the mean throughput, normalized by the number of nodes so that the value is between 0 and 1. Hence with N nodes:

$$U(t_1, t_2) = \frac{\int_{t_1}^{t_2} \rho(t)dt}{N \cdot (t_2 - t_1)}. \tag{2}$$

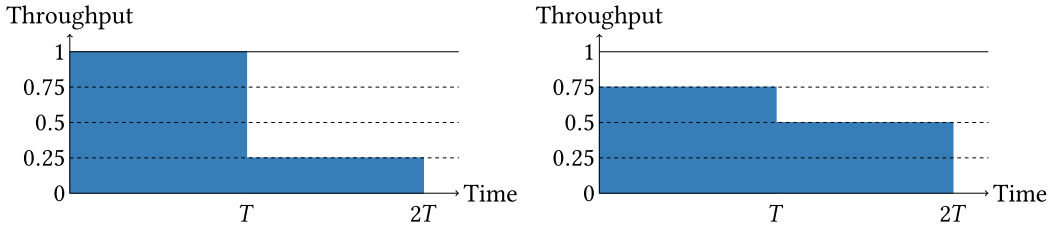
When jobs fail to complete fully (for instance because their walltime is underestimated), it is interesting to measure the "useful utilization", i.e. the volume of computation that leads to a successful execution [7].

2.2.1 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 [13]. Utilization by itself does not allow discriminating between different schedule qualities (Figure 3). The same limitations hold for non-steady states, which can occur when considering schedules of a limited workload.

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). The resulting delay for all jobs and the possible starvation would have no impact on the utilization metric.

2.2.2 Alternative Approach. Rudolph and Smith [26] have proposed to study the saturation point, i.e., the maximum utilization that can be achieved (just before system saturation). They propose to measure it by measuring the *knee* of the slowdown curve. Yet, this measurement is extremely subjective [13], and not well defined. It depends on the robustness of the slowdown curve. It is potentially very hard to measure if the slowdown has a high variability: how are you sure that you did enough experiments with various workload loads? In addition, it does not allow to discriminate behavior on systems that are not close to this point.

Similarly to Rudolph and Smith [26], our observations show that when the utilization of an HPC platform is lower than a certain threshold (which may correspond to the *saturation point*), the "quality" of a scheduler has no impact on the utilization of the schedule. 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. A good understanding of one's workload is important.



(a) Utilization: 0.625
Throughput std: 0.5625

(b) Utilization: 0.625
Throughput std: 0.125

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

We found that a way to evaluate this is to study the density function of the throughput (see Figure 3). Intuitively, for two identical job submission schemes, with different *modes*, a "better" scheduling algorithm has more phases at very high throughput (and hence more at lower throughput). Indeed, it can pack jobs as soon as they are available, whereas a poorer scheduling quality delays 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 throughput instead of its mean (i.e., its utilization). The standard deviation σ of the throughput ρ would therefore be computed as

$$\sigma^2 = \frac{\int_{t_1}^{t_2} \rho(t)^2 dt}{N^2 \cdot (t_2 - t_1)} - \frac{(\int_{t_1}^{t_2} \rho(t) dt)^2}{N^2 \cdot (t_2 - t_1)^2}. \tag{3}$$

Note that to be exact what is computed is the standard deviation of $\frac{\rho}{N}$, the throughput normalized by the number of nodes.

When a system is under-utilized, two schedules have an almost identical utilization, so we propose to measure the standard deviation of the throughput 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 completed (hence with peak of throughput vs a more balanced throughput). 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 throughput greater than the second schedule.

Some remarks on using the throughput's standard deviation:

(1) The throughput's standard deviation 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 cannot discriminate algorithms), to give information about which algorithm would be able to reach the higher utilization when the system is fully utilized.
(2) 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 overall? In Appendix A, we provide an answer for a particular example, but we have no general answer.

Table 2. Mean Response Time of the Schedules of Figure 1

Metric	Schedule A	Schedule B
Mean response time (min is better)	≈ 3.6	3

2.3 Response Time (and Wait Time)

Mean response time (or mean wait time) is a metric often used in the literature [10, 15, 21, 27, 28, 30]. 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.

2.3.1 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 1, we can see that the schedule on the left intuitively looks more efficient than the second, and yet it has a worst mean response time (see Table 2). This is because the schedule 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 3.2.1 that depending on the workload this objective is subject to a lot of variability.

2.3.2 Alternative Metric. Goponenko et al. [17] have argued for the use of the AWF, where one weights the response time by a priority proportional to the quantity of work (cores · time) of each job. In the following, we call this metric *WRT* for area-Weighted Response Time.

$$\text{WRT} = \frac{1}{\sum_i W_i} \sum_i W_i \cdot \mathcal{RT}_i. \tag{4}$$

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 throughput function (The function $t \mapsto \rho(t)$) 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).*

While the second result was mentioned [17], we did not find a formal proof of this result in the literature and have provided it in the Appendix B.

Interestingly, Proposition 1 highlights the fact that WRT is a metric that measures the quality of the throughput function, and hence evaluates the packing efficiency of algorithms. Compared with 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 utilization if one schedule performs work earlier than the other schedule. Table 3 shows that the area-Weighted Response Time is able to compare the two

Table 3. Utilization vs Area-Weighted Response Time of the Schedules of Figure 1

Metric	Schedule A	Schedule B
Utilization (max is better)	85%	39%
Weighted Response Time (min is better)	3.3	5

schedules of Figure 1 in an order that one would expect. WRT can therefore also be seen as an interesting alternative to the utilization metric as well as an alternative metric to Mean Response Time.

2.4 Additional Comments

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

How to correctly measure 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 3.1.4 and 4.2.3.

In summary, 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 [30, 31]. 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 4 by correctly analyzing the results.

In the next Sections, we show on two specific use-cases what happens when one only relies on optimizing quantitative criterion without performing a qualitative analysis.

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 [1, 23] 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 3.1.3).

Specifically, we show on real data that the undesirable behaviors of some metrics are extremely common:

— For some metric, relative comparisons between schedules are highly dependent on the workloads. A high variability in the performance should serve as a warning that the evaluation needs more qualitative analysis.
— Specifically, mean Response Time and Mean Bounded Slowdown **should not be used** to evaluate quantitatively a solution, but they can help understand qualitatively its performance.
— When a platform is under-utilized, the utilization can be an irrelevant objective. In this particular case, the standard deviation of the throughput allows to compare various algorithms in order to determine which one would be able to reach the highest utilization when the platform is fully-utilized.

3.1 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.

3.1.1 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.

3.1.2 Workloads. We used traces from Mira and Theta[2] supercomputers at Argonne National Laboratory. The Mira supercomputer was launched in 2012 at the 3rd place of TOP500[3] HPC centers. It ran 49 152 nodes and was maintained until 2019. The available trace covers years 2014 to 2018 and 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 [23]. Doing this helps the scheduler to better pack jobs at the risk of having allocated - but underused/unused - nodes.

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 sets of 10 000 consecutive jobs) for Mira, and from index 1 to 400 000 (by sets of 10 000 consecutive jobs) 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, 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 1001st 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}.

3.1.3 Runtime Estimate Functions. This use-case is focused on the precision of runtime estimates. The goal is to demonstrate the difference between a purely quantitative analysis and a qualitative analysis. We define two walltime functions. Given r the runtime of a job (in seconds):

— Exact : $r \mapsto r + 1$ second. This simulates an almost perfect estimate.
— User-walltime, it corresponds to the walltime provided by the users.

3.1.4 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.

Utilization related objectives. The utilization and the standard deviation are measured on a given time window as presented in Equation (3). If only a part of a job is inside the window, we ignore

[2] https://reports.alcf.anl.gov/data/
[3] https://www.top500.org/

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 [28]. The measurement window $[t_1, t_2]$ is defined s.t.:

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

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 means 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 [28].

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, throughput standard deviation), then

$$\text{RI}(O) = \frac{O_{\text{Exact}} - O_{\text{User-walltime}}}{O_{\text{User-walltime}}}. \tag{5}$$

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

$$\text{RI}(O) = \frac{O_{\text{User-walltime}} - O_{\text{Exact}}}{O_{\text{User-walltime}}}. \tag{6}$$

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.

3.2 Result Analysis

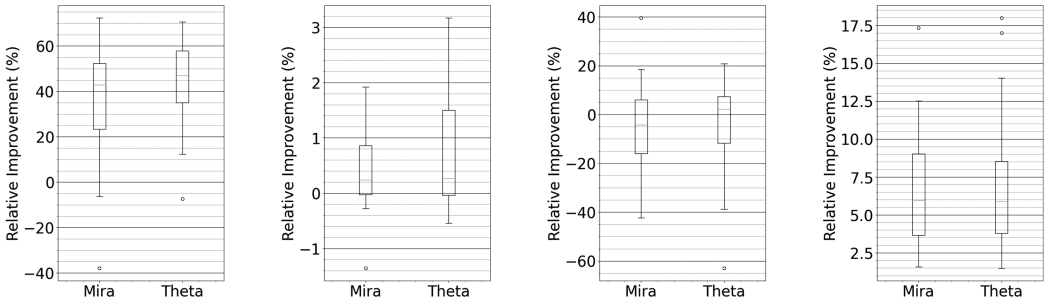
In this section, we investigate the impact of walltime accuracy in the performance of Resource and Job Management Software in order to discuss various metrics.

The two runtime estimate functions are evaluated in Figure 5 over the various criteria discussed in Section 2: Bounded Slowdown (Section 2.1), Utilization (Section 2.2), Response Time and WRT (Section 2.3). In these figures, to discuss the performance difference between the two runtime estimate functions, we use the relative improvement of Exact over User-walltime as computed by Equations (5) and (6).

The quantitative results in Figure 5 demonstrate the impact 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.

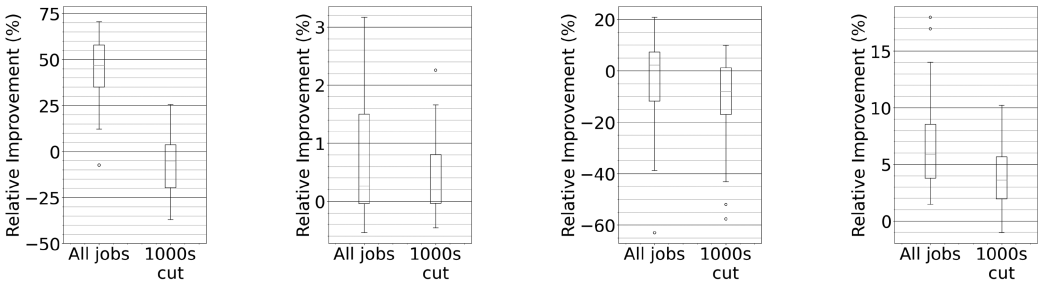
Looking at mean values is not enough. In the next section, we analyze these results in depth.

3.2.1 Mean-Bounded Slowdown and Response Time. Figure 5(a) (on bounded slowdown metric) and 5(c) (on mean response time) indicate an important variability of the performance. Discussing



(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.



(a) Mean 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.

the mean of an objective with high variability is meaningless: the performance is highly influenced by the workload

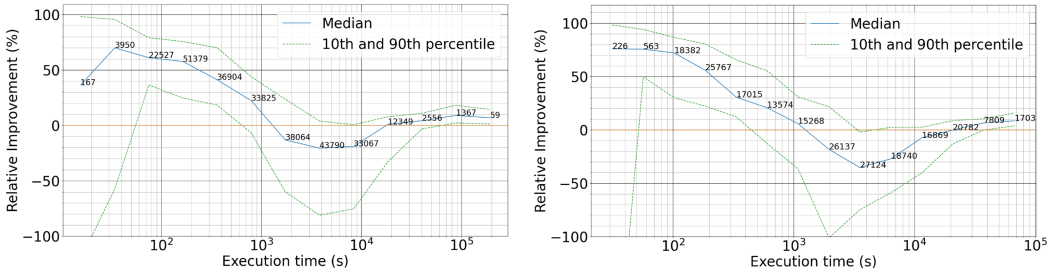
Another way to confirm this is to transform the computing load and see the impact on the objective. In Figure 6, we compare the performance of the two algorithms on the workload that contains all jobs, and on the workload where we have removed all jobs that last less than 1000 seconds. *After this transformation, 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 with respect to Mean Bounded Slowdown and Response Time 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 other 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 them 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.

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



(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.

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 the 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-length jobs and long jobs. When switching from User-walltime to Exact:

— Exact improves considerably the response time of short jobs;
— Meanwhile, medium-length jobs see an increased mean 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 qualitative analysis help understanding the behavior of an algorithm over another one. 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.

We can verify this intuition by measuring the number of jobs that are indeed backfilled (see Table 4, to read it: 82% of the 142k jobs that are smaller than 1000s are backfilled with User-walltime).

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.

3.2.2 *Utilization.*

Qualitative analysis. In Figure 5(b), the improvement of utilization is extremely small (about 0.5%). As we explained in Section 2.2, this is expected and is an artifact of the non-constant arrival

Table 4. Number of Jobs Backfilled with User-walltime and Exact in Function of their Runtime (Theta Data)

	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

We removed the first and last 15% jobs of each sample as they are not used to compute response time.

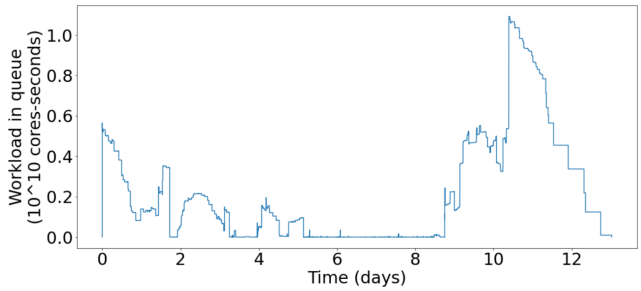
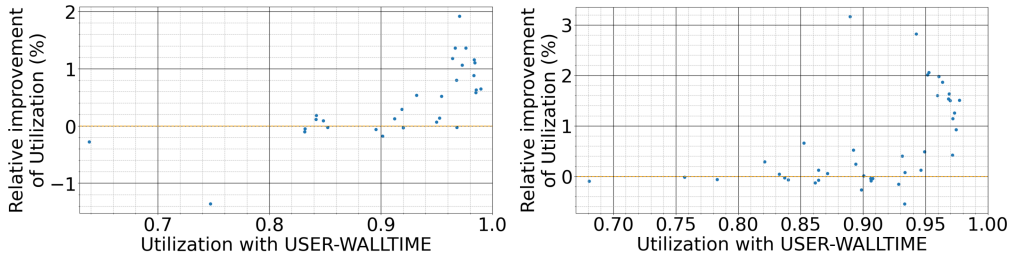


Fig. 8. Available work in the waiting queue as a function of time for one of workload sample from Mira when scheduled with the Exact walltime function. This shows that the availability of jobs is not regular throughout the execution.

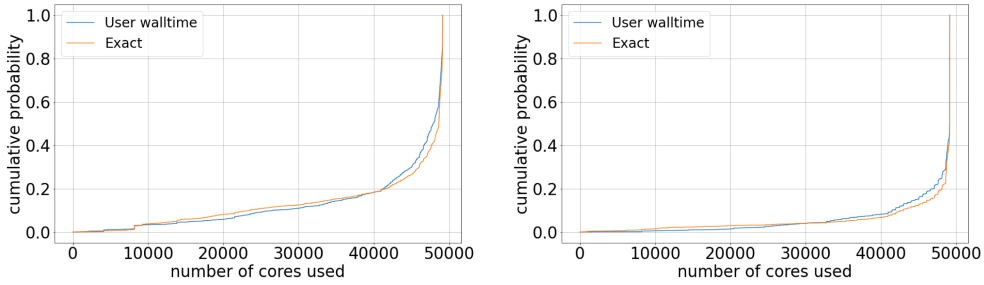


(a) Data from Mira (b) Data from Theta

Fig. 9. Relative improvement of the Utilization of Exact over User-walltime as a function of the Utilization with User-walltime.

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 over a certain threshold.



(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 throughput for two selected scenarios from Mira where the utilization difference between Exact and User-walltime is close to 0.

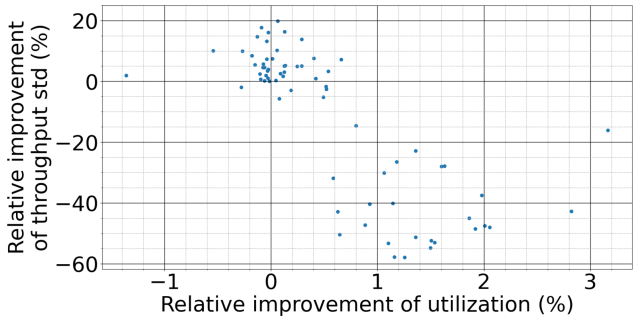


Fig. 11. Relative improvement of the standard deviation of the throughput as a function of the relative improvement of the utilization (Mira and Theta data).

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%).

Toward better objectives. As a quantitative objective to be able to compare various algorithms, we propose to measure the standard deviation (std) of the throughput. For two algorithms with identical utilization, high standard deviation imply large variation of throughput, that we correlate with more periods of high throughput and more periods of low throughput.

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 when the platform is fully-utilized.

4 Use-Case: Reinforcement-Learning for Resource Management

In the previous section, we have claimed that using the mean bounded slowdown is a problem when used to evaluate quantitatively a solution. We have claimed in Section 2.4 that this is 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. [4], 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. [30] proposes 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. [30] 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 90 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.

4.1 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.

4.1.1 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,

Table 5. Various Performance Difference between Fcfs-BF, and RLSched (Trained on the Corresponding Trace) [30]

	Mean BSD		Utilization		Max BSD	
Trace	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	–	–

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.

4.1.2 Traces. The traces included have a total of 10 000 jobs. For their evaluations, Zhang et al. [30] 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 5).

4.2 Analysis of Results

When compared with Fcfs-BF, Zhang et al. [30] reported the following performance of the RLScheduler mechanism with backfilling:

— It strongly and consistently improves the Mean Bounded Slowdown [30, Table V, Table VII].
— It sometimes slightly improves utilization, and sometimes slightly hurts it [30, Table VI].
— It strongly improves the Maximum Bounded Slowdown [30, Table VIII].

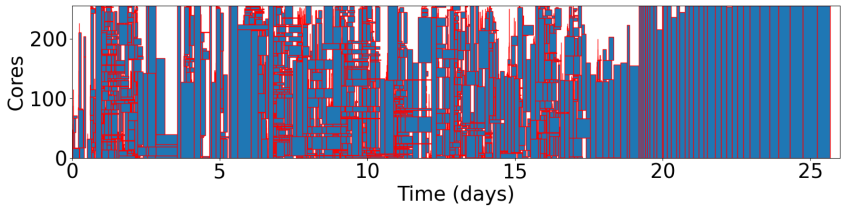
For completeness, these results are reported in Table 5.

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.

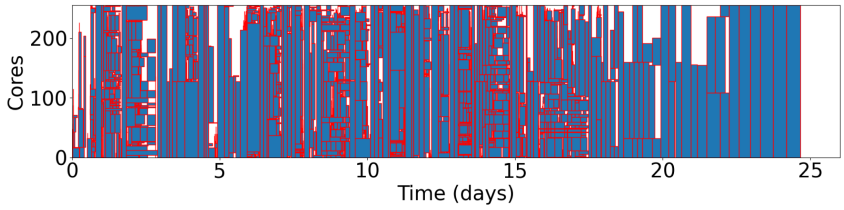
4.2.1 Visual Representation. Before providing a qualitative discussion of the results, we present in Figure 12 the Gantt chart of three schedules (with RL model and without). All these schedules are computed on the same job dataset.

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

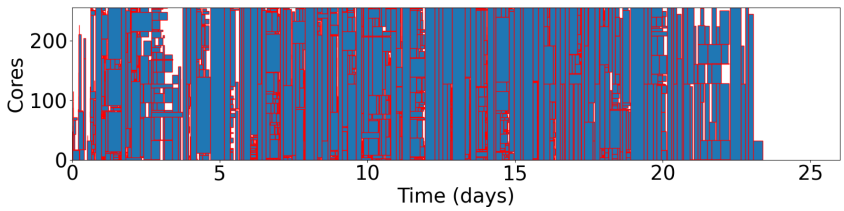
Already, one can clearly see on these examples a behavior of RLScheduler that schedules many *large* jobs at the end of the schedule. This is extremely concerning as it intuits starvation. In addition, an observation is that the utilization seems extremely unbalanced throughout the execution. All this 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.



(a) RLSched-Mbsd-Lub1



(b) RLSched-Util-Lub1



(c) Fcfs-BF

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

4.2.2 Mean Bounded Slowdown. We have showed in Sections 2.1 and 3.2.1 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 boxplotted the wait time of the jobs of each group.

The data observed in 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 6, 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.

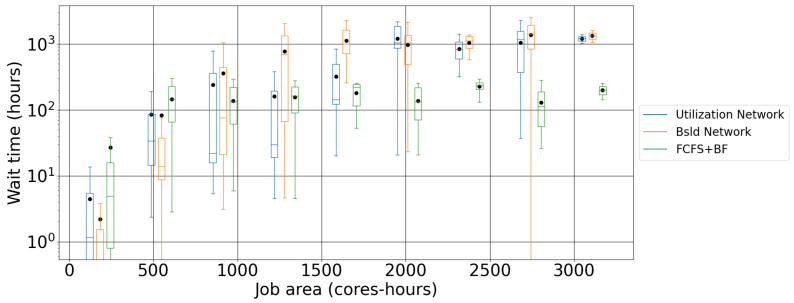


Fig. 13. Wait time as a function a job work for several schedulers.

Table 6. Evolution of the Mean Bounded Slowdown Metric when Deleting Short Jobs

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

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

4.2.3 Utilization. All scenarios have quite low utilization: the highest utilization observed by the authors is Lublin-1 with 87% (see Table 5). This contradicts many recent results about utilization in HPC centers (for instance, on Mira the average utilization is 95% [23]).

This could be explained if the trace had a low submission rate, However, in this case, on average the system has a load much higher that what it can deal with. In Figure 12 no job is submitted after day 18, but the execution of the trace lasts until after day 25. Indeed, the mean arrival rate of Lublin-1 is 272 cores-seconds per second but the platform has 256 cores.

The explanation comes from an important methodological error in the evaluation of the utilization. When measuring utilization, Zhang et al. [30] measure the utilization of the whole execution of their small traces and not only in steady-state. 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 by each new small job arrival, 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 3.1.4.

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 [23] 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.

To conclude 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

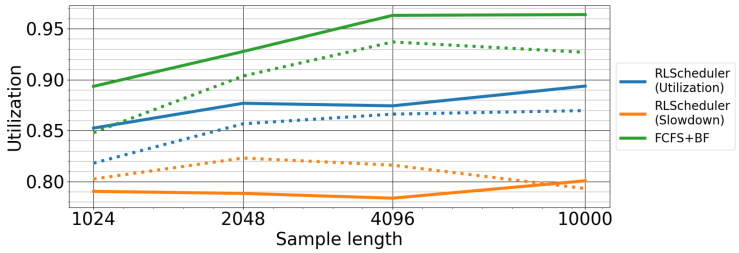
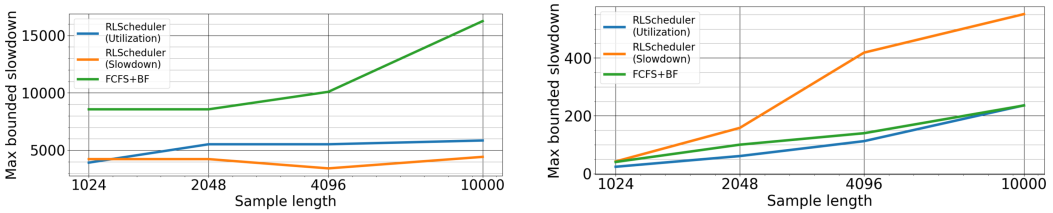


Fig. 14. Utilization for various algorithms as a function of the size of samples with two measurement strategies. The full (resp. dotted) lines show the utilization with (resp. without) measurement bounds. The samples of length 1024, 2048, and 4096 start at index 5000, and the sample of size 10 000 starts at index 0.



(a) τ = 10 seconds (b) τ = 1 hour

Fig. 15. Max bounded slowdown of FCFS+Backfilling, and RLScheduler+Backfilling.

choice of evaluation criteria, one should be extremely careful about the methodological evaluation. Indeed, the methodology provided by Zhang et al. [30] seemed to imply that the difference in utilization was extremely low, while it is actually quite important when we consider a much longer trace.

4.2.4 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 5). Note that the authors did not measure it for Lublin-1, but experimental evaluation confirms their quantitative observations (see Figure 15(a)).

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 computing the bounded slowdown (Equation (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 15(b)) to 1 hour. 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!

4.2.5 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).

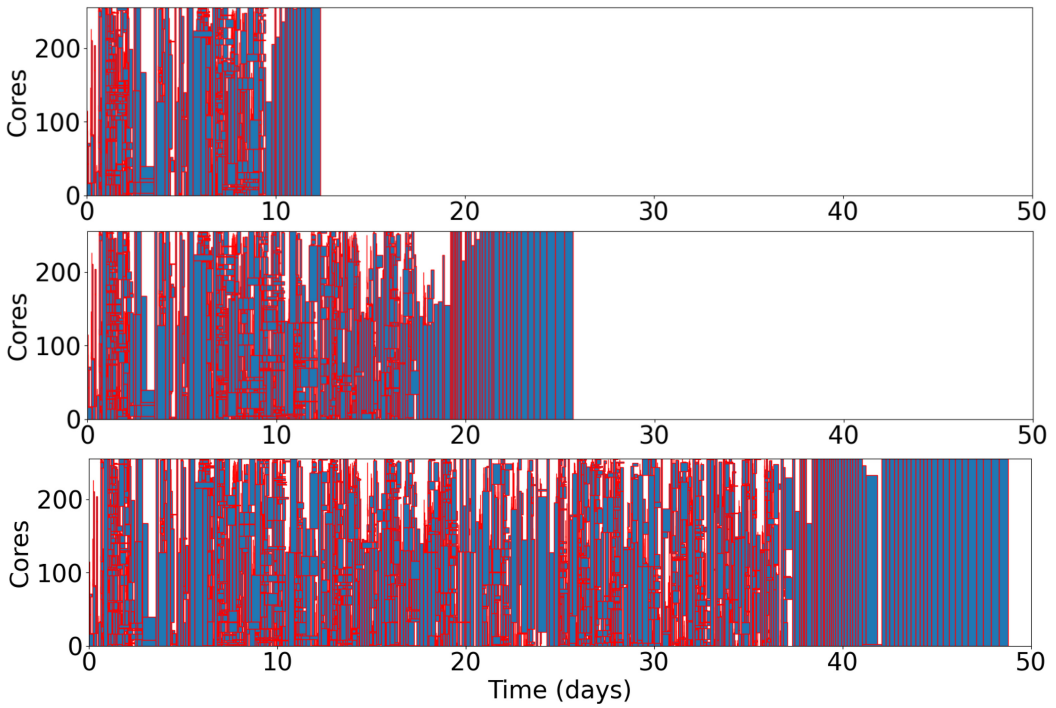


Fig. 16. Gantt chart obtained with RLScheduler-MBSD depending on the number of jobs used (Top : 1024 jobs, middle : 2048 jobs, bottom : 4096 jobs).

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 RL Strategy delays large job indefinitely. This is less visible in the article because the authors only consider sets of 1,024 jobs, but becomes more apparent if we increase the number of jobs (see Figure 16).

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

5 Related Work

The question of metrics is a long standing question. Rudolph and Smith [26] had alerted fairly early about the risks of measuring utilization and had proposed alternative methods. Similarly, several alert were made about the fact that the slowdown metric was favoring the smallest job [11, 12, 33]. Frachtenberg and Feitelson [13] had provided four pitfalls on using evaluation metrics for open models. They discussed in depth the Utilization metric. They concluded that one should use (bounded) response time or slowdown to evaluate parallel job scheduling. This seems to come from a more theoretical standpoint where utilization was not usable, but our work, and particularly our results on real traces contradicts their recommendation, as one can verify that these metrics have a high variability depending on the workload.

Here our approach leaves the broader theoretical framework, and demonstrate the results through two use-case as well as an extensive analysis of good and bad practices when evaluating schedules.

Goponenko et al. [17] recently focused 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. [29] 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. [20] 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. [32], use it to measure the average fraction of idle nodes when there are waiting jobs. Some authors [6, 24] consider the mean response time and bounded slowdown for different categories of jobs based on their duration and number of requested nodes.

Raji et al. [25] 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. We confirm the risk of such black box systems here.

6 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. We have shown in this work that there is a fundamental question about how to analyze qualitatively resource management systems. Through several examples, we have provided some guidelines on how it could be done. Yet the much larger question of developing a theory of qualitative analysis for Resource Management systems remains.

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 and 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.

Appendices

A Interpretation of Standard Deviation of the Throughput

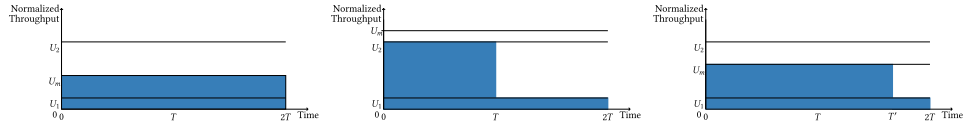
This section gives an intuition of standard deviation of the throughput and its link with utilization based on a simple example.

We consider a platform with N_c resources, and a continuous submission of tasks. Tasks have an average workload equal to W (or work = number of resources used × runtime). We consider two different phases. First from time $t = 0$ to $t = T$, jobs are submitted with a high arrival rate (=number of jobs submitted by unit of time) λ_1, and then from T to $2T$ jobs are submitted with a lower arrival rate λ_2, with $\lambda_2 < \lambda_1$. The workload arriving in queue per unit of time (called $\lambda(t)$) is equal to:

$$\lambda(t) = \begin{cases} \lambda_1 W & \text{if} \quad t \in [0, T] \\ \lambda_2 W & \text{if} \quad t \in [T, 2T] \end{cases}$$

The scheduler performance is expressed by the constant U_m, which is the maximum utilization that it can reach. We suppose that if there are enough submitted jobs, the scheduler always reaches this maximum utilization. We also introduce U_i, for $i \in \{1, 2\}$ which corresponds to the minimum utilization required to ensure that the arriving workload is smaller than the machine capacity when the arrival rate is λ_i. As the arriving workload is equal to $\lambda_i \cdot W$ and the system capacity to $N_c \cdot U$, we can deduce that $U_i = \frac{W\lambda_i}{N_c}$. As $\lambda_2 < \lambda_1$, we also have $U_2 < U_1$.

Depending on the scheduler maximum utilization of U_m compared with U_1 and U_2, we have different scheduling configurations. They are shown in Figure 17, where we use the normalized throughput to designate the throughput divided by the number of cores.



(a) Case 1 (b) Case 2 (c) Case 3

Fig. 17. The different configurations of the schedule.

— Case 1: $U_m < \frac{U_1+U_2}{2}$. The schedule does not execute the workload before $2T$, the normalized throughput is constant at U_m (and so the utilization is equal to U_m) and the throughput variance is null.
— Case 2: $U_m \geq U_1$: The schedule executes jobs when they are submitted without putting them in the queue. The normalized throughput is equal to U_1 in the first phase and U_2 in the second. This results in a utilization of $\frac{U_1+U_2}{2}$ and a throughout variance of $\frac{(U_1-U_2)^2}{4}$.
— Case 3: $\frac{U_1+U_2}{2} \leq U_m < U_1$. The schedule runs all jobs, but some jobs submitted in the first phase are scheduled during the second one. As the scheduler always reach a normalized throughput as high as possible this leads to a normalized throughput of U_m when $t \in [0, T']$ with a certain $T' > T$, and then a normalized throughput of U_2 during $[T', 2T]$. In order to

compute the mean and variance of the throughput, we need to determine T'. At $t = T'$, there are no jobs lefts in the queue, meaning that the workload arrived since $t = 0$ is equal to the quantity of work that has been run by the machine. Hence, we have $\lambda_1 TW + \lambda_2(T' - T)W = U_m T' N_c$, which gives us $T' = T\frac{U_1 - U_2}{U_m - U_2}$. We can then compute the utilization which is equal to $\frac{U_1+U_2}{2}$ (as we are scheduling a constant workload that fits in a constant time window, it is natural that the mean utilization stays a constant). The throughput variance is equal to $\frac{U_1-U_2}{2}(U_m - \frac{U_1+U_2}{2})$.

Figure 18 summarizes how the mean and variance of throughput change with the maximum utilization. Once U_m is higher than $\frac{U_1+U_2}{2}$, the utilization stops increasing and the throughput variance starts growing.

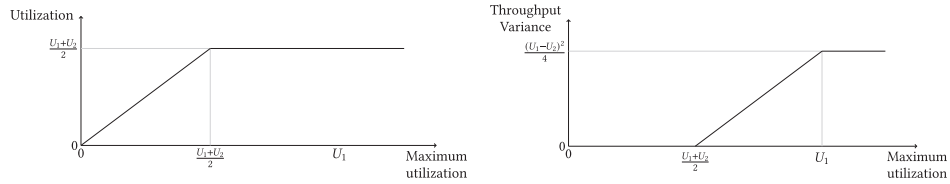


Fig. 18. Utilization (left) and throughput variance (right) function of the maximum utilization U_m.

If we have two schedules with the same utilization and throughput variances V_1 and V_2, such as $\frac{V_1}{V_2} = \alpha$, and if we make the hypothesis that both schedules correspond to the second case, we can compute that:

$$U_m^1 = \alpha U_m^2 + (1 - \alpha)\frac{U_1 + U_2}{2}$$

This allows telling how much an algorithm is better than another. In the same way, if we make the hypothesis that a schedule is in the second case, we can just invert the formula and obtain that $U_m = V\frac{2}{U_1 - U_2} + \frac{U_1+U_2}{2}$.

Although these formulas give us numerical values, this is only a particular case, and we are unable to interpret the value of the variance in the general case.

B Invariance by Job Permutation of the Area-weighted Response Time

Theorem 1. *Given a schedule, if one permutes jobs without changing its throughput function ($t \mapsto \rho(t)$), the area-weighted mean response (resp. wait) time stays the same.*

To do so, we define a new metric, the *half completion* (HC), which is the average between the area-weighted response time and the area-weighted wait time.

$$HC_i = t_i^{\text{run}} \cdot N_i^{\text{cores}} \cdot \left(t_i^{\text{wait}} + \frac{t_i^{\text{run}}}{2}\right)$$

and the global metric is equal to:

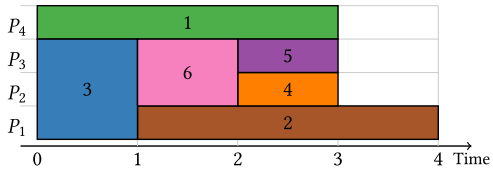
$$\text{HC} = \sum_{i \in J} HC_i = \sum_{i \in J} t_i^{\text{run}} \cdot N_i^{\text{cores}} \cdot \left(t_i^{\text{wait}} + \frac{t_i^{\text{run}}}{2}\right),$$

where J is the set of all jobs. The half completion time is also weighted by area, but for convenience, we do not precise it in its name. Note that normally the half completion time should be divided by $\sum_{i \in J} t_i^{\text{run}} \cdot N_i^{\text{cores}}$ which is equal to the total quantity of work. We will suppose without loss of generality that this constant is equal to 1.

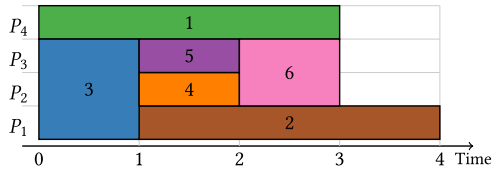
Lemma 1. *Given a schedule, the following operations do not impact the half completion time:*

(1) *Permute jobs without changing the throughput function.*
(2) *Merge jobs, and set the submission time of the new job as the mean submission time (weighted by work) of the merged jobs.*
(3) *Split a job in several smaller jobs, such as the mean submission time (weighted by work) of the new job is equal to the submission time of the separated job.*

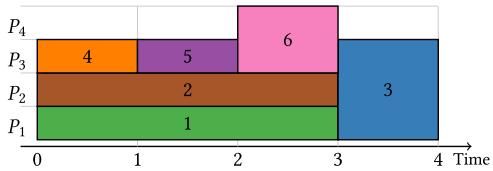
We illustrate graphically Lemma 1 on Figure 19.



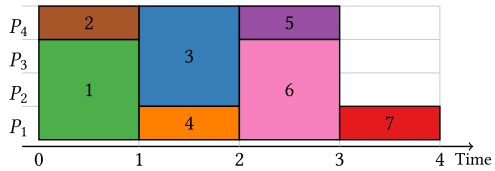
(a) Reference schedule, WRT= 34, HC= 21.5.



(b) When the set of jobs and the throughput function are constant, then neither WRT and HC change (here: WRT= 34, HC= 21.5).



(c) When the set of jobs is constant but the throughput function changes, then both WRT and HC change (here: WRT= 37, HC= 23.5).



(d) When the set of job changes, but the throughput function stays the same, WRT changes but HC stays the same (here: WRT= 28, HC= 21.5).

Fig. 19. Several scheduling configurations and their corresponding WRT and half completion time (both weighted by work). All jobs are assumed to be queued at t = 0. Area-weighted wait time behaves like the WRT.

Intuition behind the proof: using half completion time places the point of measure of a job's metric at its barycenter. This allows us to decompose the half completion time of a job as the sum of all the half completion time we obtain by dividing this job in infinitesimally small jobs. We can then swap these infinitesimally small jobs without changing the metric. This allows us to totally change the original job configuration while preserving the half completion time.

Proof of Lemma 1. The key of the proof is that HC_i can also be written as the integral of all the wait time that we obtain if we divide the job i in an infinity of infinitesimally small rectangular jobs (all submitted at the same time t_i^{sub}).

$$HC_i = t_i^{\text{run}} N_i^{\text{cores}} \left(t_i^{\text{wait}} + \frac{t_i^{\text{run}}}{2} \right) = N_i^{\text{cores}} \int_{t_i^{\text{wait}}}^{t_i^{\text{wait}}+t_i^{\text{run}}} t dt = \int_0^{N_i^{\text{cores}}} \left(\int_{t_i^{\text{wait}}}^{t_i^{\text{wait}}+t_i^{\text{run}}} t dt \right) dN$$

Then, we can rewrite HC with this formula:

$$\text{HC} = \sum_{i \in J} \int_0^{N_i^{\text{cores}}} \left(\int_{t_i^{\text{wait}}}^{t_i^{\text{wait}}+t_i^{\text{run}}} t dt \right) dN$$

Then, in each of these integrals, we perform the change of variable $t_{\text{old}} = t_{\text{new}} - t_i^{\text{sub}}$. This changes the bound $t_i^{\text{wait}} + t_i^{\text{run}}$ in $t_i^{\text{wait}} + t_i^{\text{run}} + t_i^{\text{sub}} = t_i^{\text{fin}}$, and the bound t_i^{wait} in $t_i^{\text{wait}} + t_i^{\text{sub}} = t_i^{\text{start}}$.

Therefore:

$$\text{HC} = \sum_{i \in J} \int_0^{N_i^{\text{cores}}} \left(\int_{t_i^{\text{start}}}^{t_i^{\text{fin}}} \left(t - t_i^{\text{sub}} \right) dt \right) dN = \sum_{i \in J} \int_0^{N_i^{\text{cores}}} \int_{t_i^{\text{start}}}^{t_i^{\text{fin}}} t \; dt dN - \sum_{i \in J} \int_0^{N_i^{\text{cores}}} \int_{t_i^{\text{start}}}^{t_i^{\text{fin}}} t_i^{\text{sub}} \; dt dN$$

We then define U as the surface used by the jobs contained in the set J, and we define C_i as the set of cores on which job i is running. We have:

$$U = \bigcup_{i \in J} (C_i \times [t_i^{\text{start}}, t_i^{\text{fin}}])$$

Where $\times$ represents a Cartesian product. Note that as long as the throughput function does not change, the surface U stays the same. We can then rewrite HC as

$$\text{HC} = \underbrace{\iint_{\text{U}} t \text{dt dN}}_{A} - \underbrace{\iint_{\text{U}} t_i^{\text{sub}} \text{dt dN}}_{B} = A - B$$

A is the integral of a function over the surface U. In addition, this function does not depend on the jobs. So, as long as U is kept unchanged, we can split and merge or permute jobs without affecting A.

B is the integral of another function over the surface U. However this time the function depends on the jobs. In this case, as long as U is kept unchanged, we can reorder jobs without changing the metric. In addition, we can also split and merge jobs following the rules described in the Theorem. This concludes the demonstration of the properties of half completion time. □

We can now deduce the results on WRT (and area-weighted wait time):

Proof of Theorem 1. The difference between the half completion time and WRT is equal to:

$$D = \sum_{i \in J} \frac{N_i^{\text{cores}} (t_i^{\text{run}})^2}{2}$$

This is also equal to the difference between wait time (weighted by work) and half completion time. D depends on the jobs characteristics but not their order, hence if we switch jobs order without changing the function $\rho(t)$, WRT (and wait time weighted by area) stays the same. □

Acknowledgments

We want to thank the reviewer for their thorough evaluation. Parts of this work have been supported by the Inria Exploratory Project REPAS and by the Exa-DoST project (PEPR NumPEx), reference ANR-22-EXNU-0004. We also thank Millian Poquet for the help he provided us with the software Batsim.

References

[1] Cynthia Bailey Lee, Yael Schwartzman, Jennifer Hardy, and Allan Snavely. 2004. Are user runtime estimates inherently inaccurate?. In *Workshop on Job Scheduling Strategies for Parallel Processing*. Springer, 253–263.

[2] Robin Boëzennec, Fanny Dufossé, and Guillaume Pallez. 2023. 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*. 1–19.

[3] Francieli Boito, Guillaume Pallez, Luan Teylo, and Nicolas Vidal. 2023. IO-SETS: Simple and efficient approaches for I/O bandwidth management. *IEEE Transactions on Parallel and Distributed Systems* 34, 10 (2023), 2783–2796.

[4] Danilo Carastan-Santos and Raphael Y. De Camargo. 2017. Obtaining dynamic scheduling policies with simulation and machine learning. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*. 1–13.

[5] Danilo Carastan-Santos, Raphael Y. De Camargo, Denis Trystram, and Salah Zrigui. 2019. 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'19)*. IEEE, 1–10.

[6] Su-Hui Chiang, Andrea Arpaci-Dusseau, and Mary K. Vernon. 2002. 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*. Springer, 103–127.

[7] Yishu Du, Loris Marchal, Guillaume Pallez, and Yves Robert. 2024. Improving batch schedulers with node stealing for failed jobs. *Concurrency and Computation: Practice and Experience* 36, 12 (2024), e8043.

[8] Pierre-François Dutot, Michael Mercier, Millian Poquet, and Olivier Richard. 2016. Batsim: A realistic language-independent resources and jobs management systems simulator. In *20th Workshop on Job Scheduling Strategies for Parallel Processing*. Chicago, United States. Retrieved from https://hal.archives-ouvertes.fr/hal-01333471

[9] Yuping Fan, Boyang Li, Dustin Favorite, Naunidh Singh, Taylor Childers, Paul Rich, William Allcock, Michael E. Papka, and Zhiling Lan. 2022. DRAS: Deep reinforcement learning for cluster scheduling in high performance computing. *IEEE Transactions on Parallel and Distributed Systems* 33, 12 (2022), 4903–4917. DOI: https://doi.org/10.1109/TPDS.2022.3205325

[10] Yuping Fan, Paul Rich, William E. Allcock, Michael E. Papka, and Zhiling Lan. 2017. Trade-off between prediction accuracy and underestimation rate in job runtime estimates. In *2017 IEEE International Conference on Cluster Computing (CLUSTER'17)*. IEEE, 530–540.

[11] Dror G. Feitelson. 2001. Metrics for parallel job scheduling and their convergence. In *Workshop on Job Scheduling Strategies for Parallel Processing*. Springer, 188–205.

[12] Dror G. Feitelson. 2003. Metric and workload effects on computer systems evaluation. *Computer* 36, 09 (2003), 18–25.

[13] Eitan Frachtenberg and Dror G. Feitelson. 2005. Pitfalls in parallel job scheduling evaluation. In *Job Scheduling Strategies for Parallel Processing: 11th International Workshop, JSSPP 2005, Cambridge, MA, USA, June 19, 2005, Revised Selected Papers 11*. Springer, 257–282.

[14] Ana Gainaru and Guillaume Pallez. 2019. Making speculative scheduling robust to incomplete data. In *2019 IEEE/ACM 10th Workshop on Latest Advances in Scalable Algorithms for Large-Scale Systems (ScalA'19)*. IEEE, 62–71.

[15] Ana Gainaru, Guillaume Pallez, Hongyang Sun, and Padma Raghavan. 2019. Speculative scheduling for stochastic HPC applications. In *Proceedings of the 48th International Conference on Parallel Processing*. 1–10.

[16] Eric Gaussier, Jérôme Lelong, Valentin Reis, and Denis Trystram. 2018. Online tuning of EASY-backfilling using queue reordering policies. *IEEE Transactions on Parallel and Distributed Systems* 29, 10 (2018), 2304–2316.

[17] Alexander V. Goponenko, Kenneth Lamar, Christina Peterson, Benjamin A. Allan, Jim M. Brandt, and Damian Dechev. 2022. 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'22)*. 241–252. DOI: https://doi.org/10.1109/SBAC-PAD55451.2022.00035

[18] Peter Kogge and John Shalf. 2013. Exascale computing trends: Adjusting to the "new normal" for computer architecture. *Computing in Science & Engineering* 15, 6 (2013), 16–26.

[19] Arnaud Legrand, Denis Trystram, and Salah Zrigui. 2019. Adapting batch scheduling to workload characteristics: What can we expect from online learning? In *2019 IEEE International Parallel and Distributed Processing Symposium (IPDPS'19)*. IEEE, 686–695.

[20] Vitus J. Leung, Gerald Sabin, and Ponnuswamy Sadayappan. 2010. Parallel job scheduling policies to improve fairness: A case study. In *2010 39th International Conference on Parallel Processing Workshops*. IEEE, 346–353.

[21] Ahuva W. Mu'alem and Dror G. Feitelson. 2001. 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.

[22] Martin J. Packer. 2017. *The Science of Qualitative Research*. Cambridge University Press.

[23] Tirthak Patel, Zhengchun Liu, Raj Kettimuthu, Paul Rich, William Allcock, and Devesh Tiwari. 2020. Job characteristics on large-scale systems: Long-term analysis, quantification, and implications. In *SC20: International Conference for High Performance Computing, Networking, Storage and Analysis*. IEEE, 1–17.

[24] Dejan Perkovic and Peter J. Keleher. 2000. Randomization, speculation, and adaptation in batch schedulers. In *SC'00: Proceedings of the 2000 ACM/IEEE Conference on Supercomputing*. IEEE, 7–7.

[25] Inioluwa Deborah Raji, I. Elizabeth Kumar, Aaron Horowitz, and Andrew Selbst. 2022. The fallacy of AI functionality. In *2022 ACM Conference on Fairness, Accountability, and Transparency*. 959–972.

[26] Larry Rudolph and Paul H. Smith. 2000. Valuation of ultra-scale computing systems. In *Workshop on Job Scheduling Strategies for Parallel Processing*. Springer, 39–55.

[27] Wei Tang, Zhiling Lan, Narayan Desai, and Daniel Buettner. 2009. Fault-aware, utility-based job scheduling on blue, gene/p systems. In *2009 IEEE International Conference on Cluster Computing and Workshops*. IEEE, 1–10.

[28] Dan Tsafrir, Yoav Etsion, and Dror G. Feitelson. 2007. Backfilling using system-generated predictions rather than user runtime estimates. *IEEE Transactions on Parallel and Distributed Systems* 18, 6 (2007), 789–803.

[29] Abhishek Verma, Madhukar Korupolu, and John Wilkes. 2014. Evaluating job packing in warehouse-scale computing. In *2014 IEEE International Conference on Cluster Computing (CLUSTER'14)*. IEEE, 48–56.

[30] Di Zhang, Dong Dai, Youbiao He, Forrest Sheng Bao, and Bing Xie. 2020. RLScheduler: An automated HPC batch job scheduler using reinforcement learning. In *SC20: International Conference for High Performance Computing, Networking, Storage and Analysis*. IEEE, 1–15.

[31] Di Zhang, Dong Dai, and Bing Xie. 2022. SchedInspector: A batch job scheduling inspector using reinforcement learning. In *Proceedings of the 31st International Symposium on High-Performance Parallel and Distributed Computing (HPDC'22)*. Association for Computing Machinery, New York, NY, 97–109.

[32] Yanyong Zhang, Hubertus Franke, José E. Moreira, and Anand Sivasubramaniam. 2000. Improving parallel job scheduling by combining gang scheduling and backfilling techniques. In *Proceedings 14th International Parallel and Distributed Processing Symposium (IPDPS'00)*. IEEE, 133–142.

[33] Dmitry Zotkin and Peter J. Keleher. 1999. Job-length estimation and performance in backfilling schedulers. In *Proceedings. The Eighth International Symposium on High Performance Distributed Computing (Cat. No. 99TH8469)*. IEEE, 236–243.

Received 26 February 2024; revised 30 August 2024; accepted 14 October 2024