SAND2022-12693C

Metrics for Packing Efficiency and Fairness of HPC Cluster Batch Job Scheduling

Alexander V. Goponenko*, Kenneth Lamar*, Christina Peterson*,
Benjamin A. Allan†, Jim M. Brandt†, and Damian Dechev*

* University of Central Florida, Department of Computer Science,
211 Harris Center (Building 116), 4000 Central Florida Boulevard, Orlando, FL 32816
Email: (agoponenko,kenneth,clp8199)@knights.ucf.edu, damian.dechev@ucf.edu
† Sandia National Laboratories, PO Box 5800, MS 0823, Albuquerque, NM 87185
Email: (baallan,brandt)@sandia.gov

***Abstract*—Development of job scheduling algorithms, which directly influence High-Performance Computing (HPC) clusters performance, is hindered because popular scheduling quality metrics, such as Bounded Slowdown, poorly correlate with global scheduling objectives that include job packing efficiency and fairness. This report proposes Area Weighted Response Time, a metric that offers an unbiased representation of job packing efficiency, and presents a class of new metrics, Priority Weighted Specific Response Time, that assess both packing efficiency and fairness of schedules. The provided examples of simulation of scheduling of real workload traces and analysis of the resulting schedules with the help of these metrics and conventional metrics, demonstrate that although values of Bounded Slowdown obtained with a standard First Come First Served backfilling algorithm can be readily improved with modifications of the algorithm and with existing techniques of estimating job runtime, these improvements are accompanied by significant degradation of job packing efficiency and fairness. On the other hand, improving job packing efficiency and fairness over the standard backfilling algorithm, which is designed to target those objectives, is difficult. It requires further development of the algorithms and more accurate runtime estimation techniques that reduce frequency of underpredictions.**

***Index Terms*—high performance computing, parallel job scheduling, performance metrics, schedule quality, runtime estimates, packing efficiency, fairness, weighted flow time, weighted response time**

I. Introduction

HPC clusters remain an important segment of computing infrastructure for their unique capabilities of processing large computationally-intensive problems that require many CPUs to work together on a same job and regularly exchange information. The number of such tasks keeps rising due to sustained interest in machine learning and numerical simulations. The growing HPC industry constantly demands better scheduling algorithms because they directly influence HPC clusters performance. These algorithms are employed to determine the order of processing the jobs, which are continuously submitted by users and remain in a waiting queue until dispatched. The main goal of scheduling is to maximize overall value output without wasting resources. This goal, which grows more important given the rapid increase of the volume of computations and its environmental impact [1], is deemed equivalent to achieving good job "packing efficiency," although exact measurement of "packing efficiency" has not been established yet [2]. Another goal is maintaining fairness. To increase utilization of these expensive resources, managers of HPC clusters often allow jobs from a broad user base, usually for unrelated projects. Therefore, a fair usage, such as ensuring that later requests do not unreasonably delay an earlier request is a matter of social justice [3]. One more goal is to optimize performance perceived by an individual user.

Existing scheduling quality metrics, such as Average Response Time and Slowdown (more details in Section II) assess achievement of the last goal, that is performance perceived by a user. However, because of low awareness of means to measure job packing efficiency and fairness, the ongoing efforts to improve the scheduling poorly address the first two goals, focusing predominately on reducing Slowdown only. As a result, adopting the scheduling innovations is hindered by limited information related to their effects on job packing efficiency and fairness. Further, development also slows down due to difficulty analyzing and comparing alternative solutions.

A. Contributions

In this paper, we propose metrics that allow to evaluate job packing efficiency and fairness, the two important aspects of scheduling that are poorly covered in recent studies. Through practical examples, we demonstrate that these metrics substantially boost capabilities of analyzing and comparing schedules and enable discovering trends not observable otherwise.

In Section III we show that Area-Weighted Response Time, a metric that is virtually unknown, directly corresponds to the overall delay of the computation of the workload, is not influenced by job sizes, and therefore closely matches the idea of job packing efficiency. We also develop in Section

Sandia National Laboratories is a multi-mission laboratory managed and operated by National Technology and Engineering Solutions of Sandia, LLC., a wholly owned subsidiary of Honeywell International, Inc., for the U.S. Department of Energy's National Nuclear Security Administration under contract DE-NA-0003525. SAND #: TODO

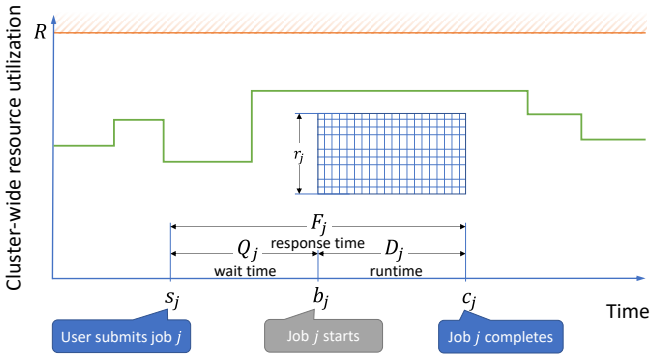


Fig. 1: Diagram of job lifetime showing the notation used herein.

IV a new metric that, while still uninfluenced by job sizes, simultaneously reflects job packing efficiency and the common idea of fairness as giving higher priority to jobs waited longer.

Using these metrics along with conventional metrics, we conduct first all-around analyses of effect of several scheduling improvement approaches on different aspects of scheduling quality. Thus, in Section V we present thorough evaluation of fundamental variants of common scheduling algorithms. This evaluation reveals degradation of the packing efficiency and fairness caused by the variants that improve Slowdown and allows to quantify this trade-off. In Section VI we evaluate effects of job runtime estimations on Slowdown, job packing efficiency, and fairness. In this example, we discover that more accurate estimates and control of underestimation are needed to improve the packing efficiency and fairness. We also outline directions toward achieving it.

In order to clearly demonstrate the use of the proposed metrics and new insights that can be attained with their help, we do not consider HPC clusters with multiple and heterogeneous resources and limit our analysis to only basic popular scheduling techniques. Although creating suitable for practical use scheduling improvements is outside the scope of this work, we uncover trends that offer new ideas and aims for future development of advanced scheduling methods.

II. Background

The scheduling algorithms for HPC clusters must operate in an "online" mode [4]. That is, jobs are being submitted over time and they cannot start before their submit time. The scheduler has no information about jobs arriving in the future. On the other hand, the scheduler can modify its scheduling decisions for the jobs that are not started yet. We will use the following notation in this paper, as described on the diagram that represent events happening to a job (Fig. 1). After a user submits a job j, at time s_j (called submit time or arrival time), the job may end up in a wait queue. After spending Q_i wait time in the queue, the job starts and completes after additional D_j runtime at time c_j. The total time that is passed between the submit time and the completion time is the job's response time, F_j (also called flow time or turnaround time):

$$F_j = Q_j + D_j \ .$$

Users are interested in having the response time as short as possible. For simplicity, we assume in the following discussion that no job runtime depends on how the jobs are scheduled, and hence all D_j values are constant. Therefore, reducing response time F_j can only be achieved by reducing wait time Q_j. We further assume that the cluster contains just one resource, nodes (R nodes in total), and do not consider any other resources that the jobs may require. Each job is assumed to require a constant number of nodes to run, r_j; the user provides this number during the job submission, along with requested runtime limit L_j.

Many metrics of the off-line scheduling, such as Makespan and Utilization, are not useful to evaluate "online" scheduling of jobs when job submit times are predefined [5]. Makespan and Utilization are determined by the completion time of the last job (that is, $\max_j c_j$). These metrics usually poorly correlate with HPC cluster scheduling quality. A good scheduling quality metric, however, should include measurable contribution of every job. Such metric not only can distinguish a wide set of schedules but also may lead to an objective function for schedule optimization. A classical "online" scheduling quality metrics is Average Response Time (*AF*):

$$AF = \frac{1}{n} \sum_{1 \le j \le n} F_j \ ,$$

where n is the total number of jobs. *AF* improves when smaller jobs start first, because this way more jobs can complete earlier. It does not reflect that computing a larger job creates more value. Indeed, consider two users that need to perform same computation but the first user submits a single job that requires 10 nodes while the second user submits 10 separate jobs with same runtime and requiring 1 node each. *AF* indicates that running 10 jobs of the second user before running the only job of the first user is preferable than vice versa, although in terms of created value these options are equal.

A popular metric that often is used to evaluate HPC job scheduling is Average Slowdown. Slowdown is the ratio between response time and runtime. The idea of this metric is that a user would agree that a longer job may spend more time in the queue. Because extremely short jobs overly affect Average Slowdown (a job with zero runtime makes an infinitely large contribution), Average Bounded Slowdown (*BSLD*) was introduced:

$$BSLD = \frac{1}{n} \sum_{1 \le j \le n} \max\left(1, \frac{F_j}{\max\left(D_j, k\right)}\right) ,$$

where k is some lower bound [5]. Introducing k is a crude patch as it produces other adverse effects. While *BSLD* merely reverts to *AF* for the subsets of jobs such that $D_j < k < F_j$, jobs stop contributing to the metric at all as soon as their response time falls below k. Therefore k is often kept small (typically 10 s [6], [7]). Yet, even with reasonable value k,

BSLD is often so strongly influenced by moderate delays of small jobs that it almost completely ignores the quality of scheduling of long and resource-demanding jobs, the very jobs HPC clusters are designed to accommodate. Nevertheless, *BSLD* is widely used for analysis of HPC job scheduling, even though, being predominantly influenced by the delays of small jobs, it almost completely ignores the quality of scheduling of long and resource-demanding jobs, the very jobs HPC clusters are designed to accommodate. As we show in Section V, this metric demonstrates dramatic improvement in response to relatively simple changes of scheduling algorithms that results in preferential scheduling of small jobs, yet it conceals the accompanying degradation of overall job packing efficiency.

III. Metric to evaluate job packing efficiency

To introduce a metric that favors neither small nor large jobs, we consider the following simple "benefit model." Computing a job produces new "knowledge," which becomes available after the completion of the job, c_j. As soon as the knowledge is available, it can start generating "value" proportional to the amount of the knowledge. Therefore, from the utilitarian perspective, the scheduler should strive to minimize the sum of weighted completion times: $\sum w_j c_j$, where w_j is proportional to the amount of the knowledge job j produces. Since $c_j = s_j + F_j$ and the scheduler cannot affect job submit times s_j, optimizing the above-mentioned metric is equal to optimizing Weighted Average Response Time with job weights w_j. Assuming farther that the efficiency of the code of all jobs is similar and all projects are equally important, the amount of the knowledge job j produces is proportional to the effort required to compute the job (which is also called "cost" [8] or "squashed area" [9] and will be called "area" herein), that is $w_j = r_j D_j$. Hence, we obtain a metric that we call Area-weighted Average Response Time (*AWF*):

$$AWF = \frac{\sum r_j D_j F_j}{\sum r_j D_j}.$$

Similarly, we can obtain Area-Weighted Average Wait Time (*AWQ*), which is nothing else but *AWF* minus area-weighted average runtime. These metrics are interchangeable under our assumption that a job runtime does not depend on scheduling decisions. Note that *AWF* could be preferable when this assumption is not valid, although the weights would need to be adjusted. Suitable metrics for such cases are subject of our future research.

Alternatively, we can arrive at *AWF* by analysing a well-known High Density First (HDF) heuristic, used for optimizing weighted response time [10]. In our case of HDF, the density of a job with weight w_j is nothing else but $\frac{w_j}{r_j D_j}$. When $w_j = r_j D_j$, the densities of all jobs equals unity. Therefore, such weights do not encourage any apparent preferential treatment of jobs based on their size. *AWF* has other attractive properties. Thus, the metric reduces if the scheduler manages to start a job earlier while keeping the starting times of the rest of the jobs the same (Fig. 2a). Yet, any jobs reordering that do not change total utilization-over-time profile (Fig. 2b) does not affect the metric. Furthermore, splitting a job into several jobs that have same runtime and together require same amount of resources as the original job (or vice versa) does not affect *AWF* if all jobs start at the same time as the original job started. Splitting a job into several shorter jobs that require same amount of resources and together use same runtime as the original job (or vice versa) also does not affect *AWF* if all jobs start one after another beginning at the original job's start time. Splitting or combining jobs does not affect *AWF* if the total utilization-over-time profile remains the same [8]. *AWF* can only be reduced by moving some computation to earlier time, the amount of change being proportional to the amount of computation and to how far the computation was moved.

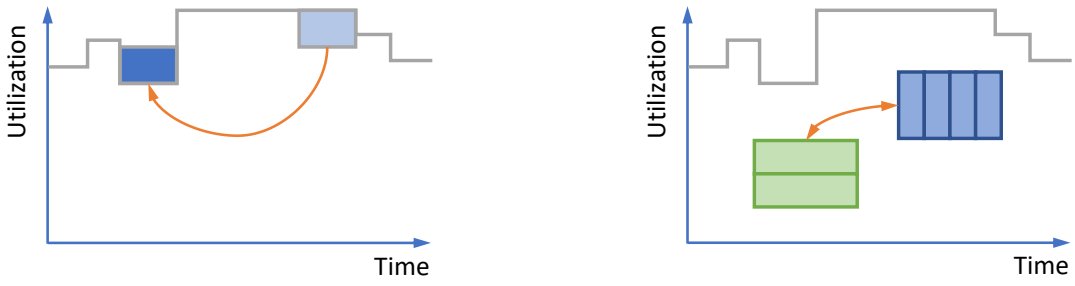
These properties make *AWF* a good unbiased measure of job packing efficiency of the scheduling. Improving packing efficiency is regarded as reducing the amount of resources unusable due to fragmentation and constrains [2]. In HPC clusters and similar system, it equals to minimizing lost resources when work is available, that is, moving as much computation to earlier time as possible. However, since *AWF* is not affected by reordering of jobs, this metric do not measure how well the scheduling complies with jobs priorities.

IV. Metrics that evaluate both packing efficiency and compliance with arrival time priority

When referring to job priorities, we can imply two rather different concepts. First, job priorities may signify difference in benefits of computing the jobs, either due to different code efficiency or because jobs belong to projects of varying importance. To represent such priorities, we can adjust *AWF*'s weights to become $w_j = p_j r_j D_j$, where coefficient p_j signify i-th job's priority of such kind. Such modification of the metric is straightforward and we will not discuss it any further.

The second type of job priorities arises from the common idea of fairness, that is jobs that waited longest should have the highest priority. To construct a corresponding weighted metric, we should increase weights of jobs that waited longer. We, however, prefer to keep the new metric unaffected by the sizes of the jobs, similar to *AWF*. That is, as long as the total utilization-over-time profile remains the same, our metric's value should not change if we split a job into smaller jobs or reorder jobs (Fig. 2b) that were submitted at the same time (for simplicity, we do not consider coefficients p_j). However, moving some computation to earlier time (Fig. 2a), as well as reordering jobs so that longer waiting jobs start earlier, should reduce the metric's value.

To obtain such a metric, we deem jobs to be composed of infinitely small "units of computation" (each requiring dr resources and dt running time), as shown on Fig. 1 and compute the metric as weighted average response time of these units. For instance, job j contributes $\int_{b_j}^{c_j} \int_0^{r_j} w_j(f_t(t))\, dr\, dt = r_j \int_{b_j}^{c_j} w_j(f_t(t))\, dt$ to the weights, where $w_j(f_t(t))$ is the weight of a unit of computation of job j given the unit is computed at time t. This weight may be a function of the response time of the unit $f_j(t) = t - s_j$, but, to keep the metric independent of the job size, it should not otherwise



(a) (b)

Fig. 2: *AWF* measures how well the jobs are packed at earliest possible times. Moving any job to earlier time while keeping the starting times of other jobs the same (a) reduces the metric. Reordering jobs while keeping the utilization profile same (b) does not affect the metric.

depend on the job parameters such as Q_j, F_j, D_j, or r_j. The relation $w_j(f_t(t))$ can be chosen to reflect the cluster policy's target balance between job fairness and packing efficiency. A good baseline is a power function of the response time, that is $w_j(t) = (t - s_j)^\alpha$ [11]. Such weights give rise to a series of metrics which we name Level-α Priority Weighted Specific Response Time ($P^\alpha SF$):

$$P^\alpha SF = \frac{\sum r_j \int_{b_j}^{c_j} (t - s_j)^{\alpha+1}\, dt}{\sum r_j \int_{b_j}^{c_j} (t - s_j)^{\alpha}\, dt} = \frac{\alpha+1}{\alpha+2} \frac{\sum r_j \left(F_j^{\alpha+2} - Q_j^{\alpha+2}\right)}{\sum r_j \left(F_j{}^{\alpha+1} - Q_j^{\alpha+1}\right)}.$$

$P^\alpha SF$ assumes a "benefit model," in which the "knowledge" created by computation starts to produce "value" immediately upon computation. In the "benefit model" used to derive *AWF*, the new "knowledge" becomes available only after completion of the whole job. Although the "benefit model" of $P^\alpha SF$ represents smaller portion of jobs currently deployed to HPC clusters, it is necessary to obtain a metric unaffected by the sizes of the jobs and is well justified by the confidence that an analysis of scheduling alternatives using $P^\alpha SF$ is not misled by any preferential treatment of jobs of particular sizes that some of the schedules may have.

The parameter α controls the ratio between job fairness and packing efficiency contributions to $P^\alpha SF$. Higher α makes the metric more sensitive to the job fairness. The most practical range is $1 \leq \alpha \leq 3$. When $\alpha > 3$, a small number of poorly scheduled jobs (for instance, during a period of high demand) may dominate $P^\alpha SF$, making the metric less responsive to the scheduling decisions related to the other jobs and thus reducing the metric usefulness. On the other hand, $P^0 SF = \frac{\sum r_j D_j (F_j - D_j/2)}{\sum r_j D_j}$ is just the mean of *AWF* and *AWQ* and completely independent of the job fairness.

As the next section demonstrates, $P^\alpha SF$ and *AWF* significantly enhance comparative analysis of the scheduling approaches and reveal pros and cons of the studied alternatives that the common metrics, such as *BSLD*, cannot uncover.

V. Example of Comparing Scheduling Algorithms Using Proposed Metrics

The problem of job scheduling on an HPC cluster belongs to the class of resource-constrained scheduling problems, which is NP-hard [12]. Therefore, in practice HPC job schedulers use approximation algorithms, usually based on a modification of the list scheduling algorithm [13], which is invoked periodically (or upon every job completion or submission). Here, we evaluate some representative examples of published job scheduling algorithms and basic algorithms that are variants of an algorithm shown in Algorithm 1. The pivotal factor that determines the behavior of the algorithm is the order of the waiting jobs in the queue (line 1). The commonplace order is First Come First Served (*FCFS*). The algorithm's behavior also depends on which option is chosen when a job being considered cannot start immediately because of insufficient available resources (line 7).

Algorithm 1 General job scheduler algorithm

```
1:  Arrange waiting jobs in Queue according to some rule.
2:  while Queue is not empty do
3:      j ← Queue.POP()
4:      if job j can start right away then
5:          Start job j.
6:      else
7:          switch Option
8:              case PASSIVE
9:                  Stop this invocation of the scheduling.
10:             case AGGRESSIVE
11:                 Skip job j but keep it in the queue
                        for the next invocation of the scheduling.
12:             case RESERVING
13:                 b_j ← earliest time job j can run.
14:                 Reserve resources r_j
                        for time period (b_j, b_j + L_j).
```

The *passive* option (line 8) effectively pauses the scheduling until some running jobs terminate and release resources

required by the job in the head of the queue. This leads to starting jobs in the original order and guarantees that no job is delayed by later jobs, but can lead to poor performance overall. According to every discussed metric, the *passive* option is inferior to the *reserving* option, so it is not discussed any further.

The *aggressive* scheduling option (line 10) skips jobs that cannot run and schedules jobs that can run instead [14], [15]. This improves resource utilization but delays jobs with high resource requirements and can lead to job starvation.

In case of the *reserving* option (line 12), if a job cannot start immediately, the resources are reserved at the earliest time the job can run (without violating the previous reservations). The reservation makes the resources unavailable (for the jobs later in the queue) for the time period the job is scheduled to run. This way, no job is delayed by later jobs, yet a job can start before a job that is earlier in the queue. This technique is called "backfilling" [16]. The *reserving* option is the most computationally intensive option due to complexity of keeping track of reservations and finding the earliest time a job can run. It also requires runtime estimates for the jobs [17], but in return allows improving resource utilization without compromising compliance with job priorities. Most efficient backfilling is attained if job runtimes D_j are used to make reservations. However, the runtime is not known until the job completes, thus job timelimits L_j are usually used instead. In order to demonstrate efficiency attainable in practice and to estimate best-case scenario, we analyze the performance of algorithms both using L_j and using D_j. We label this algorithm *JustBF*. It resembles the technique called Conservative Backfilling [18] but differs in how it addresses rescheduling when a job completes earlier than expected. Conservative Backfilling determines the earliest time the new job can run when the job is submitted and guarantees that the job is not delayed any further. When a job terminates earlier than expected, Conservative Backfilling "compresses" the schedule making sure that jobs start times do not increase. Our algorithm, which we label *JustBF*, prioritizes the rule that job later in the queue cannot delay any earlier job. Therefore, at each scheduling round, the reservation from the previous round are "forgotten" and all jobs are rescheduled "from scratch."

The most common scheduling algorithm used in practice is EASY Backfilling [16], which, as shown in Algorithm 2, represents a hybrid of *reserving* and *aggressive* options of the more general algorithm presented in Algorithm 1. For the traditional *EASY* algorithm, both *initial order* and *backfill order* (lines 1 and 14) are *FCFS*. It does not guarantee that no job will be delayed by later jobs, but nevertheless guarantees no starvation. EASY Backfilling is widely used because it is more scalable than full backfilling techniques (such as Conservative Backfilling or *JustBF*) as every round of scheduling create no more than one reservation, yet has comparable performance in practice.

Because *BSLD* metric became widely accepted as the standard of measuring the quality of HPC job scheduling, many proposed methods make the EASY Backfilling scheduling

Algorithm 2 General EASY Backfilling algorithm

```
1:  Arrange waiting jobs in Queue according to INITIAL ORDER.
2:  Option ← RESERVING
3:  while Queue is not empty do
4:    j ← Queue.POP()
5:    if job j can start right away then
6:      Start job j.
7:    else
8:      switch Option
9:        case AGGRESSIVE
10:         Skip job j but keep it in the queue
              for the next invocation of the scheduling.
11:       case RESERVING
12:         b_j ← earliest time job j can run.
13:         Reserve resources r_j
              for time period (b_j, b_j + L_j).
14:         Arrange remaining Queue
              according to BACKFILL ORDER.
15:         Option ← AGGRESSIVE
```

favor smaller jobs first. Shortest Job Backfilled First modification of EASY (*EASY-SJBF*) is one of those methods [6], [7], [19]. This modification of the algorithm still employs *FCFS* as *initial order* but uses Shortest Runtime First (*SJF*) order as *backfill order*. Because the first job per a scheduling round is not delayed, this approach still avoids job starvation. To further reduce *BSLD*, EASY Backfilling can be augmented with Smallest Estimated Area (that is, $r_j L_j$) First (*SAF*) job order as *initial order*, producing a scheduling algorithm that we call *SAF-EASY*, which is not starvation-free. To restore no-starvation guarantee, a thresholding mechanism was proposed: a job waiting longer than a pre-set threshold is placed at the head of the queue [20], [21]. However, no-starvation guarantee is a very weak indicator of the scheduling fairness. Without proper metrics, it is difficult to evaluate how such attempts to improve *BSLD* affect other aspects of the cluster performance.

In this section, we compare *EASY*, *EASY-SJBF*, and *SAF-EASY* (with no thresholding) with several scheduling algorithms based on *JustBF* and *Aggressive* algorithms and various job orders (line 1 in Algorithm 1): *SJF*, *SAF*, and Largest Area First (*LAF*). We evaluate the algorithms through simulation of scheduling of traces of real parallel workloads from production systems listed in Table I. We obtained the workload *BW201911* from publicly available December 2019 TORQUE logs of the NCSA Blue Waters supercomputer [22]. The rest of the workload logs are the recommended "cleaned" versions from Parallel Workload Archive [23]. The set of traces spans three decade and different machine sizes. Some traces have been studied in related work [6], [7], [18]. This makes the results obtained in this paper representative of large variety of practical use cases. We evaluate the performance of the algorithms using the following four metrics: *BSLD* (using 10 s for k), *AF*, *AWF*, and P^2SF. Experiments ran on an expanded

TABLE I: Characteristics of the real workload traces used in examples herein.

	# CPU	# Jobs	Utilization	Year	Duration
KTH-SP2	100	28k	70%	1996	11 months
CTC-SP2	338	77k	85%	1996	11 months
SDSC-SP2	128	60k	83%	2000	24 months
SDSC-BLUE	1,152	234k	77%	2003	32 months
CEA-CURIE	93,312	313k	62%	2012	8 months
BW201911	22,636	92k	68%	2019	1 month

version of Predictsim [7], a parallel job scheduler simulator. The source code and more details related to our simulation conditions are available at GitHub[1].

The results are summarized in Table II, which shows metrics changes in percent relative to the values obtained with regular *JustBF*. The left part of the table contains results of applying algorithms using actual job runtimes D_j, while the right part corresponds to results of using user-provided timelimits L_j. Scheduling using actual runtimes reveals possible performance of the algorithms when they have access to accurate information. In this case, *SAF-JustBF* demonstrates best improvement of *BSLD*. Possibly, *SAF* heuristic is better than *SJF* for the studied workloads because it focuses more on long-term effects of the scheduling decisions. *SAF-Aggressive* and *SJF-Aggressive* result in inferior values of *BSLD* than *SAF-JustBF* and *SJF-JustBF*, which may be surprising because one can expect little opportunities of backfilling when smallest jobs are already scheduled first.

According to *AWF*, *LAF-JustBF* is able to attain better job packing efficiency than regular *JustBF*. *LAF* improve packing efficiency because it gives larger jobs more chances to find resources and then fill the remaining "holes" with smaller jobs. When jobs are scheduled in *FCFS* order, less small jobs remains to fill the "holes" and the scheduling becomes more fragmented. *AWF* of *LAF-Aggressive* is nevertheless higher than of regular *JustBF*. Likely, even though *LAF-Aggressive* scheduler gives larger jobs a chance to start first, it still lets the small jobs jump in front of the large jobs and end up with few small jobs when needed. No surprise that scheduling using *SAF* or *SJF* results in less efficient job packing and significantly hurts *AWF*. Scheduling of *SDSC-SP2* trace exemplifies this. Fig. 3 (top) displays cumulative fraction of the number of jobs vs job area (similar to empirical cumulative distribution function): $F(x) = \sum_{j:\, r_j D_j \le x} 1/n$ and cumulative fraction of job areas vs job area: $\overline{F}(x) = \sum_{j:\, r_j D_j \le x} r_j D_j / \sum_{1 \le j \le n} r_j D_j$. Note that smallest 80% jobs of this trace contributes less than 5% total job area (which corresponds to the amount of computation), while largest 5% jobs are responsible for more than 50% of the computation. Fig. 3 (bottom) shows cumulative wait area $CWA(x) = \sum_{j:\, r_j D_j \le x} Q_j r_j$ attained on this workload trace using various schedulers. While *JustBF* keeps *CWA* roughly proportional to cumulative job area, *SAF-JustBF* depletes the wait queue of small jobs and nearly starves the large jobs.

[1] https://github.com/algo74/predictsim/tree/SBACPAD2022

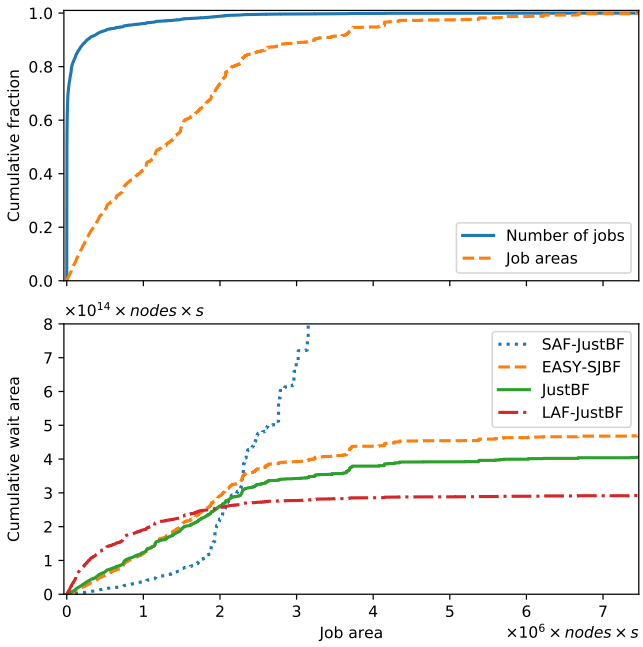


Fig. 3: Cumulative fraction of number of jobs and job areas as functions of job area in *SDSC-SP2* (top) and cumulative wait areas as functions of job area attained on this workload trace with *SAF-JustBF*, *EASY-SJBF*, *JustBF*, and *LAF-JustBF* using actual job runtimes (bottom).

LAF-JustBF increases wait time of smaller jobs, yet they do not starve. Their wait time per job area remains smaller (on average) than for the larger jobs while the overall productivity of the cluster increases.

Because of possibility of diminished packing efficiency, one should be cautious giving higher priority to shorter jobs even though it could demonstrate improvement of *BSLD*. Thus, in case of *SDSC-SP2* trace, the job packing efficiency (measured by *AWF*) of *SAF-JustBF* and *SJF-JustBF* algorithms is so poor that these algorithms are worse than regular *JustBF* even in terms of *AF*, a metric that, while still favoring scheduling smaller jobs first, is more sensitive to job packing efficiency than *BSLD*.

While *LAF-JustBF* can slightly improve *AWF* over *JustBF*, none of the considered algorithms can attain better value of P^2SF than *JustBF*. Indeed, *JustBF* is designed to attain good packing efficiency while complying with job priorities, and $P^\alpha SF$ is measuring just that.

According to the metrics, EASY Backfilling is a good practical approximation of full backfilling used in *JustBF* algorithms. *SAF-EASY* demonstrates values of *BSLD* and *AF* that are close to *SAF-JustBF*, while regular *EASY* is close to regular *JustBF* in terms of *AWF* and P^2SF. Of all algorithms considered, only *EASY-SJBF* may be a practical solution for improving *BSLD* while maintaining some fairness of the scheduling. Compared to *SAF-JustBF* in our experiments, *EASY-SJBF* leads to not as large but still significant decrease of

TABLE II: *BSLD*, *AF*, *AWF*, and P^2SF of scheduling algorithms as percent changes relative to *JustBF*. Lower is better.

BSLD

	Using actual runtime						Using requested timelimit					
	KTH-SP2	*CTC-SP2*	*SDSC-SP2*	*SDSC-BLUE*	*CEA-CURIE*	*BW201911*	*KTH-SP2*	*CTC-SP2*	*SDSC-SP2*	*SDSC-BLUE*	*CEA-CURIE*	*BW201911*
LAF-JustBF	117%	379%	366%	838%	600%	1175%	35%	229%	88%	305%	196%	154%
LAF-Aggressive	111%	258%	288%	135%	197%	146%	24%	33%	74%	32%	-79%	-87%
JustBF	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%
EASY	7%	29%	5%	-23%	25%	77%	-9%	-24%	-14%	-31%	-81%	-41%
EASY-SJBF	-26%	-39%	-15%	-67%	-55%	-47%	-32%	-53%	-19%	-64%	-83%	-60%
SJF-Aggressive	-34%	-58%	-51%	-30%	-70%	90%	-48%	-74%	-33%	-50%	-94%	-90%
SAF-Aggressive	-31%	-54%	-44%	-58%	-69%	90%	-41%	-74%	-33%	-49%	-94%	-81%
SJF-JustBF	-67%	-83%	-73%	-85%	-77%	-91%	-56%	-57%	-23%	-57%	-72%	-67%
SAF-JustBF	-69%	-83%	-77%	-88%	-84%	-92%	-56%	-42%	9%	-3%	-93%	-76%
SAF-EASY	-62%	-82%	-76%	-85%	-82%	-92%	-62%	-45%	-1%	-38%	-88%	-76%

AF

	Using actual runtime						Using requested timelimit					
	KTH-SP2	*CTC-SP2*	*SDSC-SP2*	*SDSC-BLUE*	*CEA-CURIE*	*BW201911*	*KTH-SP2*	*CTC-SP2*	*SDSC-SP2*	*SDSC-BLUE*	*CEA-CURIE*	*BW201911*
LAF-JustBF	26%	132%	139%	298%	63%	198%	20%	146%	59%	141%	48%	120%
LAF-Aggressive	10%	11%	124%	37%	-6%	-36%	0%	-12%	38%	17%	-21%	-51%
JustBF	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%
EASY	-4%	-10%	-4%	-13%	-7%	-13%	-7%	-24%	-12%	-15%	-20%	-23%
EASY-SJBF	-10%	-23%	-8%	-30%	-14%	-33%	-12%	-30%	-12%	-30%	-23%	-28%
SJF-Aggressive	-13%	-36%	32%	-22%	-20%	-47%	-16%	-45%	49%	-26%	-28%	-54%
SAF-Aggressive	-13%	-36%	49%	-32%	-20%	-48%	-14%	-45%	57%	-25%	-28%	-54%
SJF-JustBF	-18%	-23%	14%	-44%	-17%	-38%	-19%	-33%	48%	-31%	-19%	-26%
SAF-JustBF	-17%	-33%	72%	-38%	-21%	-47%	-6%	-14%	140%	-11%	-28%	-53%
SAF-EASY	-16%	-32%	59%	-40%	-21%	-47%	-14%	-18%	126%	-23%	-24%	-54%

AWF

	Using actual runtime						Using requested timelimit					
	KTH-SP2	*CTC-SP2*	*SDSC-SP2*	*SDSC-BLUE*	*CEA-CURIE*	*BW201911*	*KTH-SP2*	*CTC-SP2*	*SDSC-SP2*	*SDSC-BLUE*	*CEA-CURIE*	*BW201911*
LAF-JustBF	-6%	-6%	-17%	-8%	0%	-2%	-6%	-6%	-25%	-9%	-2%	-1%
LAF-Aggressive	7%	23%	16%	16%	3%	4%	5%	-1%	-13%	16%	1%	-1%
JustBF	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%
EASY	1%	1%	0%	0%	0%	0%	1%	-4%	-6%	0%	-1%	2%
EASY-SJBF	1%	6%	10%	3%	0%	1%	1%	5%	4%	3%	-1%	1%
SJF-Aggressive	25%	211%	430%	63%	3%	5%	21%	105%	349%	58%	1%	0%
SAF-Aggressive	25%	199%	443%	63%	3%	5%	25%	104%	385%	59%	1%	0%
SJF-JustBF	12%	120%	206%	23%	2%	29%	17%	152%	487%	66%	3%	36%
SAF-JustBF	68%	632%	995%	269%	20%	17%	194%	526%	992%	410%	3%	9%
SAF-EASY	59%	671%	888%	233%	7%	17%	104%	485%	945%	333%	60%	10%

P^2SF

	Using actual runtime						Using requested timelimit					
	KTH-SP2	*CTC-SP2*	*SDSC-SP2*	*SDSC-BLUE*	*CEA-CURIE*	*BW201911*	*KTH-SP2*	*CTC-SP2*	*SDSC-SP2*	*SDSC-BLUE*	*CEA-CURIE*	*BW201911*
LAF-JustBF	61%	591%	802%	1224%	815%	16%	27%	514%	542%	930%	75%	16%
LAF-Aggressive	406%	2149%	2094%	2517%	777%	55%	191%	921%	1436%	2645%	700%	44%
JustBF	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%
EASY	4%	6%	9%	4%	1%	1%	6%	0%	-1%	10%	2%	3%
EASY-SJBF	5%	23%	22%	19%	5%	5%	10%	19%	16%	27%	6%	3%
SJF-Aggressive	844%	10111%	11216%	5735%	287%	77%	697%	3481%	9812%	5356%	374%	64%
SAF-Aggressive	894%	8045%	11744%	6846%	315%	77%	878%	3479%	9484%	5348%	372%	64%
SJF-JustBF	102%	3060%	9315%	329%	37%	114%	229%	5226%	13113%	1222%	114%	110%
SAF-JustBF	574%	18841%	16357%	7915%	626%	108%	2946%	12090%	13291%	9456%	109%	92%
SAF-EASY	481%	18570%	16009%	10473%	222%	109%	1776%	12471%	12972%	7559%	3029%	92%

BSLD along with less severe degradation of *AWF* and P^2SF.

Hence, the demonstrated analysis with the help of the proposed metrics allows to estimate such trade-offs and advise scheduling changes depending on the cluster policy. When timelimits are used to estimate runtimes (right part of Table II), any benefits of *JustBF* over *EASY* are insignificant. The other tendencies described above also become less profound, as lower quality input data reduce the algorithms' ability to attain their goals. The relative values in Table II, however, do not show that the metrics predominantly degrade when timelimits are used instead of runtimes. The next section discusses this problem in more details.

VI. Example of comparing job runtime estimating techniques

Backfilling techniques (as well as reordering jobs in some algorithms) require job runtimes D_j, which are not known until the jobs finish. In practice, algorithms use timelimits L_j that are provided by the users. This leads to less efficient backfilling and reduced scheduling quality. The rows labeled *Runtime* in Table III demonstrate improvements that could be attained if the scheduling algorithms use actual job runtimes instead of timelimits. Therefore, estimating job runtimes for improving the efficiency of scheduling algorithms is a popular research topic.

However, more accurate estimates of job runtimes not necessarily improve every metric of every algorithm. For example, Fig. 4 shows the values of the metrics of scheduling

TABLE III: Effect of using various job runtime predictions in scheduling algorithms demonstrated through the change of the metrics the algorithms target. Metrics changes are shown as percent changes relative to values obtained using users provided timelimits. Lower is better. *Runtime* demonstrates improvements that could be attained with exact predictions.

(a) *BSLD* of *EASY-SJBF*

	KTH-SP2	*CTC-SP2*	*SDSC-SP2*	*SDSC-BLUE*	*CEA-CURIE*	*BW201911*
Runtime	-28.2%	-41.6%	-31.5%	-32.9%	-63.2%	-71.3%
Last2	-7.5%	-19.0%	-21.3%	-4.1%	-58.5%	-68.6%
CVH	-17.4%	-41.4%	-1.9%	108.6%	-50.7%	-69.8%
Hierarchy	-12.2%	4.3%	-18.4%	-19.8%	-42.3%	-68.7%

(b) *BSLD* of *SAF-JustBF*

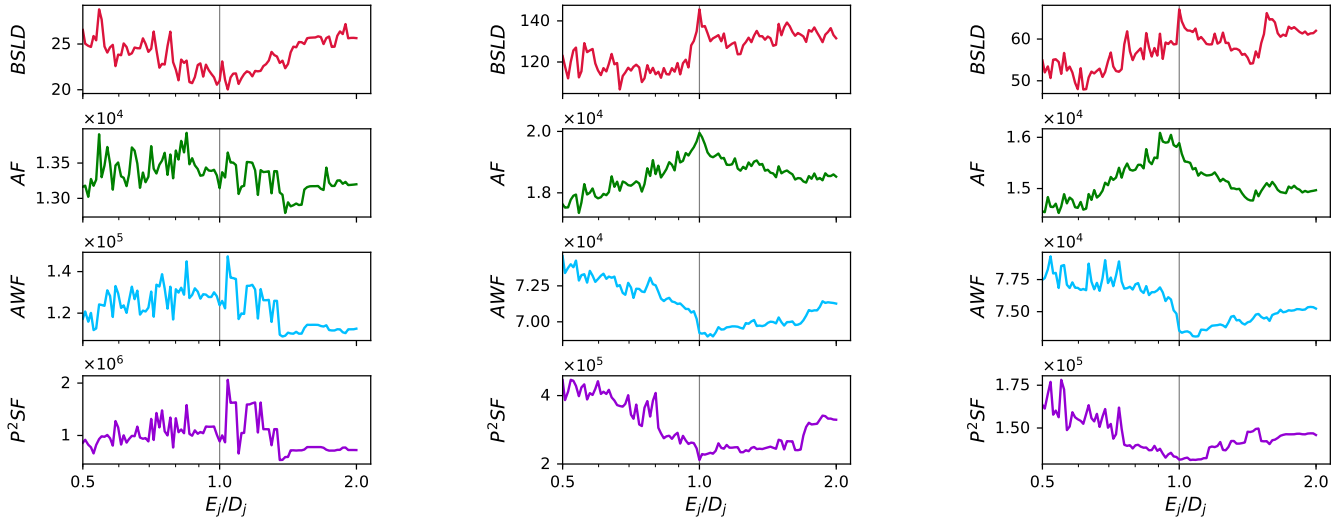
	KTH-SP2	*CTC-SP2*	*SDSC-SP2*	*SDSC-BLUE*	*CEA-CURIE*	*BW201911*
Runtime	-53.5%	-87.3%	-86.2%	-91.1%	-66.7%	-92.5%
Last2	17.2%	-30.1%	-44.4%	-75.4%	-0.8%	-85.9%
CVH	0.4%	-58.1%	-39.8%	-17.5%	-20.7%	-64.6%
Hierarchy	-13.3%	-24.9%	-58.0%	-79.4%	1.2%	-77.5%

(c) *AWF* of *LAF-JustBF*

	KTH-SP2	*CTC-SP2*	*SDSC-SP2*	*SDSC-BLUE*	*CEA-CURIE*	*BW201911*
Runtime	-1.3%	-17.1%	-8.9%	-5.2%	-0.1%	-4.9%
Last2	8.2%	19.0%	17.2%	6.1%	0.8%	15.1%
CVH	14.4%	25.9%	41.4%	24.2%	-0.1%	0.4%
Hierarchy	4.4%	10.1%	18.1%	11.3%	1.5%	15.5%
SP-2%	-0.2%	-1.0%	0.4%	-0.8%	5.5%	-0.4%
SP-3%	-0.2%	-0.9%	1.3%	-1.2%	3.3%	-0.6%
SP-4%	-0.2%	0.2%	1.0%	-0.7%	2.5%	0.8%

(d) P^2SF of *JustBF*

	KTH-SP2	*CTC-SP2*	*SDSC-SP2*	*SDSC-BLUE*	*CEA-CURIE*	*BW201911*
Runtime	-0.6%	-25.4%	-19.6%	-7.3%	-3.3%	-7.3%
Last2	17.9%	439.4%	9.5%	107.3%	15.2%	32.2%
CVH	355.6%	817.4%	1837.6%	394.5%	111.3%	43.8%
Hierarchy	21.0%	402.9%	8.3%	41.5%	111.1%	19.7%
SP-2%	0.6%	0.0%	0.0%	1.1%	-1.2%	-0.5%
SP-3%	0.0%	-0.2%	1.3%	3.1%	-0.7%	-0.4%
SP-4%	0.0%	-3.5%	0.7%	5.8%	-0.5%	-0.7%



(a) *SAF-JustBF* (b) *LAF-JustBF* (c) *JustBF*

Fig. 4: Effect of runtime estimate distortion on metrics when scheduling *KTH-SP2* trace using various algorithms. Lower metric value is better. The estimates E_j are distorted by multiplying the actual runtimes D_j by a factor k in the range from 0.5 to 2, that is $E_j = k \times D_j$.

KTH-SP2 trace using as the job runtime estimates the actual runtimes multiplied by a factor k in the range from 0.5 to 2, that is $E_j = k \times D_j$, where E_j is the estimate used for the scheduling. The accurate runtimes in case of *SAF-JustBF* minimize *BSLD*, but not the other three metrics (Fig. 4a), suggesting that *SAF-JustBF* is not working towards improving these three metrics. On the contrary, *LAF-JustBF* and *JustBF* are not fit for *BSLD* or *AF* and they do not attain better values for these metrics with accurate runtime predictions (Fig. 4b and 4c). Thus, if an algorithm is not suited for a metric, more accurate estimates may fail to improve the metric. An opposite is possible: deliberately inaccurate predictions may attain better values of *BSLD* because it leads to more *SJF*-like schedule [17], albeit a more suitable algorithm with accurate predictions should easily outperform such unsound approach. Meanwhile, if an algorithm achieves its own best value of a metric using actual runtimes, it still may be an inappropriate algorithm to optimize that metric. For instance, P^2SF for *LAF-JustBF* is smallest when $E_j = D_j$ (the bottom plot of Fig. 4b), yet *JustBF* is much better algorithm for P^2SF (Table II).

In this example, we analyze different job runtime prediction methods using the four metrics. The first prediction method, which we label *Last2*, estimates the runtime as the average runtime of the last two jobs of the same user [6]. This simple approach is actually a result of wide hyper-parameter tuning that aimed at improving *BSLD* of scheduling traces *KTH-SP2*, *CTC-SP2*, *SDSC-SP2* and *SDSC-BLUE* [6] with *EASY-SJBF*.

The second predictor, labeled *CVH*, is also a result of hyper-parameter tuning that targeted improving of *BSLD* of scheduling with *EASY-SJBF* the traces that *Last2* used (plus *CEA-CURIE*), but incorporated a more elaborate machine learning model and employed cross-validation to select best performing set of heuristics [7].

The third prediction method, labeled *Hierarchy*, predicts job runtime as an exponentially decaying weighted average of most similar historical jobs [24]. It implements hierarchy of groups, from more specific to more general, based on parameters know at job submission (job name, user id, timelimit, and required number of nodes) in a manner similar to Gibbons [25] and Smith et al. [26], and looks through the hierarchy until finding a non-empty group. A finished job causes prediction update of all groups to which the job belongs. The order of the groups in the hierarchy, as well as the decaying parameters, were selected using an additional dataset [27] and were not tuned to the data analyzed here.

According to the values of the coefficient of determination R^2 (Table IV), *Hierarchy* is the most accurate predictor among considered, followed by *Last2*. *CVH* is the least accurate of the three. The machine learning model used in *CVH* should be capable of more accurate predictions, but *CVH* was tuned to reduce *BSLD* of *EASY-SJBF* scheduling, not to attain the highest accuracy.

Since *AWF* and P^2SF of *LAF-JustBF* and *JustBF* degrade from underprediction more than from overprediction (Fig. 4b and 4c), we added one more prediction technique to the analysis of these metrics. This method estimates job runtime as a shortest time such that the probability of surviving longer than it for the historical jobs with same job name, user id, timelimit, and required number of nodes is below a given threshold [24]. The survival probability is calculated using non-parametric Kaplan-Meier method [28], which we modified with exponentially decaying weights [24]. We consider survival probability of 2, 3, and 4%, which we label *SP-2%*, *SP-3%*, and *SP-4%* correspondingly.

Table IIIa shows how the use of the predictors improves *BSLD* of *EASY-SJBF*, the algorithm for which *Last2* and *CVH* are optimized. Although the prediction methods have different complexity and prediction accuracy, all three predictors help improve *BSLD* in most experiments. The high variability in the performance of the predictors does not allow to identify the best predictor. When the three predictors are used with *SAF-JustBF* (Table IIIb), *BSLD* also predominantly improves and the high variability again does not allow to identify the best predictor. Nevertheless, in this case *Hierarchy* may be a little better, possibly because *Last2* and *CVH* are optimized for a different algorithm and have less accuracy in general.

These three predictors, however, have a detrimental effects on *AWF* when used in *LAF-JustBF* (Table IIIc) and on P^2SF when used in *JustBF* (Table IIId). Although the high variation in the results remains, *CVH* leads to noticeably worse outcomes for both cases. Poor performance of these three predictors here is likely due to frequent underpredictions. Degradation of P^2SF is not surprising as underpredictions can lead to violation of job priorities. Yet, it remains intriguing that underprediction may hurt job packing efficiency more than overprediction. Predictors *SP-2%*, *SP-3%*, and *SP-4%* (which are designed to reduce underprediction) performed better than the other predictors in both cases. They often lead to better values of the metrics than the scheduling performed using timelimits, but high variability of the results does not allow to conclude whether this prediction approach can improve over scheduling with timelimits.

This analysis demonstrates that *BSLD* can be readily improved not only with algorithms that give priorities to smaller jobs (Section V), but also by using job runtime estimates obtained by a variety of prediction techniques. On the other hand, improving *AWF* and P^2SF is not trivial. Not only little improvement over *JustBF* was attained by any algorithm in Section V, no prediction techniques in this section allow noticeable improvement either. Thus, the current state of the art in estimating job runtime is not adequate for practical application in job scheduling.

VII. Related Work

AWF was used in analysis of preemptive gang scheduling [8], [29] and to demonstrate benefits of combining HPC clusters in a grid [9]. This metric was also suggested as an objective function when high system load is desired [13]. Yet, *AWF* remains virtually unknown. It was not used in the rest of the publications covered in this Section. Another metric suggested to measure job packing is Loss of Capacity (*LOC*) [30]. This metric corresponds to the fraction of processor cycles that were idle when waiting jobs could have use them:

$$LOC = \frac{\int_{\min_{1\le j\le n} s_j}^{\max_{1\le j\le n} c_j} \min\left(\sum_{j:\, s_j<t<b_j} r_j \,,\; R - \sum_{j:\, b_j<t<c_j} r_j \right) dt}{\left(\max_{1\le j\le n} (c_j) - \min_{1\le j\le n} (s_j) \right) \times R},$$

where R is the total number of nodes in the cluster. *LOC* is harder to compute than *AWF*, but it is also inferior in other ways. During the periods when the waiting queue is long, *LOC* do not change if a job is moved to another time (Fig. 2a). When the queue is short, *LOC* may favor scheduling shorter jobs first in a situation shown on Fig. 2b. Thus, *LOC* is a less reliable metric of job packing efficiency than *AWF*.

Several approaches were suggested to measure fairness and compliance with priority policy of scheduling. Thus, Jain et al. suggested various fairness indices, which are not intended for use as metrics [31].

Maximum wait time [32] and similar trivial metrics have little use because they indicate quality of scheduling of a single job (or a small sub-set of the jobs) and disregards all other scheduling decisions. Pruhs et al. suggested l_p norm of response time, that is $(\sum {F_j}^p)^{1/p}$, to measure job fairness [4]. This metric is somewhat similar to $P^\alpha SF$ but may encourage preferential treatment of jobs on the basis of their size. The weights of $P^\alpha SF$, in contract, are designed to prevent such

TABLE IV: Coefficient of determination R^2 of the job walltime estimations for the discussed scheduling algorithms. Higher is better.

(a) *BSLD* of *EASY-SJBF*

	KTH-SP2	*CTC-SP2*	*SDSC-SP2*	*SDSC-BLUE*	*CEA-CURIE*	*BW201911*
Last2	0.33	0.36	0.52	0.56	0.30	0.06
CVH	-0.05	-0.22	-0.07	0.04	0.23	-0.07
Hierarchy	0.64	0.51	0.64	0.68	0.50	0.05
timelimit	0.59	-0.58	-0.34	-0.06	-7.08	-8.02

(b) *BSLD* of *SAF-JustBF*

	KTH-SP2	*CTC-SP2*	*SDSC-SP2*	*SDSC-BLUE*	*CEA-CURIE*	*BW201911*
Last2	0.33	0.36	0.52	0.55	0.30	0.01
CVH	-0.02	-0.22	-0.13	-0.03	0.22	-0.09
Hierarchy	0.64	0.50	0.63	0.68	0.50	0.18
timelimit	0.59	-0.58	-0.34	-0.06	-7.08	-8.02

(c) *AWF* of *LAF-JustBF*

	KTH-SP2	*CTC-SP2*	*SDSC-SP2*	*SDSC-BLUE*	*CEA-CURIE*	*BW201911*
Last2	0.32	0.35	0.50	0.55	0.29	0.04
CVH	-0.02	-0.20	-0.18	-0.03	0.26	-0.17
Hierarchy	0.64	0.51	0.64	0.69	0.48	0.10
SP-2%	0.60	-0.44	-0.33	0.06	-1.97	-4.63
SP-3%	0.60	-0.41	-0.33	0.07	-1.85	-4.60
SP-4%	0.60	-0.40	-0.32	0.08	-1.73	-4.57
timelimit	0.59	-0.58	-0.34	-0.06	-7.08	-8.02

(d) *P^2SF* of *JustBF*

	KTH-SP2	*CTC-SP2*	*SDSC-SP2*	*SDSC-BLUE*	*CEA-CURIE*	*BW201911*
Last2	0.33	0.35	0.50	0.55	0.28	0.03
CVH	-0.02	-0.22	-0.21	-0.01	0.23	-0.17
Hierarchy	0.64	0.50	0.63	0.68	0.49	0.05
SP-2%	0.60	-0.43	-0.33	0.06	-1.84	-4.68
SP-3%	0.60	-0.40	-0.33	0.07	-1.80	-4.66
SP-4%	0.60	-0.38	-0.32	0.08	-1.74	-4.63
timelimit	0.59	-0.58	-0.34	-0.06	-7.08	-8.02

interference with the measurement of job packing efficiency and fairness. A popular approach to measure fairness of scheduling is based on job Fair Start Times (*FST*), which can be calculated, either by simulating a "fair" schedule, or by simulating scheduling as if no later jobs arrived, or both [21], [30], [33]. Such metrics require simulation of the scheduling, which, may be computationally intensive (as in case of *JustBF*) and may be tainted by the use of a "gold standard" schedule [30]. The "gold standard" schedule must be not only ideally fair but also very efficient since the metric becomes zero for any schedule that improves start times of all jobs over the "gold standard." Comparing job start times to *FST* obtained with the same schedule as if no later jobs arrived may uncover interesting aspects of the schedule's inner consistency with fairness in strong "social justice" sense. It is somewhat "orthogonal" to packing efficiency, as in terms of such a metric one schedule can be inferior to another even if it improves the start time of every job. $P^{\alpha}SF$, in contrast, combines fairness and packing efficiency into a single scalar metric by assigning priorities to jobs, based not on "who came first" but on "who waited longer." $P^{\alpha}SF$ also makes better objective functions for optimization techniques.

Due to deficiency of suitable metrics to measure job packing and fairness and insufficient awareness of these aspects of the scheduling quality, the majority of publications that present techniques for improvement of job scheduling do not consider any metrics beyond *BSLD* and *AF*. However, as we show in Section V, improvements of *BSLD* and *AF* are relatively easy to achieve with SJF-like scheduling but likely are accompanied by degradation of packing efficiency and fairness. Some publications introduce ad-hoc techniques to evaluate fairness of scheduling, such as measuring Average Slowdown separately for jobs of various percentile [32] or categorizing workload into classes ("long" and "short," "narrow" and "wide," etc.) and calculating performance for the classes separately [21], [34]. Results of such ad-hoc techniques are hard to interpret and compare.

Our examples in Sections V and VI are basic demonstrations of using the proposed metrics to evaluate improving of HPC job scheduling. We do not consider all proposed sort orders, including those designed to balance *BSLD* and fairness [32], [35], and do not analyze relaxed backfill strategies [11], heuristic-based optimization [36], and more advanced scheduling optimization such as constraint programming scheduling [37], genetic algorithms [38], and simulated annealing technique [39]. We also do not evaluate all job runtime prediction approaches, including neural networks model [40], k-nearest neighbors method [41], two-stage approach involving radial basis function/Bayesian classifier [42], and Predicting Query Runtime Regression [43]. Comprehensive review and rigorous evaluation of the state-of-the-art techniques are outside of the scope of this paper. Besides, many published techniques are tuned solely to improve *BSLD*. It may be unproductive to evaluate the techniques' potential in their present state, as they performance may improve if they are tuned with the new metrics in mind.

An interesting research direction consists of improving cluster utilization by stimulating users to submit more jobs [44]–[47]. These activities reveal important findings that, depending on the users' behavior, reducing *BSLD* may lead to more submitted jobs and thus higher cluster utilization. Although these findings seem to undermine the need to improve job packing efficiency and fairness, both approaches have their place. Stimulating users to submit more jobs is important when the cluster is underutilized while improving job packing efficiency and fairness become crucial to retain users once the cluster is popular.

The higher negative effect of job runtime underestimation than of the overestimation that we report in Section VI, to the best of our knowledge, has not been observed before. Such phenomenon was suggested, but not confirmed by Galleguillos et al. [37]. In the rest of the cases, the problem of runtime underestimation is associated with premature cancellation of jobs. Thus, Fan et al. [48] employed job run time predictions using a Tobit model to reduce the underestimation because in their simulations jobs were cancelled as soon as they

reached their estimated runtime. We, similar to Tsafrir et al. [6] and others, do not cancel jobs until they reach their requested timelimits. Therefore, the negative effect of the underestimation we report is not related to premature job cancellation.

VIII. Conclusions

Both metrics *AWF* and $P^{\alpha}SF$ neither penalize nor encourage any preferential scheduling based on job size. This and other properties make *AWF* an accurate indicator of job packing efficiency. The class of $P^{\alpha}SF$ metrics measure various degrees of balance between packing efficiency and fairness of schedules.

Our proof-of-concept examples of evaluation of scheduling (Sections V and VI) demonstrate that the proposed metrics are highly advantageous for comparing alternative schedules because they allow to evaluate trade-offs between improving *BSLD* or *AF* on one hand and preserving job packing efficiency and fairness on the other hand.

The value of *BSLD* attainable by standard backfilling algorithms, such as *JustBF*, readily improves with relatively straightforward modifications of the algorithms and can be further improved with existing job runtime prediction techniques. However, algorithms that improve *BSLD* have high risk of degrading both job packing efficiency and fairness, the two highly important aspects of job scheduling in HPC clusters.

On the other hand, *AWF* and P^2SF are notoriously difficult to improve with either modification of the scheduling algorithm or with ordinary runtime estimates. It does not mean that these metrics are not responsive; their values for *SJF*-like schedules are significantly lower than for *FCFS* backfilling schedules. Improving these metrics is difficult because *JustBF* (our baseline algorithm) is already designed to attain good job packing and high standard of fairness. Finding an algorithm that improve over it is difficult. Using job runtime estimates obtained with existing prediction techniques instead of user-provided timelimits also cannot improve *AWF* and $P^{\alpha}SF$, possibly due to low accuracy, as well as high frequency of underprediction, which degrades the metrics stronger than overprediction. Our custom-made predictor, which limits the frequency of underpredictions, shows performance similar to the scheduling with timelimit. Therefore, current prediction techniques are not mature enough to be used in practice and further development is needed with emphasis on accuracy and proper balance between underprediction and overprediction.

For the reasons presented herein, we argue that *AWF* and $P^{\alpha}SF$ must be key components of HPC job scheduling researchers' toolbox. This metrics allow characterizing multiple aspects of scheduling quality during analysis. They are also useful as targets for future algorithm development or as prototypes of objective functions for optimization techniques.

Acknowledgment

The authors thank Benjamin Schwaller and Omar Aaziz for their valuable discussions. The works at the University of Central Florida were supported by contracts with Sandia National Laboratories. The authors acknowledge the University of Central Florida Advanced Research Computing Center for providing computational resources.

References

[1] N. Ensmenger, "The Environmental History of Computing," *Technology and Culture*, vol. 59, no. 4, pp. S7–S33, 2018.

[2] A. Verma, M. Korupolu, and J. Wilkes, "Evaluating job packing in warehouse-scale computing," in *2014 IEEE Int. Conf. CLUSTER*. Madrid, Spain: IEEE, Sep. 2014, pp. 48–56.

[3] R. C. Larson, "Perspectives on Queues: Social Justice and the Psychology of Queueing," *Oper. Res.*, vol. 35, no. 6, pp. 895–905, 1987.

[4] K. Pruhs, J. Sgall, and E. Torng, "Online scheduling," in *Handbook of Scheduling: Algorithms, Models, and Performance Analysis*. Boca Raton, FL, USA: Chapman & Hall/CRC, 2004.

[5] D. G. Feitelson and L. Rudolph, "Metrics and benchmarking for parallel job scheduling," in *Job Scheduling Strategies for Parallel Processing*, ser. Lecture Notes in Computer Science, D. G. Feitelson and L. Rudolph, Eds. Berlin, Heidelberg: Springer, 1998, pp. 1–24.

[6] D. Tsafrir, Y. Etsion, and D. G. Feitelson, "Backfilling Using System-Generated Predictions Rather than User Runtime Estimates," *IEEE Trans. Parallel Distrib. Syst.*, vol. 18, no. 6, pp. 789–803, Jun. 2007.

[7] E. Gaussier, D. Glesser, V. Reis, and D. Trystram, "Improving backfilling by using machine learning to predict running times," in *SC '15: Proc. Int. Conf. High Perform. Comput., Netw., Storage and Anal.* Austin, TX, USA: IEEE, Nov. 2015, pp. 1–10.

[8] U. Schwiegeishohn and R. Yahyapour, "Improving first-come-first-serve job scheduling by gang scheduling," in *Job Scheduling Strategies for Parallel Processing*, ser. Lecture Notes in Computer Science, D. G. Feitelson and L. Rudolph, Eds. Berlin, Heidelberg: Springer, 1998, pp. 180–198.

[9] C. Ernemann, V. Hamscher, and R. Yahyapour, "Benefits of global grid computing for job scheduling," in *5th IEEE/ACM Int. Workshop on Grid Comput.* Pittsburgh, PA, USA: IEEE, Nov. 2004, pp. 374–379.

[10] L. Becchetti, S. Leonardi, A. Marchetti-Spaccamela, and K. R. Pruhs, "Online Weighted Flow Time and Deadline Scheduling," in *Approximation, Randomization, and Combinatorial Optimization: Algorithms and Techniques*, ser. Lecture Notes in Computer Science, M. Goemans, K. Jansen, J. D. P. Rolim, and L. Trevisan, Eds. Berlin, Heidelberg: Springer, 2001, pp. 36–47.

[11] W. A. Ward, C. L. Mahood, and J. E. West, "Scheduling Jobs on Parallel Systems Using a Relaxed Backfill Strategy," in *Job Scheduling Strategies for Parallel Processing*, ser. Lecture Notes in Computer Science, D. G. Feitelson, L. Rudolph, and U. Schwiegelshohn, Eds. Berlin, Heidelberg: Springer, 2002, pp. 88–102.

[12] J. Blazewicz, J. K. Lenstra, and A. H. G. R. Kan, "Scheduling subject to resource constraints: Classification and complexity," *Discret. Appl. Math.*, vol. 5, no. 1, pp. 11–24, Jan. 1983.

[13] J. Krallmann, U. Schwiegelshohn, and R. Yahyapour, "On the Design and Evaluation of Job Scheduling Algorithms," in *Job Scheduling Strategies for Parallel Processing*, ser. Lecture Notes in Computer Science, D. G. Feitelson and L. Rudolph, Eds. Berlin, Heidelberg: Springer, 1999, pp. 17–42.

[14] M. R. Garey and R. L. Graham, "Bounds for Multiprocessor Scheduling with Resource Constraints," *SIAM J. Comput.*, vol. 4, no. 2, pp. 187–200, Jun. 1975.

[15] K. Aida, H. Kasahara, and S. Narita, "Job scheduling scheme for pure space sharing among rigid jobs," in *Job Scheduling Strategies for Parallel Processing*, ser. Lecture Notes in Computer Science, D. G. Feitelson and L. Rudolph, Eds. Berlin, Heidelberg: Springer, 1998, pp. 98–121.

[16] D. A. Lifka, "The ANL/IBM SP scheduling system," in *Job Scheduling Strategies for Parallel Processing*, ser. Lecture Notes in Computer Science, D. G. Feitelson and L. Rudolph, Eds. Berlin, Heidelberg: Springer, 1995, pp. 295–303.

[17] D. G. Feitelson, L. Rudolph, and U. Schwiegelshohn, "Parallel Job Scheduling — A Status Report," in *Job Scheduling Strategies for Parallel Processing*, ser. Lecture Notes in Computer Science, D. G. Feitelson, L. Rudolph, and U. Schwiegelshohn, Eds. Berlin, Heidelberg: Springer, 2005, pp. 1–16.

[18] D. Feitelson and A. Weil, "Utilization and predictability in scheduling the IBM SP2 with backfilling," in *Proc. 1st Merged Int. Parallel Process. Symp. and Symp. Parallel Distrib. Process.* Orlando, FL, USA: IEEE, Mar. 1998, pp. 542–546.

[19] S. Srinivasan, R. Kettimuthu, V. Subramani, and P. Sadayappan, "Characterization of backfilling strategies for parallel job scheduling," in *Proceedings. International Conference on Parallel Processing Workshop*. Vancouver, BC, Canada: IEEE, Aug. 2002, pp. 514–519.

[20] E. Gaussier, J. Lelong, V. Reis, and D. Trystram, "Online Tuning of EASY-Backfilling using Queue Reordering Policies," *IEEE Trans. Parallel Distrib. Syst.*, vol. 29, no. 10, pp. 2304–2316, Oct. 2018.

[21] S. Srinivasan, R. Kettimuthu, V. Subramani, and P. Sadayappan, "Selective Reservation Strategies for Backfill Job Scheduling," in *Job Scheduling Strategies for Parallel Processing*, ser. Lecture Notes in Computer Science, D. G. Feitelson, L. Rudolph, and U. Schwiegelshohn, Eds. Berlin, Heidelberg: Springer, 2002, pp. 55–71.

[22] B. W. Team, "Blue Waters User Portal," https://bluewaters.ncsa.illinois.edu/data-sets, 2022.

[23] D. G. Feitelson, D. Tsafrir, and D. Krakov, "Experience with using the Parallel Workloads Archive," *Journal of Parallel and Distributed Computing*, vol. 74, no. 10, pp. 2967–2982, Oct. 2014.

[24] K. Lamar, A. Goponenko, C. Peterson, B. A. Allan, J. M. Brandt, and D. Dechev, "Backfilling HPC Jobs with a Multimodal-Aware Predictor," in *2021 IEEE Int. Conf. CLUSTER*. Portland, OR, USA: IEEE, Sep. 2021, pp. 618–622.

[25] R. Gibbons, "A historical application profiler for use by parallel schedulers," in *Job Scheduling Strategies for Parallel Processing*, ser. Lecture Notes in Computer Science, D. G. Feitelson and L. Rudolph, Eds. Berlin, Heidelberg: Springer, 1997, pp. 58–77.

[26] W. Smith, I. Foster, and V. Taylor, "Predicting application run times with historical information," *Journal of Parallel and Distributed Computing*, vol. 64, no. 9, pp. 1007–1016, Sep. 2004.

[27] B. Allan, "Two Weeks In The Life of Skybridge," Tech. Rep. SAND–2019-4915, 1762352, 675183, Apr. 2019.

[28] E. L. Kaplan and P. Meier, "Nonparametric Estimation from Incomplete Observations," *Journal of the American Statistical Association*, vol. 53, no. 282, pp. 457–481, Jun. 1958.

[29] U. Schwiegelshohn and R. Yahyapour, "Analysis of first-come-first-serve parallel job scheduling," in *Proc. 9th Annu. ACM-SIAM SODA*. USA: Society for Industrial and Applied Mathematics, Jan. 1998, pp. 629–638.

[30] V. J. Leung, G. Sabin, and P. Sadayappan, "Parallel Job Scheduling Policies to Improve Fairness: A Case Study," in *39th Int. Conf. Parallel Process. Workshops*. San Diego, CA, USA: IEEE, Sep. 2010, pp. 346–353.

[31] R. Jain, D.-M. Chiu, and W. Hawe, "A Quantitative Measure Of Fairness And Discrimination For Resource Allocation In Shared Computer Systems," Digital Equipment Corporation, Hudson, MA, USA, DEC Research Report TR-301, Sep. 1984.

[32] S.-H. Chiang, A. Arpaci-Dusseau, and M. K. Vernon, "The Impact of More Accurate Requested Runtimes on Production Job Scheduling Performance," in *Job Scheduling Strategies for Parallel Processing*, ser. Lecture Notes in Computer Science, D. G. Feitelson, L. Rudolph, and U. Schwiegelshohn, Eds. Berlin, Heidelberg: Springer, 2002, pp. 103–127.

[33] G. Sabin and P. Sadayappan, "Unfairness Metrics for Space-Sharing Parallel Job Schedulers," in *Job Scheduling Strategies for Parallel Processing*, ser. Lecture Notes in Computer Science, D. Feitelson, E. Frachtenberg, L. Rudolph, and U. Schwiegelshohn, Eds. Berlin, Heidelberg: Springer, 2005, pp. 238–256.

[34] D. Perkovic and P. Keleher, "Randomization, Speculation, and Adaptation in Batch Schedulers," in *SC '00: Proc. 2000 ACM/IEEE Conf. Supercomputing*. Dallas, TX, USA: IEEE, Nov. 2000, pp. 7–7.

[35] W. Tang, Z. Lan, N. Desai, and D. Buettner, "Fault-aware, utility-based job scheduling on Blue, Gene/P systems," in *2009 IEEE Int. Conf. on Cluster Comput. and Workshops*. New Orleans, LA, USA: IEEE, Aug. 2009, pp. 1–10.

[36] D. Talby and D. Feitelson, "Supporting priorities and improving utilization of the IBM SP scheduler using slack-based backfilling," in *Proc. 13th IPPS and 10th SPDP*. San Juan, PR, USA: IEEE, Apr. 1999, pp. 513–517.

[37] C. Galleguillos, A. Sîrbu, Z. Kiziltan, O. Babaoglu, A. Borghesi, and T. Bridi, "Data-Driven Job Dispatching in HPC Systems," in *Machine Learning, Optimization, and Big Data*, ser. Lecture Notes in Computer Science, G. Nicosia, P. Pardalos, G. Giuffrida, and R. Umeton, Eds. Cham: Springer, 2018, pp. 449–461.

[38] A. Agarwal, S. Colak, and S. Erenguc, "A Neurogenetic approach for the resource-constrained project scheduling problem," *Comput. Oper. Res.*, vol. 38, no. 1, pp. 44–50, Jan. 2011.

[39] X. Zheng, Z. Zhou, X. Yang, Z. Lan, and J. Wang, "Exploring Plan-Based Scheduling for Large-Scale Computing Systems," in *2016 IEEE Int. Conf. CLUSTER*. Taipei, Taiwan: IEEE, Sep. 2016, pp. 259–268.

[40] M. R. Wyatt, S. Herbein, T. Gamblin, A. Moody, D. H. Ahn, and M. Taufer, "PRIONN: Predicting Runtime and IO using Neural Networks," in *Proc. 47th Int. Conf. ICPP*. Eugene, OR, USA: Association for Computing Machinery, Aug. 2018, pp. 1–12.

[41] T. H. Le Hai, L. La Hoang, and N. Thoai, "Potential of Applying kNN with Soft Walltime to Improve Scheduling Performance," in *2021 IEEE Int. Conf. ICCMA*. Brest, France: IEEE, Jul. 2021, pp. 1–8.

[42] Q. Wang, J. Li, S. Wang, and G. Wu, "A Novel Two-Step Job Runtime Estimation Method Based on Input Parameters in HPC System," in *2019 IEEE 4th Int. Conf. ICCCBDA*. Chengdu, China: IEEE, Apr. 2019, pp. 311–316.

[43] A. Matsunaga and J. A. Fortes, "On the Use of Machine Learning to Predict the Time and Resources Consumed by Applications," in *2010 10th IEEE/ACM International Conference on Cluster, Cloud and Grid Computing*. Melbourne, VIC, Australia: IEEE, May 2010, pp. 495–504.

[44] D. G. Feitelson, "Resampling with Feedback: A New Paradigm of Using Workload Data for Performance Evaluation," in *Job Scheduling Strategies for Parallel Processing*, ser. Lecture Notes in Computer Science, D. Klusáček, W. Cirne, and G. P. Rodrigo, Eds. Cham: Springer, 2021, pp. 3–32.

[45] J. P. Jones and B. Nitzberg, "Scheduling for Parallel Supercomputing: A Historical Perspective of Achievable Utilization," in *Job Scheduling Strategies for Parallel Processing*, ser. Lecture Notes in Computer Science, D. G. Feitelson and L. Rudolph, Eds. Berlin, Heidelberg: Springer, 1999, pp. 1–16.

[46] E. Shmueli and D. G. Feitelson, "On Simulation and Design of Parallel-Systems Schedulers: Are We Doing the Right Thing?" *IEEE Trans. Parallel Distrib. Syst.*, vol. 20, no. 7, pp. 983–996, Jul. 2009.

[47] N. Zakay and D. G. Feitelson, "Semi-Open Trace Based Simulation for Reliable Evaluation of Job Throughput and User Productivity," in *2015 IEEE 7th Int. Conf. CloudCom*, Nov. 2015, pp. 413–421.

[48] Y. Fan, P. Rich, W. E. Allcock, M. E. Papka, and Z. Lan, "Trade-Off Between Prediction Accuracy and Underestimation Rate in Job Runtime Estimates," in *2017 IEEE Int. Conf. CLUSTER*. Honolulu, HI, USA: IEEE, Sep. 2017, pp. 530–540.