An experimental analysis of regression-obtained HPC scheduling heuristics

Lucas Rosa, Danilo Carastan-Santos, Alfredo Goldman

An experimental analysis of regression-obtained HPC scheduling heuristics

Lucas Rosa[1][0000−0001−9651−7781], Danilo Carastan-Santos[2][0000−0002−1878−8137], and Alfredo Goldman[1][0000−0001−5746−4154]

[1] Institute of Mathematics and Statistics, University of São Paulo
[2] Univ. Grenoble Alpes, CNRS, Inria, Grenoble INP, LIG
roses.lucas@usp.br, danilo.carastan-dos-santos@inria.fr, gold@ime.usp.br

Abstract. Scheduling jobs in High-Performance Computing (HPC) platforms typically involves heuristics consisting of job sorting functions such as First-Come-First-Served or custom (hand-engineered). Linear regression methods are promising for exploiting scheduling data to create simple and transparent heuristics with lesser computational overhead than state-of-the-art learning methods. The drawback is lesser scheduling performance. We experimentally investigated the hypothesis that we could increase the scheduling performance of regression-obtained heuristics by increasing the complexity of the sorting functions and exploiting derivative job features. We used multiple linear regression to develop a factory of scheduling heuristics based on scheduling data. This factory uses general polynomials of the jobs' characteristics as templates for the scheduling heuristics. We defined a set of polynomials with increasing complexity between them, and we used our factory to create scheduling heuristics based on these polynomials. We evaluated the performance of the obtained heuristics with wide-range simulation experiments using real-world traces from 1997 to 2016. Our results show that large-sized polynomials led to unstable scheduling heuristics due to multicollinearity effects in the regression, with small-sized polynomials leading to a stable and efficient scheduling performance. These results conclude that (i) multicollinearity imposes a constraint when one wants to derive new features (i.e., feature engineering) for creating scheduling heuristics with regression, and (ii) regression-obtained scheduling heuristics can be resilient to the long-term evolution of HPC platforms and workloads.

Keywords: Scheduling Heuristics · High Performance Computing · Machine Learning · Linear Regression.

1 Introduction

In many fields of science and industry (climate, health, economics, artificial intelligence, *etc.*), High-Performance Computing (HPC) is an essential element to solve complex problems and to process the ever-increasing amount of data being generated. Informally, HPC consists of utilizing highly parallel and distributed computing platforms. We have reached an extreme-scale of such platforms, with

ranks such as the Top500 [21] listing supercomputers with millions of interconnected processors.

Among the many important problems that arise at such a scale, resource management is a critical problem that needs to be solved efficiently for a proper use of such platforms. Resource management involves assigning when and where HPC applications (hereafter referred to as jobs) will be processed in an HPC platform. HPC platform users (research groups, companies, *etc.*) submit their jobs to be processed at any given point, and limited information about the jobs is available for decision-making. This decision-making typically requires solving instances of a problem called online parallel job scheduling.

Despite the numerous theoretical advancements, the practical solution for online parallel job scheduling is typically based on scheduling heuristics that employ waiting queue sorting algorithms. The widely used algorithm is the First-Come-First-Served (FCFS) job ordering, alongside the utilization of backfilling algorithms [22]. One of possible reasons for this phenomenon is the explainability of the scheduling heuristics, which may be given by their simplicity. Waiting queue sorting heuristics are easy to understand by both the HPC platform maintainers and users, with FCFS ordering arguably being the easiest to understand. Regarding FCFS, this high level of explainability, comes with the drawback of poor scheduling performances [7].

Recent works [6, 17] employ Machine Learning (ML) techniques to exploit simulation and platform workload data to create scheduling heuristics, as an attempt encode scheduling knowledge obtained by ML with explainable and efficient scheduling heuristics. With this regard, regression methods [6] appear as a promising approach, since we can express scheduling knowledge in the form of simple features of the jobs' characteristics. The hypothesis of a trade-off between simplicity/performance is still present. We may need to sacrifice simplicity of the features to obtain better scheduling performances. Another underexplored concern is whether we need or not to readjust the regression-obtained heuristics as the platforms and jobs characteristics evolve.

Our work is a step towards clarifying this hypothesis by concerning the following research questions: *(i) By using regression methods to create scheduling heuristics, do we gain scheduling performance by using more complex features?* and *(ii) do regression-obtained heuristics provide stable scheduling performance over the evolution of HPC platforms and workloads?*

This work builds upon the work of Carastan-Santos and de Camargo [6] to develop a factory of scheduling heuristics based on scheduling data. This factory uses general polynomials of the jobs' characteristics as templates for the scheduling heuristics. We elaborated an experimental methodology to evaluate polynomials, with increasing levels of complexity, as scheduling heuristics.

We found a surprising result that multicollinearity – a phenomenon intrinsic to regression methods – may drastically degrade the scheduling performance of obtained scheduling heuristics constituted by complex, large-sized polynomials. A functional form constituted by a linear combination of the jobs' characteristics – the simplest and smallest of all polynomials evaluated – presented the best

scheduling performances. More specifically, in this paper we present the following contributions:

- We performed an experimental study on how to obtain online parallel job scheduling policies in the form of polynomials of the jobs' characteristics. We adopted simulation and regression methods to evaluate a class of polynomials with increasing degrees of complexity. Our results indicate that multicollinearity effects may happen when using regression methods with high degree polynomials, leading to instability of the coefficients and inefficient scheduling performances.
- We showed experimental evidence that small-sized polynomials – a linear combination of the jobs' characteristics in our example – can lead to efficient scheduling heuristics. We also showed that regression-obtained heuristics can be resilient to the long-term evolution of HPC platforms and workloads without needing to be readjusted over time.

We organized the remaining of this paper as follows: in Section 2, we present an overview of related works, and Section 3 presents some preliminary definitions of online parallel job scheduling. We present our methodology in Section 4 and our experimental results in Section 5. Lastly, in Section 6, we present our concluding remarks and future works. The code base for this paper is openly available at a Github repository[3].

2 Related Work

Scheduling in theory The research community studied the parallel job scheduling problem under a general problem called multiple-strip packing problem [4]. Given a set of rectangles (jobs) and set of strips (set of computing resources), the objective is to find a packing of the rectangles into the strips, that minimize the height among the used strips. Such problem in the single-strip case is already NP-hard [4]. Many theoretical works [5, 27, 16, 30, 28] proposed approximation algorithms and performance bounds, but many of these works relied on analytically tractable objective metrics such as makespan (i.e., the completion time of the last finishing job). Makespan is arguably less relevant in the online case, when compared to metrics such as waiting time or slowdown (see Section 4), since we do not have information about a "last job" in an online scheduling configuration.

Scheduling in practice: scheduling heuristics In online parallel job scheduling, many works explore list scheduling [23] based algorithms that rely on waiting queue ordering heuristics. These heuristics can be created by hand-engineering methods [25], and by tuning methods [18, 17], that forecast possible outcomes of a certain heuristic.

[3] https://github.com/fredgrub/scheduling-simulator

It is long known [12], however, that most practitioners employ First-Come-First-Served (FCFS) based heuristics with a backfilling [22] mechanism, or arbitrary job prioritization, using for instance multiple queue priorities [24]. A possible explanation for the popularity of FCFS-based heuristics is the fact that FCFS is (i) easily understandable by both practitioners and users and (ii) it offers desirable properties such as no-starvation (i.e., every job will eventually execute).

Machine learning in scheduling Machine Learning (ML) is used in the context of online job scheduling mainly in two scenarios: (i) to improve scheduling by predicting the jobs' characteristics [14, 19, 31], and (ii) to create novel heuristics by using techniques such as non-linear regression [6] and evolutionary [17] strategies. More recently, deep reinforcement learning methods [10, 29] are being explored to perform online job scheduling.

Our work focuses on using ML methods to create novel scheduling heuristics. Specifically, we aim to produce knowledge about regression methods to obtain clear and interpretable heuristics. Our approach differs from the work of Legrand *et al.* [17] as we want to investigate how much we can gain in scheduling performance by deriving new features from existing jobs' characteristics. Carastan-Santos and de Camargo [6] took the first step in using regression methods to obtain clear and interpretable scheduling heuristics. However, their heuristics relied on features based on terms such as square roots and logarithms, which are difficult to interpret in the scheduling context. We supplement the work of Carastan-Santos and de Camargo [6] by using a functional form generation procedure to create heuristics in simple polynomials of the jobs' characteristics, taking into account increasing levels of complexity and derivative features.

3 Background

3.1 Online Parallel Job Scheduling

On a typical HPC platform, users submit their applications at any given moment to be processed. The resource and job management system (RJMS) manages both the applications and platform resources [15].

Among other tasks, the RJMS has to deal with a scheduling problem called online parallel job scheduling problem. As an overview, given a set of n machines (processors) $\mathcal{M} = \{m_1, \ldots, m_n\}$ that need to process parallel jobs $j_1, j_2, \ldots$ that arrive at any time, the online parallel job scheduling problem consists of deciding when and at which machines the jobs will be processed, in a way to maximize a certain performance objective. We assume that these machines are homogeneous (i.e., have the same processing power), and are connected by any interconnection topology.

Under the view of the RJMS, the processors of an HPC platform are modeled as the set of machines $\mathcal{M}$, and the applications submitted by the users are modeled as the jobs j. The RJMS has limited information about the jobs, and

this information is only available after the job's submission. The main information available are notably (i) the estimated job's processing time ($\tilde{p}_j$), normally informed by the user, (ii) the number of processors required to process the job (q_j), and (iii) the time when the job was submitted in the platform (r_j, also known as release time).

Another important piece of information that is only available after finishing a job j is its actual processing time (p_j). The estimation of the processing time $\tilde{p}_j$ usually works as an upper bound of the actual processing time p_j. We also consider that the number of processors q_j is fixed and can not change over time (also referred to as rigid jobs). The RJMS registers information about the jobs in a log file, often shared using the *Standard Workload Format* (SWF)[13].

3.2 Scheduling policies and Backfilling

The order in which the jobs are submitted for processing is often determined by a *scheduling policy*, which is a crucial component of many online parallel job scheduling heuristics. A scheduling policy typically consists by a job sorting function $f(j)$, that takes as input the characteristics of a job j (e.g., $\tilde{p}_j$, q_j, and r_j), and outputs a number that quantifies the priority of processing j. Typically, the lower the value of $f(j)$, the more priority a job j has. The RJMS applies $f(j)$ to all jobs in the waiting queue to set their execution priorities in two events: (i) when a new job arrives in the waiting queue and (ii) when one or more processors are released and becomes available.

On top of a scheduling policy, often a *backfilling* [22] mechanism is applied. Informally, backfilling mechanisms allow a job j to be selected for processing earlier – and breaking the priority order defined by the scheduling policy – if j does not delay the processing of jobs with higher priority. Backfilling is known to consistently bring positive performance effects, regardless of the scheduling policy [7], and it is widely used by practitioners, in conjunction with the First-Come-First-Served (FCFS, $f(j) = r_j$) scheduling policy.

3.3 Scheduling performance metric

In this work we adopt the average bounded slowdown (AVGbsld) as the scheduling performance metric. Given a job j, its bounded slowdown (bsld) value can be computed as follows

$$\text{bsld}_j = \max\left\{\frac{w_j + p_j}{\max(p_j, \tau)}, 1\right\} \tag{1}$$

where w_j is the time that a job waited for processing since its submission and τ is a constant that is typically set in the order of 10 seconds, that prevents small jobs from having excessively large slowdown values. The average bounded slowdown takes into account the slowdown average for a set of jobs J and is defined as

$$\text{AVGbsld}(J) = \frac{1}{|J|}\sum_{j \in J} \max\left\{\frac{w_j + p_j}{\max(p_j, \tau)}, 1\right\} \tag{2}$$

AVGbsld is an interesting metric in the sense that it can express the expectation that the waiting time of the jobs should be proportional of their processing time [11], though this metric is more sensible to small jobs (i.e., jobs with small p and q values). AVGbsld is also a hard objective to be treated theoretically in online parallel job scheduling, and it is in practice mainly manageable by heuristics.

4 Experimental Procedure

We adapted the method of Carastan-Santos and de Camargo [6] to suit to our study. Our method consists of a two-step approach, (i) we gain and register scheduling knowledge by exploiting fast simulations of a *proxy scheduling problem* (i.e., a similar though simpler scheduling problem, see Section 4.1). The strategy is that with simulations and a proxy problem we can perform a larger search of possible scheduling decisions in feasible time. Then (ii) we feed the knowledge observed in the simulations in a linear regression method, that models the observed scheduling knowledge into functions of the jobs' characteristics. The main hypothesis is that these functions will perform as good scheduling policies for the original scheduling problem (see Section 3).

4.1 Simulation strategy

The proxy scheduling problem differs from the main problem (as described in Section 3) in a few key ways. Firstly, in the proxy problem, we have access to perfect information about the job processing time (i.e., $\tilde{p}_j = p_j$). Secondly, the proxy problem is an offline scheduling problem, meaning that all information about the jobs (p, q, and r) is known in advance. Lastly, the number of jobs that arrive in the proxy problem is finite. The purpose of defining this simpler scheduling problem is to facilitate quick simulations that are similar to the target problem.

Let S and Q two sets of jobs. We frame a proxy scheduling problem as the scheduling of the jobs in Q, in an HPC platform with m interconnected processors where the jobs in S are currently being processed. This configuration is a way to model the waiting queue (Q) and the initial state of the platform (S). The job sets S and Q are generated in the following way: from a large enough job log file (also referred to as trace) N, we randomly select a subtrace $M \subset N$ with size $|S| + |Q|$. The first $|S|$ jobs from M belongs S and the remaining $|Q|$ jobs from M belongs Q.

For a pair of sets of jobs (S, Q), we start the simulation by submitting the jobs from the set S in FCFS order. When the last job of S is submitted, jobs from S may still be processing in the platform, thus representing a possible initial state. At this point we start branching the simulation, with different branch (also referred in the text as *trials*) representing a possible way to schedule the jobs in Q.

Next, we randomly sample permutations Q^* of the set Q, and each sample Q^* represents a branch of simulation. For each sample Q^*, we simulate the scheduling of the jobs, following the order that the jobs are present in Q^*. The total number of permutations of Q increases exponentially in function of the size of Q, and the number of sample permutations to decently represent the permutation space rapidly increases as well. One of the reasons to use an aforementioned scheduling proxy task is that the above simulation branching strategy can only be done in feasible time with fixed (and small, see Section 5) number of jobs in Q.

With $\mathcal{P}$ being the set containing all sampled permutations, when the scheduling simulation of all branches end, we compute and assign a score for each job $j \in Q$:

$$\text{score}(j) = \frac{\sum_{Q_j^* \in \mathcal{P}(j_0=j)} \text{AVGbsld}(Q_j^*)}{\sum_{Q_k^* \in \mathcal{P}} \text{AVGbsld}(Q_k^*)} \tag{3}$$

The score represents the consequence of scheduling a job $j \in Q$ first (represented by j_0 in Equation 3), in terms of the average bounded slowdown of all jobs in Q. Jobs with low score resulted in a positive consequence on the average slowdown when they are chosen to be executed first. The set $\{\text{score}(j) \mid \forall j \in Q\}$ defines a *score distribution* of the jobs of Q given the initial state S.

The score(j) along with the characteristics p_j, q_j and r_j are the central output of the simulations. We conjecture that with an initial state represented by the scheduling of the jobs in S, sorting the jobs in Q in increasing order of score(j) results in an efficient schedule regarding the AVGbsld(Q) (Equation 2).

We repeat the aforementioned simulation strategy with many samples of job set pairs (S, Q). The idea is that each distinct pair (S, Q) represents a certain scheduling situation, and the computed $\{\text{score}(j) \mid \forall j \in Q\}$ represents a good scheduling strategy for this scheduling situation.

4.2 Multiple Linear Regression model

After the simulation step, we obtain a data set that includes information about the characteristics of the jobs, such as p_j, q_j, and r_j, as well as their computed score(j). This data set is referred to as the *training data set*. The next step aims to utilize this data set to generate a general representation of the relationship between the values of score(j) and the job's characteristics, that is, p_j, q_j, and r_j.

Let J be all jobs present in the training data set with their compute score values, the problem consists in finding functions $f(p_j, q_j, r_j)$ that provide a good approximation to the score values of all jobs $j \in J$. For this task, we modified the original strategy proposed by Carastan-Santos and de Camargo [6]. We defined a function family $\mathcal{F}$, with parametrized functions of the form

$$f(\theta, \mathbf{x}) = \theta^T \mathbf{x} \tag{4}$$

where θ is a parameter vector, and $\mathbf{x}$ is the features vector of the jobs' characteristics p, q and r.

We defined a function family containing four functions whose vectors $\mathbf{x}$ are presented by Table 1. Each function works as a template for obtaining the scheduling heuristics. Each element in the vectors (also referred to as basis functions) are polynomials of the jobs' characteristics, with degrees in the set $\{1, 2, 3, 4\}$. The reasons behind the choice of these functions are the following:

- Our intention was to avoid possible non-linearity between the parameters, which allows us to use multiple linear regression, in which a least squares minimization algorithm can provide optimal parameters θ with regards to the sum of squared loss (see Equation 5).
- We wanted to evaluate the effectiveness of using multiple regression to develop scheduling heuristics for functions of varying complexity. Thus, we have defined a limited collection of polynomials that exhibit significant variation in complexity among them.
- We sought to use more easily understandable terms in the scheduling sense, as opposed to the more complex terms that were used in [6], such as square roots and logarithms. Additionally, we aimed to avoid using job characteristic polynomials with unconventional degrees (such as 0.5 for square root, -0.5 for inverse square root, or 1.5), and combinations of job characteristics that do not correspond to a known scheduling heuristic (such as derivative features of pr or qr).
- We wanted to evaluate the effects of derivative features of p, q and r and the area (see explanation below) pq, with increasing degrees of penalization for their values.

Table 1: Features of the four parametrized functions used in multiple linear regression.

Vector components	Vector $\mathbf{x}$			
	Lin	Qdr	Cub	Qua
$(1, p, q, r)$	✓	✓	✓	✓
(p^2, q^2, r^2, pq)		✓	✓	✓
$(p^3, q^3, r^3, p^2q, pq^2)$			✓	✓
$(p^4, q^4, r^4, p^3q, p^2q^2, pq^3)$				✓

We defined four functions, Lin, Qdr, Cub and Qua. The function Lin is just a linear combination of the jobs' characteristics p, q and r. The others Qdr, Cub and Qua are functions that progressively increase the degree of the basis functions, and with multiplicative factors related to pq, which is often referred in the literature as the area of the jobs. We considered derivative factors that correspond to well-known scheduling policies, notably the First-Come First-Served (FCFS $= r$), Shortest Processing Time first (SPT $= p$), Shortest Number of Processors First (SQF $= q$), and Shortest Area First (SAF $= pq$). Therefore, we discarded derivative forms of pr and qr.

We employ a weighted multiple linear regression [8] procedure, which minimizes the weighted sum of squared loss function:

$$\Sigma_{\text{wL}} = \sum_{j \in J} \left[(p_j q_j) \cdot (f(\theta, \mathbf{x}) - \text{score}(j)) \right]^2 \tag{5}$$

The weight $(p_j q_j)$ emphasizes that the approximation must perform a good estimation of the score of large area jobs (i.e., jobs with p and q large), as they can end up blocking the execution of small jobs, degrading the overall scheduling performance.

After obtaining the coefficients $\hat{\theta}^f$ from the multiple linear regression approximation for all functions $f(\theta, \mathbf{x}) \in \mathcal{F}$, we can measure the approximation quality through the *Mean Absolute Error* function (MAE, Equation 6).

$$\text{MAE}(f) = \frac{1}{|J|} \sum_{j \in J} \| f(\hat{\theta}^{\mathbf{f}}, \mathbf{x}) - \text{score}(j) \| \tag{6}$$

Using these approximated functions, we performed an experimental campaign applying simulations to determine the efficiency of using them as scheduling policies in an online parallel scheduling scenario.

5 Results and Discussion

Within this section, we present the primary outcomes attained through our research. A training data set with $(p_j, q_j, r_j, \text{score}(j))$ values was created and used to evaluate the effectiveness of the regression-obtained scheduling heuristics. The simulations were conducted on an Amazon EC2 c5a.2xlarge instance equipped with a second-generation AMD EPYC 7002 processor with up to 3.3 GHz frequencies, 8 virtual CPU cores, and 16GB of RAM. All simulations, including obtaining the training data set and evaluating the scheduling policies, were performed using the SimGrid [9] simulation software. The job characteristics were obtained from a trace generated with the Lublin and Feitelson [20] workload model, with set sizes of $|S| = 16$ and $|Q| = 32$.

While obtaining the training data set, we had to determine the maximum number of trials that we could perform with a pair (S, Q) in the `c5a.2xlarge` instance. This was necessary to strike a balance between precision in calculating score(j) and efficient execution time. Figure 1 shows the normalized standard deviation of calculating score(j) in function of the number of trials, and we added their respective processing times, when running for one pair (S, Q) in the `c5a.2xlarge` instance. We decided to keep 256 thousand trials, as it showed to provide accurate calculations of score(j) and its processing time was acceptable within our budget. Moreover, we used the SciPy's `curve_fit` [26] method to perform the multiple linear regression.

Throughout this section, we refer to the term *on-line scheduling experiment* as being multiple simulations of the on-line scheduling of jobs, whose information are obtained from a certain workload log (trace) or model. For each simulation

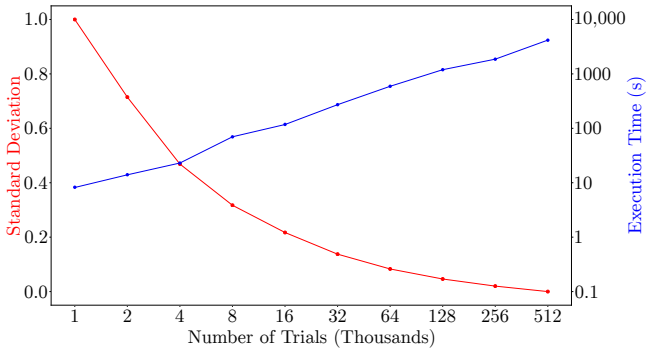

Fig. 1: The figure, adapted from [6], shows, in red, the normalized standard deviations of the trial score distributions obtained with different numbers of trials, and in blue, the execution time in the chosen instance in logarithmic scale.

we choose policies from Tables 1 and 2, and we simulate the on-line scheduling of jobs, using the chosen policies as scheduling heuristic. The output of the on-line scheduling experiment is the average bounded slowdown (see Equation 2) from each simulation performed, using all chosen policies. Each simulation considers a job workload consisting of a fifteen-days (15) jobs submission. When using real workload logs, we assured that there was no overlap between sequences used on distinct on-line scheduling experiments.

Table 2: Scheduling policies used for comparison. Detailed information regarding WFP3, UNICEF, and F2 policies can be found in [25] and [6].

Policy name	Function
FCFS	r_j
SPT	$\tilde{p}_j$
SAF	$\tilde{p}_j \cdot q_j$
WFP3	$-(w_j/\tilde{p}_j)^3 \cdot q_j$
UNICEF	$-w_j/(\log_2(q_j) \cdot \tilde{p}_j)$
F2	$\sqrt{\tilde{p}_j \cdot q_j} + 2.56 \times 10^4 \cdot \log_{10}(r_j)$

The simulation of the on-line scheduling works as follows: tasks arrive in a centralized waiting queue and the scheduler performs a reschedule – using a scheduling policy – of the tasks present in this queue in two distinct events: (i) when a task arrives in the queue or (ii) when a resource (set of processors) is released and becomes available. When a job j is selected for processing and if the requested number of processors q_j is lower than the total number of processors available, then q_j processors are reserved for this task and they become

unavailable. These processors will become available again only when p_j units of time have passed since the start of the execution of j. If there are not enough processors to process j, the scheduler will wait for one of the two rescheduling events mentioned previously. The scheduler has no information about future job arrivals and the jobs' data p, $\tilde{p}$, q, and r are only known to the scheduler when the jobs arrive.

We explored the methodology presented in Section 4 to bring light to several research questions that we elaborate in the following sections.

5.1 Does the effectiveness of regression-based scheduling heuristics increases as a function of polynomial size?

To address this question, we employed the methodology detailed in Section 4. Specifically, we used multiple linear regression on the functions Lin, Qdr, Cub, and Qua; using a dataset comprising of 14,081 jobs and their corresponding score(j). Our analysis was conducted on a synthetic homogeneous platform with 256 processors. The complete list of coefficients θ, obtained through regression, is presented in Table 3. Below, we provide a qualitative description of these coefficients.

Table 3: Features of the four defined functions, their optimal coefficients, and their Variance Inflation Factor (VIF).

Vector x	Coefficients θ				VIF			
	Lin	Qdr	Cub	Qua	Lin	Qdr	Cub	Qua
1	$3.24 \cdot 10^{-2}$	$3.70 \cdot 10^{-2}$	$3.33 \cdot 10^{-2}$	$4.83 \cdot 10^{-2}$	–	–	–	–
p	$1.15 \cdot 10^{-7}$	$2.65 \cdot 10^{-7}$	$2.83 \cdot 10^{-7}$	$-6.42 \cdot 10^{-7}$	1.3	3.7	12.1	36.1
q	$2.61 \cdot 10^{-5}$	$-3.05 \cdot 10^{-5}$	$8.71 \cdot 10^{-5}$	$-2.40 \cdot 10^{-4}$	1.3	9.6	35.5	99.0
r	$-1.57 \cdot 10^{-7}$	$-3.96 \cdot 10^{-7}$	$-5.83 \cdot 10^{-7}$	$-6.50 \cdot 10^{-7}$	1.2	6.0	21.8	58.6
p^2	–	$-3.04 \cdot 10^{-12}$	$-6.33 \cdot 10^{-14}$	$1.66 \cdot 10^{-11}$	–	2.6	41.8	338.5
q^2	–	$1.17 \cdot 10^{-7}$	$-5.06 \cdot 10^{-7}$	$2.41 \cdot 10^{-6}$	–	8.3	275.0	2835.5
r^2	–	$2.75 \cdot 10^{-12}$	$6.78 \cdot 10^{-12}$	$8.81 \cdot 10^{-12}$	–	4.8	82.1	620.8
pq	–	$6.77 \cdot 10^{-10}$	$-6.02 \cdot 10^{-10}$	$1.14 \cdot 10^{-8}$	–	3.9	49.4	295.3
p^3	–	–	$-7.55 \cdot 10^{-18}$	$-2.03 \cdot 10^{-16}$	–	–	20.8	961.6
q^3	–	–	$7.05 \cdot 10^{-10}$	$-9.52 \cdot 10^{-9}$	–	–	147.3	10491.8
r^3	–	–	$-2.14 \cdot 10^{-17}$	$-4.67 \cdot 10^{-17}$	–	–	34.6	1387.1
p^2q	–	–	$-2.72 \cdot 10^{-14}$	$9.29 \cdot 10^{-15}$	–	–	9.8	393.1
pq^2	–	–	$9.52 \cdot 10^{-12}$	$-7.94 \cdot 10^{-11}$	–	–	30.1	1032.9
p^4	–	–	–	$9.34 \cdot 10^{-22}$	–	–	–	322.7
q^4	–	–	–	$1.36 \cdot 10^{-11}$	–	–	–	3572.5
r^4	–	–	–	$9.84 \cdot 10^{-23}$	–	–	–	373.6
p^3q	–	–	–	$-9.07 \cdot 10^{-19}$	–	–	–	63.8
p^2q^2	–	–	–	$1.89 \cdot 10^{-16}$	–	–	–	125.0
pq^3	–	–	–	$1.55 \cdot 10^{-13}$	–	–	–	457.2

We observed instabilities in the features' coefficients between Lin, Qdr, Cub, and Qua. These instabilities were observed through changes in the sign of the coefficients of a particular functional form, which carry implications for the job scheduling. For example, the Lin function prioritizes jobs with smaller values of q (positive coefficients) while the Qdr function prioritizes jobs with greater values of q (negative coefficients).

For Lin, the positive coefficient for q is expected to be beneficial for the schedule, as it is associated with the Smaller Number of Processors First (SQF) scheduling policy which has been reported to improve scheduling performance compared to the First Come First Serve (FCFS) policy [7]. For Qdr, the negative coefficient for q is likely to have a detrimental effect on the scheduling, since it is linked to the Largest Number of Processors First (LQF) policy which has been shown to cause worse scheduling performance than FCFS, and is thus often disregarded in scheduling policy research [7, 17].

To illustrate these effects, Figure 2 depicts the normalized score(j) values of the functions Lin, Qdr, Cub and Qua when varying the values of p and q, and with a fixed value of r. Darker regions in this figure illustrate higher job priorities. We can observe that all functions have different behavior. Lin prioritizes jobs with lower p and q values – which is similar to the quite efficient [7] Shortest Area First (SAF) scheduling policy – Qdr, Cub and Qua give more or less priority in specific (less interpretable) regions.

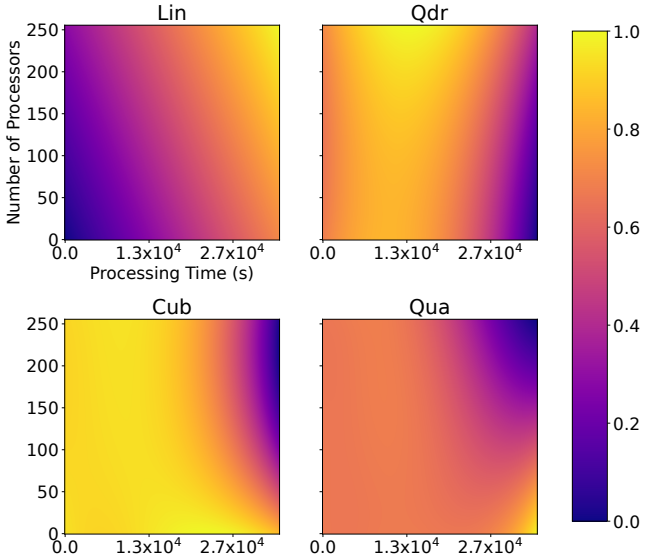

Fig. 2: Dependency on the jobs characteristics p and q for the functions Lin, Qdr, Cub and Qua.

We conducted online job scheduling simulations to evaluate the performance of our four candidate policies. The jobs were generated using the Lubin & Feitelson synthetic workload model (as described in Section 4.1). We also simulated a homogeneous HPC platform composed of 256 processors.

Figure 3 displays the average bounded slowdown for fifty dynamic scheduling experiments on the homogeneous HPC platform. The results highlighted that the Qdr, Cub and Qua functions, which are more complex, presented the worst performance compared to the scheduling policies. Although the mean absolute error values remained close for almost all functions (Lin = 4.48×10^{-3}, Qdr = 4.66×10^{-3}, Cub = 10.5×10^{-3}, and Qua = 6.42×10^{-3}), the performance of Qdr, Cub and Qua was as inefficient as FCFS and this further cemented the negative effects of the coefficients for Qdr, Cub and Qua. The Lin function showed better results, comparable to SAF, which is known to be quite efficient. We will further discuss the reasons for the inefficiency of Qdr, Cub and Qua in the following section.

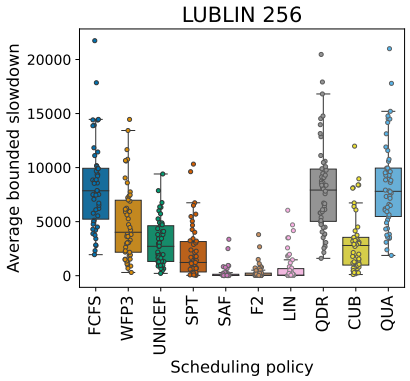

Fig. 3: Scheduling performance comparison with jobs generated from Lublin & Feitelson workload model and using the actual processing time.

Multicollinearity results in poor scheduling heuristics This instability in the coefficients of Qdr, Cub and Qua indicates that these functions suffer from the phenomenon of multicollinearity [2], that happens when the features are highly correlated between each other. At this light, multicollinearity may be happening when we add derivative features in conjunction to the jobs' characteristics p, q and r (e.g., q^2, q^3, q^4).

To verify this multicollinearity hypothesis, we calculated the Variance Inflation Factor [1] (VIF) for each individual feature present in the functions Lin, Qdr, Cub and Qua. Variance inflation factors range from 1 upwards. The numerical value for VIF tells (in decimal form) what percentage the variance is inflated for each coefficient. For example, a VIF of 1.9 tells you that the variance

of a particular coefficient is 90% larger than what you would expect if there was no multicollinearity (i.e., if there was no correlation with other features).

Therefore, one way to classify the correlation of the features in terms of their VIF is the following: (i) not correlated ($VIF = 1$), (ii) moderately correlated ($1 \leq VIF \leq 5$), and (iii) highly correlated ($VIF > 5$). The calculated VIF values are described on Table 3. The Qdr function displays VIF values greater than 5 as in q, r, q^2. Furthermore, the higher degree functions (Cub and Qua) presented very high values, reaching values greater than 10 thousand as in the term q^3 of the function Qua. We conjecture therefore that adding more derivative features will result to larger VIFs, and the multicollinearity effect will be present.

These results suggest that obtaining new scheduling heuristics for HPC platforms depends on the selection of key features and maintaining a simplistic model. In the rest of the experiments we put emphasis in the Lin function, since it was the function with the lowest level of multicollinearity.

5.2 How regression-obtained scheduling heuristics behave in long term?

The purpose of the experiments described below was to evaluate the generalizability of regression-obtained scheduling heuristics across a range of different workloads. This was done by comparing them with the policies presented in Table 2, using workload traces obtained from large-scale HPC platforms. The traces, which can be found at the Parallel Workloads Archive [13] and the ALAS Repository [3], were chosen to represent the long-term development of HPC platforms and workloads, covering 19 years from the oldest to the newest traces. The analyzed traces range from an old IBM SP2 machine with a few hundred CPUs to a modern supercomputer with hundreds of thousands of CPUs.

We conducted online scheduling experiments in two distinct situations: (i) scheduling utilizing jobs' processing time estimates and (ii) scheduling using processing time estimates in conjunction with aggressive backfilling [22]. The second scenario is particularly relevant to a real-world HPC platform situation. It is noteworthy that the SPT, SAF, and Lin policies do not prevent job starvation. Although the F2 policy does prevent starvation, its anti-starvation capacity diminishes over time due to its term $\log_{10}(r)$. We evaluated these policies without any starvation prevention mechanism. However, job starvation can be effectively prevented for any of these policies through a thresholding mechanism, whereby a job is given the highest priority if it passes a certain threshold, such as waiting time or estimated slowdown.

Scheduling experiments based solely on user-estimated processing times The experimental results are depicted in Figure 4. The linear function exhibited a low average bounded slowdown for varying workloads, with small data dispersion in nearly all cases, ensuring greater stability and predictability. Moreover, the Lin function exhibited one of the three lowest median bounded slowdown averages across all experiments, contending with SAF and F2 policies for these positions.

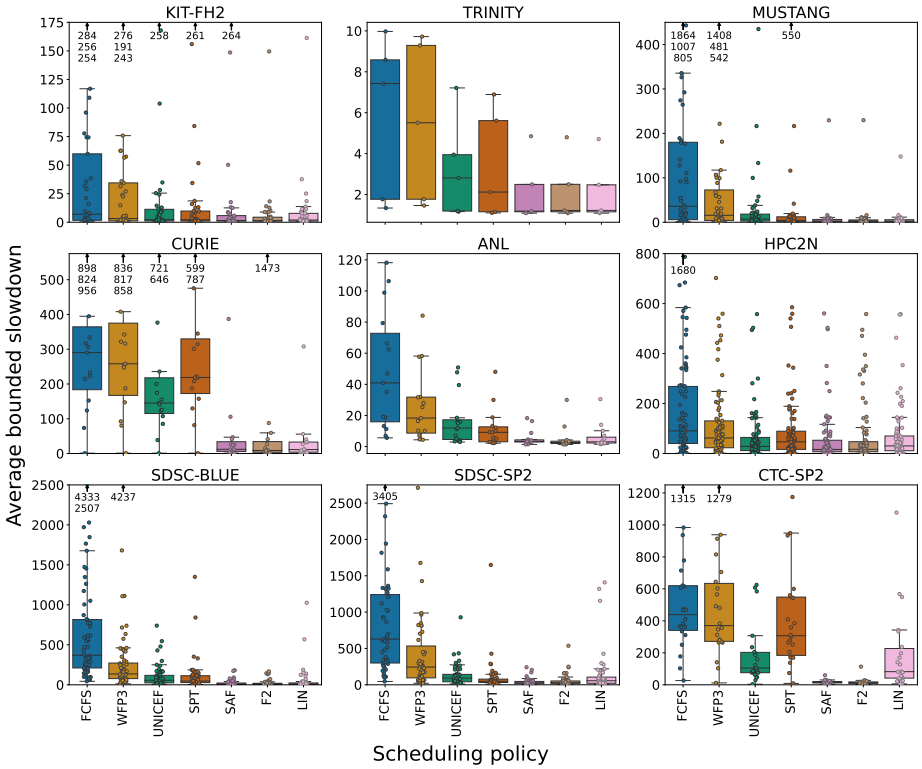

Fig. 4: Computed average bounded slowdown for different scheduling policies. Experiments based solely on user-estimated processing times.

There were some cases where the average slowdowns were significantly high. This was attributed to the presence of uncommon fifteen-day workloads. One such example is the HPC2N trace, which includes bursts of jobs with large processing time, thereby overloading the platform, irrespective of the policy in place. Despite distinct workload traces, the regression-obtained heuristics were found to be effective scheduling policies for diverse job types.

Scheduling experiments using estimated processing times and aggressive backfilling As the most realistic scenario, an aggressive backfilling technique was employed to reduce the idleness of the platform. Figure 5 illustrates that backfilling significantly improves the performance of poorly-performing heuristics, such as FCFS. However, it is not sufficient to outperform a better sorting of the waiting queue, notably the sorting performed by SAF, F2, and Lin.

Although the Lin function exhibited lower performance levels compared to the F2 and SAF policies, in most cases, it resulted in lower average bounded slowdowns and narrower differences between the extreme quartiles across most

workloads. This highlights its favorable scheduling performance, notwithstanding its simplicity.

Moreover, these results show that regression-obtained scheduling heuristics (F2 and Lin) can provide stable and efficient scheduling performances over a wide evolution of HPC platforms and workloads. For instance, the Mustang trace comprises a 5-year workload evolution (from 2011 to 2016), and F2 and Lin performed well in all workloads samples from Mustang. Both F2 and Lin were created once, and they did not need to be readjusted over time to provide efficient scheduling performances.

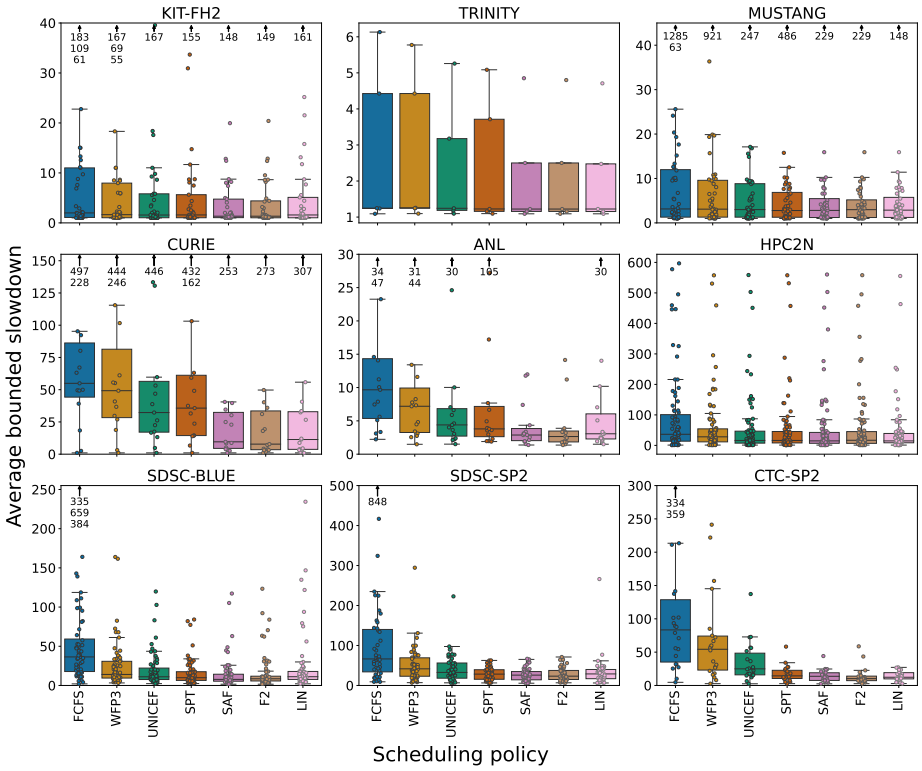

Fig. 5: Computed average bounded slowdown for different scheduling policies. Experiments based on user-estimated processing times, with aggressive backfilling algorithm.

6 Conclusions and Future Work

The complexity of online parallel scheduling algorithms has always been a negative factor for their deployment in real scenarios. Scheduling heuristics that sorts

the jobs in a waiting queue scheduling policies were always the scheduling algorithm of choice in real deployments. This choice is arguably due to the simplicity and interpretability of scheduling heuristics.

Machine learning (ML) methods, particularly regression methods, can be a compelling approach to creating scheduling heuristics that are simple, transparent, and efficient. However, an unexplored aspect of such methods is determining how much scheduling performance can be gained by increasing the complexity of the heuristics. Another uncharted aspect is the long-term stability of these heuristics since both workloads and HPC platforms are constantly changing.

In this work, we conducted an experimental analysis to investigate how the performance of regression-obtained scheduling heuristics can be improved. Initially, we created a data set of scheduling observations by conducting quick scheduling simulations for various scheduling contexts. Subsequently, we utilized a multiple linear regression model to learn the optimal parameters of the proposed functions. Finally, we evaluated the scheduling performance of these heuristics as scheduling policies.

Our study revealed that the scheduling performance of the aforementioned policies highly depends on the parameters (coefficients) obtained from the linear regression. We further discovered a multicollinearity effect that had a detrimental impact on the stability of the coefficients of the scheduling policies, ultimately leading to unsatisfactory scheduling outcomes.

To achieve efficient scheduling policies, we had to keep the functions simple. Specifically, a linear combination (Lin) of three terms: p, q, and r. We found that other policies containing more derivative terms resulted in instabilities in the obtained parameters and poorer scheduling performance. While removing terms to reduce multicollinearity would result in functions similar to the Lin policy, we found the Lin policy attractive as it combines three simple known scheduling policies. The coefficients obtained through multiple linear regression weigh the relevance of these three policies. Given these factors, we decided not to mitigate multicollinearity in this work by removing terms in the other policies.

Then, we validated the performance of regression-obtained scheduling heuristics on a diverse set of HPC platforms and workloads. We performed an experimental simulation campaign with workload data spanning 19 years, encompassing numerous platform and workload generations. We assessed the scheduling efficiency of the Lin and F2 functions by comparing them against other scheduling policies. Our results demonstrate that the regression-obtained scheduling heuristics can deliver stable and efficient scheduling performance across numerous HPC platforms and workloads without the need to readjust their coefficients over time.

We also observed that regression-obtained scheduling heuristics that combine the characteristics p, q, and r in more unusual ways, like the policy F2 [6], possibly lead to slightly better performances than Lin. This slight performance increase comes with the drawback of lesser interpretability, as it is harder to interpret why the square roots and logarithms in F2 lead to better scheduling.

We can obtain better policies than Lin by increasing the complexity of combining p, q, and r, but we do not foresee significant performance gains.

For future work, we plan to enhance the proposed regression method by incorporating new platform and workload features, such as platform utilization, remaining time of processing jobs, and the time of day. We plan to extend the work by searching for possible scheduling heuristics while considering the level of multicollinearity among the features. Moreover, we plan to extend this work to consider the jobs' power demand as a feature, with the objective of reaching similar levels of scheduling performance with the lowest possible overall power consumption of the platform.

Acknowledgement

This research was supported by the EuroHPC EU Regale project (g.a. 956560), São Paulo Research Foundation (FAPESP, grants 19/26702-8 and 22/06906-0), and the MIAI Grenoble-Alpes institute (ANR project number 19-P3IA-0003).

References

1. Akinwande, M.O., Dikko, H.G., Samson, A.: Variance inflation factor: As a condition for the inclusion of suppressor variable(s) in regression analysis. Open Journal of Statistics **05**, 754–767 (2015). https://doi.org/10.4236/ojs.2015.57075
2. Alin, A.: Multicollinearity. Wiley Interdisciplinary Reviews: Computational Statistics **2**, 370–374 (5 2010). https://doi.org/10.1002/wics.84
3. Amvrosiadis, G., Kuchnik, M., Park, J.W., Cranor, C., Ganger, G.R., Moore, E., DeBardeleben, N.: The atlas cluster trace repository. Usenix Mag **43**(4) (2018)
4. Baker, B.S., Coffman, Jr, E.G., Rivest, R.L.: Orthogonal packings in two dimensions. SIAM Journal on computing **9**(4), 846–855 (1980)
5. Bougeret, M., Dutot, P., Jansen, K., Otte, C., Trystram, D.: Approximation algorithms for multiple strip packing. In: Approximation and Online Algorithms, 7th International Workshop, WAOA 2009, Copenhagen, Denmark, September 10-11, 2009. Revised Papers. pp. 37–48 (2009). https://doi.org/10.1007/978-3-642-12450-1_4, https://doi.org/10.1007/978-3-642-12450-1_4
6. Carastan-Santos, D., de Camargo, R.Y.: Obtaining dynamic scheduling policies with simulation and machine learning. In: Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis. pp. 32:1–32:13. SC '17, ACM, New York, NY, USA (2017). https://doi.org/10.1145/3126908.3126955, http://doi.acm.org/10.1145/3126908.3126955
7. Carastan-Santos, D., De Camargo, R.Y., Trystram, D., Zrigui, S.: One can only gain by replacing easy backfilling: A simple scheduling policies case study. In: 2019 19th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGRID). pp. 1–10 (2019). https://doi.org/10.1109/CCGRID.2019.00010
8. Carroll, R., Ruppert, D.: Transformation and Weighting in Regression. Chapman & Hall/CRC Monographs on Statistics & Applied Probability, Taylor & Francis (1988), https://books.google.com.br/books?id=I5rGEPJd57AC

9. Casanova, H., Giersch, A., Legrand, A., Quinson, M., Suter, F.: Versatile, scalable, and accurate simulation of distributed applications and platforms. Journal of Parallel and Distributed Computing **74**(10), 2899–2917 (Jun 2014)
10. Fan, Y., Lan, Z., Childers, T., Rich, P., Allcock, W., Papka, M.E.: Deep reinforcement agent for scheduling in hpc. In: 2021 IEEE International Parallel and Distributed Processing Symposium (IPDPS). pp. 807–816 (2021). https://doi.org/10.1109/IPDPS49936.2021.00090
11. Feitelson, D.G.: Metrics for parallel job scheduling and their convergence. In: Workshop on Job Scheduling Strategies for Parallel Processing. pp. 188–205. Springer (2001)
12. Feitelson, D.G., Rudolph, L., Schwiegelshohn, U., Sevcik, K.C., Wong, P.: Theory and practice in parallel job scheduling. In: Workshop on Job Scheduling Strategies for Parallel Processing. pp. 1–34. Springer (1997)
13. Feitelson, D.G., Tsafrir, D., Krakov, D.: Experience with using the parallel workloads archive. Journal of Parallel and Distributed Computing **74**(10), 2967–2982 (2014)
14. Gaussier, E., Glesser, D., Reis, V., Trystram, D.: Improving backfilling by using machine learning to predict running times. In: Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis. pp. 64:1–64:10. SC '15, ACM, New York, NY, USA (2015). https://doi.org/10.1145/2807591.2807646, http://doi.acm.org/10.1145/2807591.2807646
15. Georgiou, Y.: Resource and Job Management in High Performance Computing. Ph.D. thesis, PhD Thesis, Joseph Fourier University, France (2010)
16. Hurink, J.L., Paulus, J.J.: Online algorithm for parallel job scheduling and strip packing. In: International Workshop on Approximation and Online Algorithms. pp. 67–74. Springer (2007)
17. Legrand, A., Trystram, D., Zrigui, S.: Adapting batch scheduling to workload characteristics: What can we expect from online learning? In: 2019 IEEE International Parallel and Distributed Processing Symposium (IPDPS). pp. 686–695 (2019). https://doi.org/10.1109/IPDPS.2019.00077
18. Lelong, J., Reis, V., Trystram, D.: Tuning EASY-Backfilling Queues. In: 21st Workshop on Job Scheduling Strategies for Parallel Processing. 31st IEEE International Parallel & Distributed Processing Symposium, Orlando, United States (May 2017), https://hal.archives-ouvertes.fr/hal-01522459
19. Li, J., Zhang, X., Han, L., Ji, Z., Dong, X., Hu, C.: Okcm: improving parallel task scheduling in high-performance computing systems using online learning. The Journal of Supercomputing **77**(6), 5960–5983 (2021)
20. Lublin, U., Feitelson, D.G.: The workload on parallel supercomputers: Modeling the characteristics of rigid jobs. J. Parallel Distrib. Comput. **63**(11), 1105–1122 (nov 2003). https://doi.org/10.1016/S0743-7315(03)00108-4, https://doi.org/10.1016/S0743-7315(03)00108-4
21. Meuer, H., Strohmaier, E., Dongarra, J., Simon, H., Meuer, M.: TOP500 Supercomputer Sites. https://www.top500.org/ (2023), online; last access 21 february 2023
22. Mu'alem, A.W., Feitelson, D.G.: Utilization, predictability, workloads, and user runtime estimates in scheduling the IBM SP2 with backfilling. IEEE Transactions on Parallel and Distributed Systems **12**(6), 529–543 (2001)
23. Pinedo, M.L.: Scheduling: theory, algorithms, and systems. Springer (2016)

24. Rodrigo, G.P., Östberg, P.O., Elmroth, E., Antypas, K., Gerber, R., Ramakrishnan, L.: Towards understanding hpc users and systems: a nersc case study. Journal of Parallel and Distributed Computing **111**, 206–221 (2018)
25. Tang, W., Lan, Z., Desai, N., Buettner, D.: Fault-aware, utility-based job scheduling on BlueGene/P systems. In: Cluster Computing and Workshops, 2009. CLUSTER'09. IEEE International Conference on. pp. 1–10. IEEE (2009)
26. Virtanen, P., Gommers, R., Oliphant, T.E., Haberland, M., Reddy, T., Cournapeau, D., Burovski, E., Peterson, P., Weckesser, W., Bright, J., van der Walt, S.J., Brett, M., Wilson, J., Millman, K.J., Mayorov, N., Nelson, A.R.J., Jones, E., Kern, R., Larson, E., Carey, C.J., Polat, İ., Feng, Y., Moore, E.W., VanderPlas, J., Laxalde, D., Perktold, J., Cimrman, R., Henriksen, I., Quintero, E.A., Harris, C.R., Archibald, A.M., Ribeiro, A.H., Pedregosa, F., van Mulbregt, P., SciPy 1.0 Contributors: SciPy 1.0: Fundamental Algorithms for Scientific Computing in Python. Nature Methods **17**, 261–272 (2020). https://doi.org/10.1038/s41592-019-0686-2
27. Ye, D., Han, X., Zhang, G.: Online multiple-strip packing. Theoretical Computer Science **412**(3), 233 – 239 (2011). https://doi.org/https://doi.org/10.1016/j.tcs.2009.09.029, http://www.sciencedirect.com/science/article/pii/S0304397509006896, combinatorial Optimization and Applications
28. Ye, D., Zhang, G.: On-line scheduling of parallel jobs in a list. Journal of scheduling **10**(6), 407–413 (2007)
29. Zhang, D., Dai, D., He, Y., Bao, F.S., Xie, B.: Rlscheduler: An automated hpc batch job scheduler using reinforcement learning. In: SC20: International Conference for High Performance Computing, Networking, Storage and Analysis. pp. 1–15 (2020). https://doi.org/10.1109/SC41405.2020.00035
30. Zhuk, S.: Approximate algorithms to pack rectangles into several strips. Discrete Mathematics and Applications dma **16**(1), 73–85 (2006)
31. Zrigui, S., de Camargo, R.Y., Legrand, A., Trystram, D.: Improving the performance of batch schedulers using online job runtime classification. Journal of Parallel and Distributed Computing **164**, 83–95 (2022). https://doi.org/https://doi.org/10.1016/j.jpdc.2022.01.003, https://www.sciencedirect.com/science/article/pii/S0743731522000090