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 to 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 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 is a step forward from the work of Carastan-Santos and de Camargo [6]. We used the same methodology of exploiting simulations to gather scheduling knowledge, but extended it by creating a collection of scheduling heuristics based on empirical scheduling data. These heuristics rely on general polynomials of job characteristics as templates, which we hypothesize to be both efficient and more explainable in the scheduling context than the functions defined by Carastan-Santos and de Camargo.

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 adapted 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 in terms of the average bounded slowndown performance metric. 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 in Section 3 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 minimizes the height among the used strips. Such a 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

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

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.

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.

Legrand *et al.* [17] explored a larger class of mixed policies by combining various features extracted from the jobs in a linear expression, and by determining optimal weights for each feature. On the other hand, Carastan-Santos and de Camargo [6] introduced regression methods to derive clear and interpretable scheduling heuristics.

Our work goes beyond the approach of Carastan-Santos and de Camargo. While their heuristics relied on features based on terms such as square roots and logarithms, which can be challenging to interpret in the context of scheduling, our work is based on a functional form generation procedure that creates heuristics using simple polynomials of the job characteristics. This approach takes into account increasing levels of complexity and derivative features.

3 Background

3.1 Online Parallel Job Scheduling

In a usual 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]. The RJMS registers information about the jobs in a log file, often shared using the *Standard Workload Format* (SWF) [13]. Moreover, the online parallel job scheduling problem is a key issue that the RJMS must address in order to effectively manage platform resources and applications.

The online parallel job scheduling problem involves the decision-making process of determining the optimal time and machine allocation to process parallel jobs, which arrive at any given time, on a set of n homogeneous machines that are connected by an arbitrary interconnection topology. The primary goal of this

decision-making process is to maximize performance objective(s) related to the efficient utilization of platform resources.

A job, denoted by j, is commonly described as a workload with specific data, including (i) the estimated processing time of the job ($\tilde{p}_j$), typically provided by the user, (ii) the number of processors required to execute the job (q_j), and (iii) the time at which the job was submitted to the platform (r_j, also known as the release time). This characterization of jobs is often used in scheduling algorithms, where jobs are treated as independent entities requiring fixed amounts of resources for processing.

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).

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 of 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. The lower the value of $f(j)$, the more priority a job j has. Ideally, the decision-making process 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 become 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

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

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

As previously mentioned, we have adapted the method proposed by Carastan-Santos and Camargo [6] in our work. We retained their approach of acquiring scheduling knowledge through simulations of sets of jobs executed under different conditions, represented by an approximate scheduling problem, termed as *proxy scheduling problem*, similar yet simpler than online parallel job scheduling. However, we have made modifications to the models defined in the regression step. We incorporated the knowledge obtained from the simulations into a multiple linear regression model and transformed this knowledge into polynomial functions capable of determining the priority of the jobs in the queue based on their characteristics. Our main hypothesis is that these functions represent effective scheduling policies for the online parallel job scheduling problem.

4.1 Simulation strategy

The proxy scheduling problem differs from the online parallel job scheduling problem (described in Section 3) in a few key ways. First, the proxy problem have access to perfect information about the job processing time (i.e., $\tilde{p}_j = p_j$). Second, the proxy problem is an offline scheduling problem, which means that all the information regarding the jobs' characteristics, including p, q, and r, is known in advance. Finally, the number of jobs arriving in the proxy problem is finite. The purpose of defining this simplified scheduling problem is to enable quick simulations that are similar to the target problem.

We define a proxy scheduling problem as the scheduling of a set of jobs Q in an HPC platform with m interconnected processors, while another set of jobs S is currently being processed. This configuration models the waiting queue (Q) and the initial state of the platform (S). Job sets S and Q are generated from a job log file, or trace, denoted by N. A subtrace $M \subset N$, consisting of $|S| + |Q|$ jobs, is randomly selected from N, where the first $|S|$ jobs from M belong to S and the remaining $|Q|$ jobs from M belong to 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.

We proceed by randomly generating permutations Q^* of the set Q, where each Q^* is a simulation branch. For each permutation Q^*, we simulate the scheduling of the jobs in Q in the order specified by Q^*. It is important to note that the number of possible permutations of Q grows exponentially with the size of Q, and the number of sample permutations required to adequately represent the permutation space increases rapidly as well. The use of a proxy scheduling problem, as discussed previously, is necessary to ensure that this simulation branching strategy can be executed within a feasible amount of time for a fixed and small number of jobs in Q, as detailed in Section 5.

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 $j \in Q$:

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

The score represents the impact 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 lower *score* values can have a positive impact on the overall average slowdown when they are executed first. 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).

To generate a distribution of scores, denoted as $score(p, q, r)$, we replicated the aforementioned simulation strategy with multiple samples of job set pairs (S, Q). This distribution represents the primary outcome of the simulations. The idea is that the scheduler of an HPC system could prioritize a job from the queue with the smallest $score(p, q, r)$ value for the corresponding (p, q, r) values.

4.2 Creating regression-based scheduling heuristics

Regression methods can be employed on the scores distribution to obtain a more generalized and smoother representation. This representation is in the form of functions that can be used to prioritize jobs on an HPC platform. Carastan-Santos and de Camargo's approach utilized features such as square roots and logarithms, which can be difficult to interpret in the context of scheduling. In contrast, the approach proposed in this work explores polynomial-based functions.

Let J denote the set of all jobs present in the scores distribution, and let $score(j)$ be the computed value of score for each job $j \in J$. The problem at hand is to determine functions $f(p_j, q_j, r_j)$ that provide an accurate approximation of the score values for all jobs in J. To accomplish this, we have modified the original strategy proposed by Carastan-Santos and de Camargo [6]. Specifically,

we have defined a family of functions $\mathcal{F}$, consisting of 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.

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 have defined a function family comprising four functions, whose vectors $\mathbf{x}$ are displayed in Table 1. Each function works as a template for deriving the scheduling heuristics, and each element in the vectors (also known as basis functions) are monomials of the jobs' characteristics.

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 $= pq$). We chose these functions for the following reasons:

- 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.

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).

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

5 Results and Discussion

In this section, we present the main results of our research. Firstly, we generated three distributions of scores by employing the simulation strategy presented in Section 4.1. The traces were derived from one synthetic and two real-world workloads. Subsequently, we performed multiple linear regression to obtain the optimal parameters for the polynomial functions, taking into account the different score distributions. Finally, we analyzed the relationship between the size of the polynomials and the scheduling effectiveness, as well as the robustness of the regression-obtained heuristics over the long term.

5.1 Simulation-based approach for extracting scheduling knowledge

To generate the scores distribution, we considered an HPC platform composed by homogeneous processors, simulated using the SimGrid [9] simulation framework. The traces used to construct the characteristics of the jobs, both synthetic and real-world, were obtained from the Parallel Workloads Archive [13]. Finally, we employed sets of 16 and 32 jobs for the sets S and Q, respectively.

To construct the synthetic workload, we used the Lublin and Feitelson [20] model (with 256 nodes), which is capable of representing the geometry of jobs (p and q) as well as including a daily cycle arrival pattern. Additionally, we utilized the Cornell Theory Center (CTC) IBM SP2 log, which contains 11 months of records from the 512-node IBM SP2, and the San Diego Supercomputer Center (SDSC) Blue Horizon log, a 144-node IBM SP covering over two years of production use.

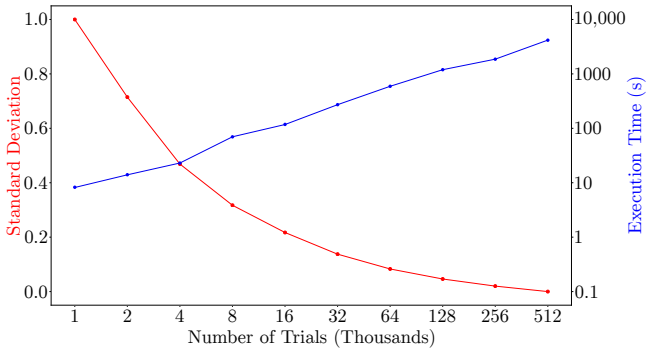

Fig. 1: The figure, adapted from [6], shows, in red, the normalized standard deviations of the trial score distributions obtained with different numbers of trials, and in blue, the execution time in the chosen instance in logarithmic scale.

Table 2: Mean Absolute Error (MAE) of the functions for the different workloads used in the training step.

Functions	MAEs		
	Synthetic	CTC-SP2	SDSC-Blue
Lin	$4.5 \cdot 10^{-3}$	$3.6 \cdot 10^{-3}$	$12.1 \cdot 10^{-3}$
Qdr	$4.5 \cdot 10^{-3}$	$3.7 \cdot 10^{-3}$	$13.9 \cdot 10^{-3}$
Cub	$5.4 \cdot 10^{-3}$	$5.9 \cdot 10^{-3}$	$3.3 \cdot 10^{-3}$
Qua	$7.2 \cdot 10^{-3}$	$36.2 \cdot 10^{-3}$	$14.8 \cdot 10^{-3}$

The simulations were carried out on an Amazon EC2 instance equipped with a second-generation AMD EPYC 7002 processor that can reach up to 3.3 GHz frequencies, 8 virtual CPU cores, and 16GB of RAM. In obtaining the scores distribution, we had to determine the maximum number of trials we could execute with a pair (S, Q) on this instance. This was necessary to achieve a balance between precision in calculating the scores distributions and efficient execution time.

Figure 1 illustrates the normalized standard deviation of calculating the scores as a function of the number of trials and the corresponding processing times for one pair (S, Q) executed on the instance. After considering these factors, we decided to use 256 thousand trials as it provided accurate calculations of the scores distribution within an acceptable processing time and budget.

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

To address this question, we employed multiple linear regression to compute the optimal parameters of the heuristics derived from the model functions for var-

ious score distributions. The synthetic score distribution encompasses 14,081 job characteristics linked to their corresponding score values. On the other hand, the distribution obtained from the CTC SP2 and SDSC-Blue workloads comprises 3,745 and 3,201 entries, respectively. Moreover, we used the SciPy's `curve_fit` [26] method to perform the multiple linear regression.

Table 3: Optimal coefficients of the four functions obtained by regression using the synthetic workload to generate the distribution of scores and the Variance Inflation Factors (VIFs) of each feature.

Coefficients	Regression-obtained optimal coefficients				Vector x	VIF			
	Lin	Qdr	Cub	Qua		Lin	Qdr	Cub	Qua
θ_0	$3.24 \cdot 10^{-2}$	$3.70 \cdot 10^{-2}$	$3.33 \cdot 10^{-2}$	$4.83 \cdot 10^{-2}$	1	–	–	–	–
θ_1	$1.15 \cdot 10^{-7}$	$2.65 \cdot 10^{-7}$	$2.83 \cdot 10^{-7}$	$-6.42 \cdot 10^{-7}$	p	1.3	3.7	12.1	36.1
θ_2	$2.61 \cdot 10^{-5}$	$-3.05 \cdot 10^{-5}$	$8.71 \cdot 10^{-5}$	$-2.40 \cdot 10^{-4}$	q	1.3	9.6	35.5	99.0
θ_3	$-1.57 \cdot 10^{-7}$	$-3.96 \cdot 10^{-7}$	$-5.83 \cdot 10^{-7}$	$-6.50 \cdot 10^{-7}$	r	1.2	6.0	21.8	58.6
θ_4	–	$-3.04 \cdot 10^{-12}$	$-6.33 \cdot 10^{-14}$	$1.66 \cdot 10^{-11}$	p^2	–	2.6	41.8	338.5
θ_5	–	$1.17 \cdot 10^{-7}$	$-5.06 \cdot 10^{-7}$	$2.41 \cdot 10^{-6}$	q^2	–	8.3	275.0	2835.5
θ_6	–	$2.75 \cdot 10^{-12}$	$6.78 \cdot 10^{-12}$	$8.81 \cdot 10^{-12}$	r^2	–	4.8	82.1	620.8
θ_7	–	$6.77 \cdot 10^{-10}$	$-6.02 \cdot 10^{-10}$	$1.14 \cdot 10^{-8}$	pq	–	3.9	49.4	295.3
θ_8	–	–	$-7.55 \cdot 10^{-18}$	$-2.03 \cdot 10^{-16}$	p^3	–	–	20.8	961.6
θ_9	–	–	$7.05 \cdot 10^{-10}$	$-9.52 \cdot 10^{-9}$	q^3	–	–	147.3	10491.8
θ_{10}	–	–	$-2.14 \cdot 10^{-17}$	$-4.67 \cdot 10^{-17}$	r^3	–	–	34.6	1387.1
θ_{11}	–	–	$-2.72 \cdot 10^{-14}$	$9.29 \cdot 10^{-15}$	p^2q	–	–	9.8	393.1
θ_{12}	–	–	$9.52 \cdot 10^{-12}$	$-7.94 \cdot 10^{-11}$	pq^2	–	–	30.1	1032.9
θ_{13}	–	–	–	$9.34 \cdot 10^{-22}$	p^4	–	–	–	322.7
θ_{14}	–	–	–	$1.36 \cdot 10^{-11}$	q^4	–	–	–	3572.5
θ_{15}	–	–	–	$9.84 \cdot 10^{-23}$	r^4	–	–	–	373.6
θ_{16}	–	–	–	$-9.07 \cdot 10^{-19}$	p^3q	–	–	–	63.8
θ_{17}	–	–	–	$1.89 \cdot 10^{-16}$	p^2q^2	–	–	–	125.0
θ_{18}	–	–	–	$1.55 \cdot 10^{-13}$	pq^3	–	–	–	457.2

The mean absolute error (MAE) values obtained by the multiple linear regression for the heuristics Lin, Qdr, Cub, and Qua are reported in Table 2. It can be observed that the MAE values are relatively small and close to each other for the same function, regardless of the workload used. This shows that the functions appropriately describe the distribution of scores used to build the models.

Furthermore, the optimal coefficients of the heuristics obtained by regression for each of the functions are illustrated in Tables 3, 4, and 5. However, we observed instabilities in the features' coefficients of Qdr, Cub, and Qua. These instabilities were observed through changes in the sign of the coefficients of a particular feature, which carry implications for job scheduling. In the case of synthetic workload, the Lin function prioritizes jobs with lower q values (posi-

Table 4: Optimal coefficients of the four functions obtained by regression using the CTC-SP2 workload to generate the distribution of scores and the Variance Inflation Factors (VIFs) of each feature.

Coefficients	Regression-obtained optimal coefficients				Vector x	VIF			
	Lin	Qdr	Cub	Qua		Lin	Qdr	Cub	Qua
θ_0	$3.11 \cdot 10^{-2}$	$2.86 \cdot 10^{-2}$	$2.42 \cdot 10^{-2}$	$6.09 \cdot 10^{-2}$	1	–	–	–	–
θ_1	$3.00 \cdot 10^{-8}$	$2.00 \cdot 10^{-7}$	$1.18 \cdot 10^{-6}$	$-3.13 \cdot 10^{-6}$	p	1.3	17.2	85.7	282.4
θ_2	$5.42 \cdot 10^{-5}$	$2.27 \cdot 10^{-5}$	$-1.72 \cdot 10^{-4}$	$8.73 \cdot 10^{-5}$	q	1.1	5.5	18.6	43.2
θ_3	$-6.41 \cdot 10^{-8}$	$1.81 \cdot 10^{-7}$	$-1.37 \cdot 10^{-8}$	$-2.40 \cdot 10^{-6}$	r	1.4	9.2	31.6	79.4
θ_4	–	$-3.14 \cdot 10^{-12}$	$-3.50 \cdot 10^{-11}$	$1.42 \cdot 10^{-10}$	p^2	–	15.2	483.3	5148.2
θ_5	–	$-2.08 \cdot 10^{-8}$	$1.66 \cdot 10^{-6}$	$-8.57 \cdot 10^{-6}$	q^2	–	4.1	114.1	709.1
θ_6	–	$-4.19 \cdot 10^{-12}$	$4.57 \cdot 10^{-12}$	$1.86 \cdot 10^{-10}$	r^2	–	7.2	141.9	972.4
θ_7	–	$8.50 \cdot 10^{-10}$	$3.89 \cdot 10^{-9}$	$1.49 \cdot 10^{-8}$	pq	–	1.5	23.7	215.7
θ_8	–	–	$2.97 \cdot 10^{-16}$	$-2.46 \cdot 10^{-15}$	p^3	–	–	201.3	11733.9
θ_9	–	–	$-3.91 \cdot 10^{-9}$	$6.03 \cdot 10^{-8}$	q^3	–	–	60.4	2059.7
θ_{10}	–	–	$-1.02 \cdot 10^{-16}$	$-4.92 \cdot 10^{-15}$	r^3	–	–	62.4	2016.0
θ_{11}	–	–	$-3.33 \cdot 10^{-14}$	$-6.88 \cdot 10^{-13}$	p^2q	–	–	16.6	844.4
θ_{12}	–	–	$-6.33 \cdot 10^{-12}$	$2.37 \cdot 10^{-10}$	pq^2	–	–	9.1	339.8
θ_{13}	–	–	–	$1.47 \cdot 10^{-20}$	p^4	–	–	–	2784.2
θ_{14}	–	–	–	$-1.12 \cdot 10^{-10}$	q^4	–	–	–	617.3
θ_{15}	–	–	–	$4.07 \cdot 10^{-20}$	r^4	–	–	–	474.2
θ_{16}	–	–	–	$6.11 \cdot 10^{-18}$	p^3q	–	–	–	378.6
θ_{17}	–	–	–	$-7.14 \cdot 10^{-16}$	p^2q^2	–	–	–	99.5
θ_{18}	–	–	–	$-9.75 \cdot 10^{-13}$	pq^3	–	–	–	69.9

tive coefficients), whereas the Qdr function prioritizes jobs with higher q values (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].

Figure 2 demonstrates the impact of varying the values of p and q on the normalized *score* values of the functions Lin, Qdr, Cub and Qua, while holding r constant, in order to illustrate these instabilities. Darker regions in the figure indicate higher job priorities. The behavior of all functions is distinct, independent of the type of workload considered. The Lin function prioritizes jobs with lower values of p and q, which is similar to the Shortest Area First (SAF) scheduling policy, widely known for its efficiency [7]. The Qdr, Cub, and Qua functions give varying degrees of priority to certain regions that are less interpretable.

To assess the effectiveness of the obtained functions as scheduling policies, we conducted simulations of parallel job scheduling with the same workload used

Table 5: Optimal coefficients of the four functions obtained by regression using the SDSC-Blue workload to generate the distribution of scores and the Variance Inflation Factors (VIFs) of each feature.

Coefficients	Regression-obtained optimal coefficients				Vector x	VIF			
	Lin	Qdr	Cub	Qua		Lin	Qdr	Cub	Qua
θ_0	$1.83 \cdot 10^{-2}$	$1.39 \cdot 10^{-2}$	$3.34 \cdot 10^{-2}$	$2.27 \cdot 10^{-2}$	1	–	–	–	–
θ_1	$8.88 \cdot 10^{-8}$	$-1.47 \cdot 10^{-7}$	$5.97 \cdot 10^{-8}$	$1.58 \cdot 10^{-6}$	p	1.1	8.1	32.3	81.2
θ_2	$3.05 \cdot 10^{-5}$	$-8.40 \cdot 10^{-5}$	$3.09 \cdot 10^{-5}$	$1.07 \cdot 10^{-4}$	q	1.1	5.3	22.9	53.1
θ_3	$-6.87 \cdot 10^{-8}$	$-4.02 \cdot 10^{-7}$	$-4.79 \cdot 10^{-7}$	$-1.77 \cdot 10^{-6}$	r	1.2	5.4	15.3	39.6
θ_4	–	$5.59 \cdot 10^{-14}$	$1.79 \cdot 10^{-12}$	$-4.00 \cdot 10^{-11}$	p^2	–	6.7	175.1	1479.2
θ_5	–	$6.92 \cdot 10^{-8}$	$-1.02 \cdot 10^{-7}$	$-1.91 \cdot 10^{-7}$	q^2	–	4.6	140.6	1109.7
θ_6	–	$5.77 \cdot 10^{-12}$	$1.12 \cdot 10^{-11}$	$1.04 \cdot 10^{-10}$	r^2	–	4.6	59.1	393.1
θ_7	–	$5.50 \cdot 10^{-10}$	$-6.67 \cdot 10^{-10}$	$-6.55 \cdot 10^{-9}$	pq	–	1.7	28.2	164.2
θ_8	–	–	$-1.51 \cdot 10^{-17}$	$3.93 \cdot 10^{-16}$	p^3	–	–	79.7	3719.5
θ_9	–	–	$7.15 \cdot 10^{-11}$	$1.21 \cdot 10^{-10}$	q^3	–	–	73.6	4177.1
θ_{10}	–	–	$-9.27 \cdot 10^{-17}$	$-2.26 \cdot 10^{-15}$	r^3	–	–	26.3	704.1
θ_{11}	–	–	$-6.38 \cdot 10^{-16}$	$1.25 \cdot 10^{-13}$	p^2q	–	–	13.1	465.1
θ_{12}	–	–	$1.08 \cdot 10^{-12}$	$4.03 \cdot 10^{-12}$	pq^2	–	–	8.2	453.6
θ_{13}	–	–	–	$-1.28 \cdot 10^{-21}$	p^4	–	–	–	987.8
θ_{14}	–	–	–	$-1.49 \cdot 10^{-14}$	q^4	–	–	–	1462.3
θ_{15}	–	–	–	$1.50 \cdot 10^{-20}$	r^4	–	–	–	156.1
θ_{16}	–	–	–	$-6.68 \cdot 10^{-19}$	p^3q	–	–	–	129.6
θ_{17}	–	–	–	$-4.72 \cdot 10^{-17}$	p^2q^2	–	–	–	63.7
θ_{18}	–	–	–	$3.65 \cdot 10^{-16}$	pq^3	–	–	–	192.9

Table 6: 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	$\sqrt{\tilde{p}_j} \cdot q_j + 2.56 \times 10^4 \cdot \log_{10}(r_j)$

in the score distribution generation, enabling us to gain deeper insights into the capabilities of these functions. From the workloads, we selected a subtrace consisting of a fifteen-day (15) job submission. The output of the scheduling experiments is the average bounded slowdown (Equation 2). Moreover, to enable a meaningful comparison, we included established scheduling policies, according to Table 6, alongside the obtained functions.

The online scheduling simulation involves jobs arriving at a centralized waiting queue, where a decision-making mechanism reschedules jobs in the queue following a scheduling policy when a new job arrives or a resource becomes

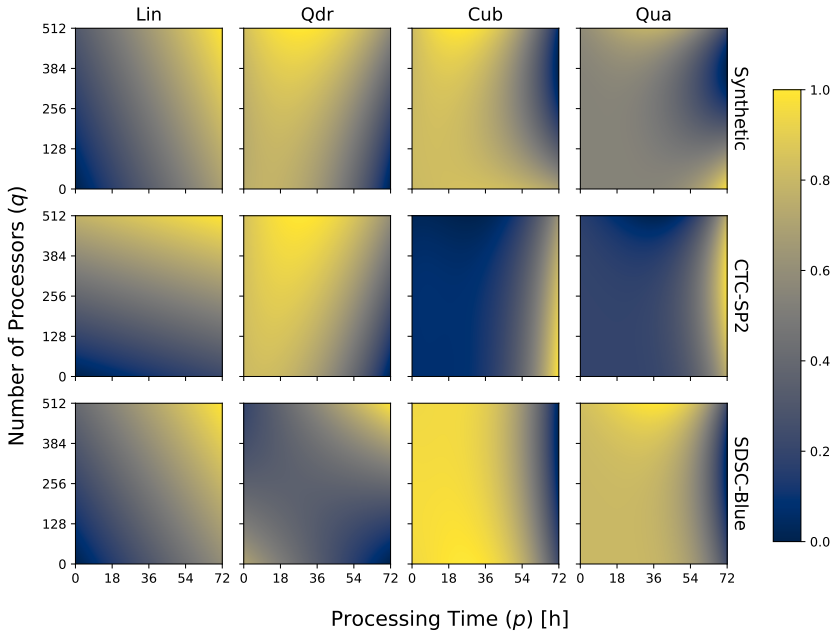

Fig. 2: Dependency of the Lin, Qdr, Cub, and Qua functions on the job characteristics p and q ($r = 300$) for the synthetic score distribution, as well as for the CTC SP2 and SDSC-Blue score distributions.

available. If the number of processors requested by a job is fewer than the total number of available processors, the scheduling mechanism reserves and makes the required processors inaccessible until the job is completed. If there are insufficient processors available, the decision-making process is delayed until one of the two rescheduling occurrences occurs. Job data such as $\tilde{p}$, q, and r are only available to the scheduling mechanism at the time of job arrival.

The average bounded slowdown for all simulations is presented in Figure 3. The results indicate that the more complex scheduling functions Qdr, Cub, and Qua perform worse than the established scheduling policies regardless of the workload used to generate them. Additionally, the performance of Qdr, Cub and Qua was as inefficient as FCFS and this further cemented the negative effects of their coefficients. Conversely, the Lin function demonstrated better results that are comparable to SAF, a well-known efficient scheduling policy. The reasons for the inefficiency of the Qdr, Cub, and Qua functions will be discussed further in the following section.

Multicollinearity results in poor scheduling heuristics The instability observed in the coefficients of Qdr, Cub, and Qua implies that these functions may be affected by multicollinearity [2]. Multicollinearity occurs when the features

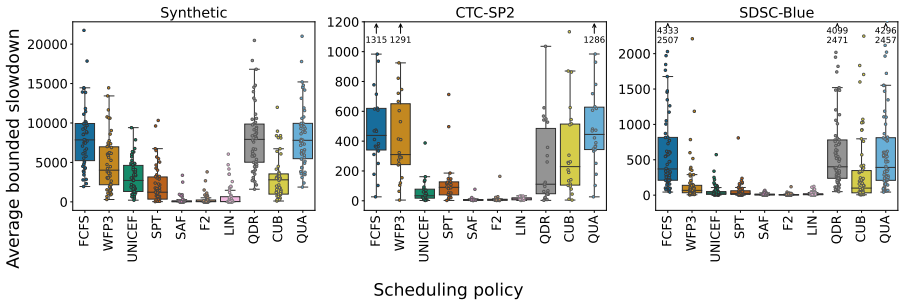

Fig. 3: Schedule performance results by scheduling jobs derived from the same workloads used in the simulation phase, and by incorporating actual processing time when making scheduling decisions.

used in a regression analysis are highly correlated with each other, which can lead to unstable and unreliable coefficient estimates. In our case, adding derivative features in combination with the original job characteristics p, q, and r (such as q^2, q^3, and q^4) may have contributed to multicollinearity. This phenomenon can affect the accuracy of the regression models and make their coefficients less interpretable.

To test for multicollinearity, we used the Variance Inflation Factor (VIF) [1] to examine the relationship between the individual features present in the functions Lin, Qdr, Cub, and Qua. With values ranging from 1 to infinity, the VIF measures the extent to which a feature is linearly related to the other features in the model. Specifically, a VIF of 1 indicates no correlation between one feature and the other features, whereas a VIF greater than 1 suggests some degree of correlation.

VIF's numerical value indicates how much the variance of a specific coefficient is inflated, with a larger VIF indicating a stronger association with other features. For example, a VIF of 1.9 indicates that the variance of a coefficient is 90% larger than expected if there was no multicollinearity with other features. In general, a VIF threshold of 5 or 10 is used to identify features with high multicollinearity, as values above these thresholds suggest that the features may be too closely related to each other to be included in the model.

The calculated Variance Inflation Factor (VIF) values for the functions obtained from three different workloads are presented in Tables 3, 4, and 5. The synthetic workload produced VIF values that were already quite high, particularly for the higher degree functions. However, the VIF values for the CTC-SP2 and SDSC-Blue workloads were even higher, with some functions having values greater than 10,000.

These results suggest that the addition of more derivative features in the models would lead to even larger VIFs, indicating the presence of multicollinearity. This problem might have a substantial influence on model accuracy and stability, potentially leading to overfitting and difficulty in interpreting the coefficients.

Hence, it is advisable to carefully evaluate the incorporation of extra features in developing novel scheduling heuristics for HPC platforms.

In the subsequent experiments, our attention was directed solely towards examining the Lin function due to its relatively lower level of multicollinearity when compared to the other functions. As there was no compelling reason to prefer a specific version of the Lin function, we proceeded to conduct the experiments using the version obtained from the synthetic workload.

5.3 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 6, 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 simulations by selecting subtraces consisting of a fifteen-day job submission, in two different scenarios: (i) scheduling using jobs' processing time estimates and (ii) scheduling using processing time estimates along 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.

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.

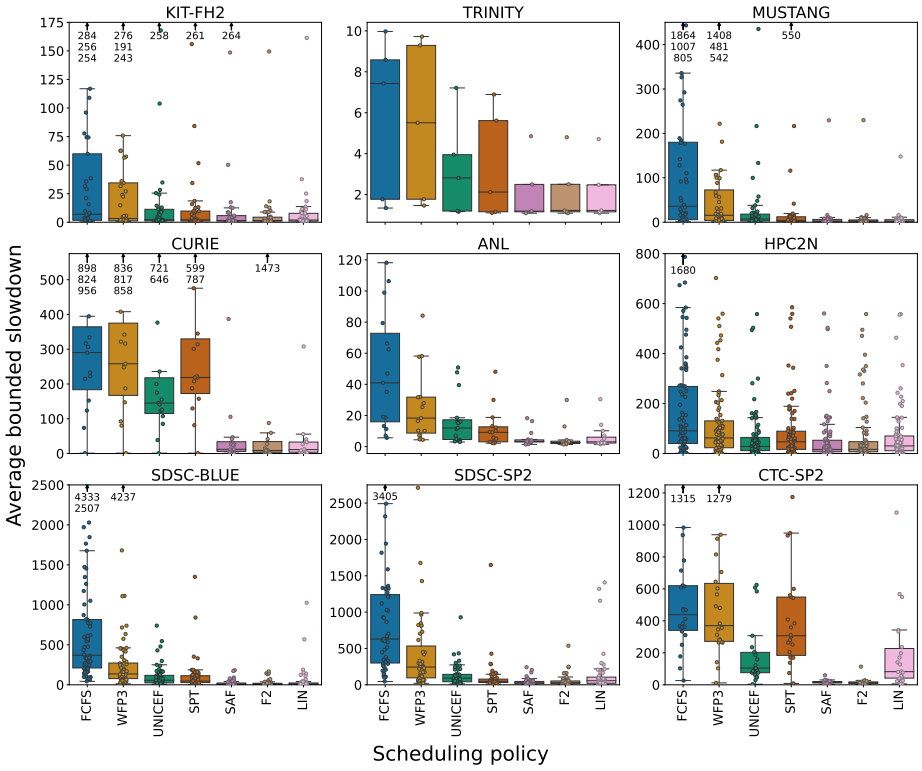

Fig. 4: Computed average bounded slowdown for different scheduling policies. Experiments based solely on user-estimated processing times.

Scheduling experiments using estimated processing times and aggressive backfilling We used an aggressive backfilling technique to reduce the platform's idleness in order to achieve a more realistic scenario. Figure 5 shows that backfilling improves noticeably the performance of suboptimal heuristics like FCFS. Nevertheless, it is not adequate to surpass a superior sorting of the waiting queue, particularly the sorting executed 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.

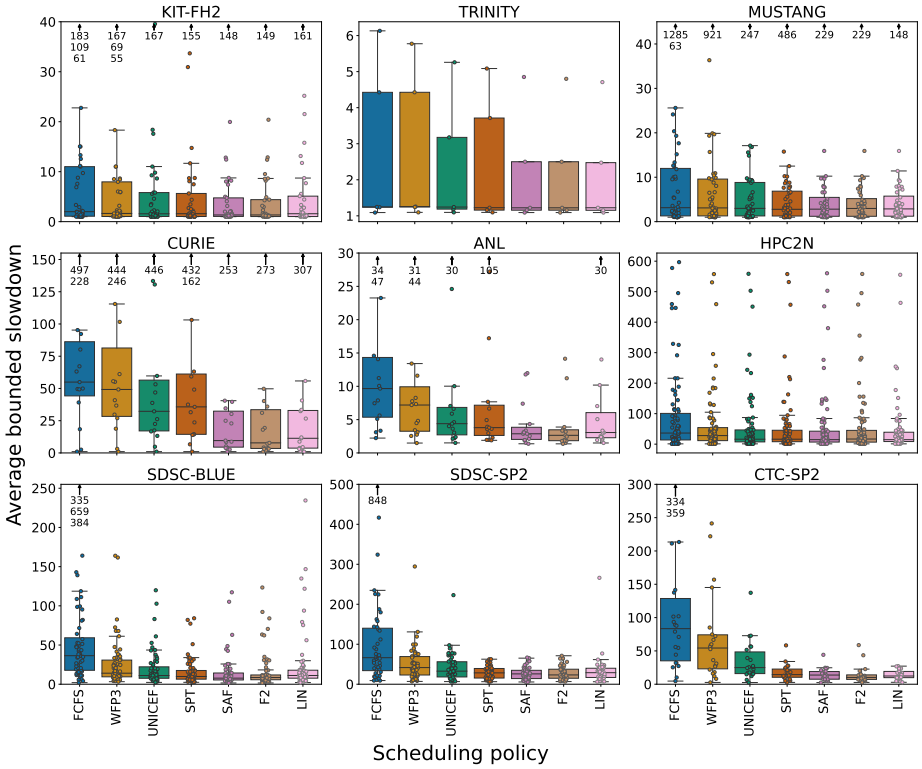

Fig. 5: Computed average bounded slowdown for different scheduling policies. Experiments based on user-estimated processing times, with aggressive backfilling algorithm.

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, three datasets were generated by conducting quick scheduling simulations for various scheduling contexts using a synthetic workload trace and the CTC-SP2 and SDSC-Blue workloads traces. These datasets were then used to train a multiple linear regression model to learn the optimal parameters of the proposed scheduling heuristics. Finally, we evaluated the performance of these heuristics as scheduling policies.

Our study revealed that the scheduling performance of the aforementioned heuristics 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.

In order to develop efficient scheduling heuristics, we found simple functions to be effective, specifically a linear combination of three terms: p, q, and r, denoted Lin. While more complex functions with additional derivative terms were considered, they resulted in instabilities in the obtained parameters and poorer scheduling performance. While removing terms would result in functions similar to the *Lin* policy, we found the *Lin* policy appealing because it combines three simple known scheduling policies, with the coefficients obtained through multiple linear regression weighing the relevance of these 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