On limits of Machine Learning techniques in the learning of scheduling policies

Student: Lucas de Sousa Rosa[1]
Advisers: Danilo Carastan-Santos[2]**, Alfredo Goldman**[1] **and Denis Trystram**[2]

[1] Institute of Mathematics and Statistics, Department of Computer Science
University of São Paulo (USP) — São Paulo/SP — Brazil

[2]Laboratoire d'Informatique de Grenoble, CNRS, Inria, Grenoble INP
Univ. Grenoble Alpes (UGA) — Grenoble/FR — France

roses.lucas@usp.br, danilo.carastan-dos-santos@inria.fr,

gold@ime.usp.br, denis.trystram@univ-grenoble-alpes.fr

Abstract. *This scientific initiation work explores the emerging relationship between managing resources on high-performance computing (HPC) platforms and the use of regression-derived scheduling heuristics to optimize performance. Recent research has shown that machine learning (ML) techniques can be used to generate scheduling heuristics that are simple and efficient. This work proposes an alternative approach using polynomial functions to generate scheduling heuristics. The simplest polynomial was found to be one of the most efficient heuristic. We also evaluated the resilience of the regression-derived heuristics over time. We published two papers in peer-reviewed national and international workshops (Qualis-B3/B4).*

1. Introduction

The rapid growth of consumer electronics, data centers, big data, and scientific research has resulted in an increasing demand for computing performance that is both fast and affordable [Shalf 2020]. High-performance computing (HPC) has become an integral tool for managing and analyzing massive amounts of data and solving complex computational problems in these areas. However, as HPC platforms continue to evolve, addressing the challenges of resource management, energy efficiency, and software complexity becomes more crucial to enable trans-petascale computing.

A common approach to managing resources on HPC platforms involves the use of a Resource and Job Management System (RJMS). Typically, HPC applications, referred to as jobs, are submitted to the RJMS for execution on the platform. The arrival of jobs in the RJMS queue is unpredictable, posing a substantial challenge for the RJMS to assign a priority order that optimizes one or more performance metrics.

The prioritization assignment problem is a complex and intricate task which has been defined in academic literature as the parallel online job scheduling problem [Brucker 2007]. A viable and compelling solution to this problem is to use scheduling heuristics to dictate the order of jobs in the queue. Scheduling heuristics can be defined as functions that take as input various job characteristics, including processing time, requested resources, wait time, and other relevant factors. These functions output a value that determines the priority of the job's execution, with First-Come-First-Served (FCFS) based functions being the most used in practice [Feitelson et al. 1997].

Furthermore, the use of machine learning (ML) in online job scheduling predominantly occurs in two specific scenarios: (i) to enhance scheduling through predicting the traits of jobs [Li et al. 2021, Zrigui et al. 2022], and (ii) to generate novel heuristics leveraging techniques such as non-linear regression [Carastan-Santos and de Camargo 2017] and evolutionary strategies [Legrand et al. 2019]. More recently, deep reinforcement learning methods are being investigated for the execution of online job scheduling [Fan et al. 2021, Zhang et al. 2020].

Regarding the second scenario, these studies have utilized ML techniques to leverage simulation and platform workload data in order to devise scheduling heuristics. These approaches aim to encode the scheduling knowledge obtained through simulations into regression-obtained scheduling heuristics that are both explainable and efficient. In this light, **this scientific initiation work was proposed in order to explore the emerging relationship between machine learning and scheduling optimization**.

This work is a step forward from the work of Carastan-Santos and de Camargo [Carastan-Santos and de Camargo 2017]. We kept the same strategy of collecting scheduling knowledge using simulations. However, we modified the functions used in the machine learning phase. We define polynomial functions of increasing complexity with the hypothesis that they would be both efficient and more explicable heuristics, in the scheduling context, than the functions defined in Carastan-Santos and de Camargo work.

During the machine learning phase, the results obtained indicated the presence of multicollinearity – a phenomenon in which more than two explanatory variables in a multiple regression model are highly linearly related – in the more complex polynomials, which leads to instability in the coefficients and inefficient schedules. On the other hand, we provided experimental evidence that the simplest polynomial proved to be an efficient scheduling heuristic.

We also evaluated the resilience of the regression-derived heuristics in light of the changing landscape of HPC platforms and workloads over time. Our results show that the regression-derived heuristics can generate effective schedules in all the contexts analyzed, without requiring adjustments over time.

This work was executed under a FAPESP Scientific Initiation scholarship[1], including a two-month internship[2] at the Grenoble Computer Science Laboratory, in France. The research culminated in the student's primary authorship of two publications; one in an international workshop [Rosa et al. 2023] (Qualis-B3, Core Rank B), and another in a national workshop [Rosa and Goldman 2022] (Qualis B4, CAPES), which received an

[1]`https://bv.fapesp.br/pt/bolsas/202861/sobre-os-limites-das-tecnicas-de-machine-learning-no-aprendizado-de-politicas-de-escalonamento/`

[2]`https://bv.fapesp.br/pt/bolsas/206621/avaliando-tecnicas-de-aprendizado-de-maquina-e-simulacoes-para-criar-heuristicas-de-escalonamento-ef/`

honorable mention.

We organized the remaining of this work as follows. In Section 2, we present the necessary background to introduce our research methodology in Section 3. Then, in Section 4, we present and discuss the results obtained. Finally, in Section 5, we present the concluding considerations.

2. Preliminary Background

We addressed the online scheduling problem of executing a group of simultaneous parallel jobs on an HPC platform. The resource requirements for these jobs are predetermined and unchangeable, also known as rigid jobs. The HPC platform is composed of a set of m homogeneous resources, which are connected by arbitrary interconnection topologies.

Jobs arrive in the platform's central queue in an online manner over time. A job, denoted as j, is characterized as a workload with specific data that includes: (i) the estimated processing time $\tilde{p}_j$, as reported by the user; (ii) the actual processing time p_j (known only upon completion of its execution); (iii) the resource requirements q_j, measured in terms of the number of processors; (iv) the submit time r_j, also known as the release date.

A typical approach in scheduling is to make several simplifications regarding the characteristics of jobs. Instead, a job is often viewed as an independent entity, or "black box", that requires a fixed amount of resources q for an estimated processing time of p units.

In the online scheduling problem, multiple objective functions can be evaluated. For this study, the average bounded slowdown ($\mathrm{AVGbsld}$) is chosen as the performance metric for scheduling. This objective function in online parallel job scheduling is difficult to handle theoretically, often requiring the use of heuristics in practice. Furthermore, the $\mathrm{AVGbsld}$ expresses an expected proportional relationship between the waiting time of jobs and their processing time [Feitelson 2001].

The bounded slowdown (bsld) value for a job j is computed as follows:

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

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

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

3. Investigation Method

3.1. Simulation Design

We used the simulation approach proposed by Carastan-Santos and de Camargo [Carastan-Santos and de Camargo 2017]. The primary goal of this approach is to capture scheduling knowledge through simulations of job sets executed under different conditions. The approach is delineated hereafter.

Let S and Q denote two sets of jobs. The proposed approach involves simulating the scheduling of the jobs in Q on a homogeneous cluster of m resources (processors) representing an HPC platform while the jobs in S are being processed. At the onset of the simulation, the jobs from set S are executed in the order of their arrival to function as a warm-up workload. Once all jobs from S have arrived, the jobs in set Q, which are utilized to extract scheduling performance information, begin to arrive.

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 given pair (S, Q), permutations Q^* of the set Q are randomly sampled, referred to as a *trial* of Q, and the scheduling of jobs is simulated in the order in which they occur in (S, Q^*). The set $\mathcal{P}$ contains all the sampled permutations, and upon the completion of scheduling simulation for all the trials, a score is computed and assigned to each job $j \in Q$:

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

The $score$ represents the impact of scheduling a job $j \in Q$ as the first job (represented by j_0 in Equation 3), in terms of the average bounded slowdown (Equation 2) of all jobs in Q. Jobs with low scores have a positive impact on reducing the overall average slowdown when they are executed first.

We replicated the previously mentioned simulation strategy with multiple samples of job set pairs (S, Q), and the collected data were used to generate a distribution of scores, denoted as $score(p, q, r)$. 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.

3.2. 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 utilized functions that relied on features based on terms such as square roots and logarithms, which can be challenging to interpret in the context of scheduling. In contrast, our proposed approach in this work involves the exploration of polynomial-based functions.

We defined a function family $\mathcal{F}$, with parameterized functions of the form

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

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

We built four functions, illustrated in Table 1. The function Lin is a linear combination of the jobs' characteristics p, q and r. The others Qdr, Cub and Qua are functions

Table 1. Vector components of the four parameterized 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)		✓	✓	✓
$(p^3, q^3, r^3, p^2q, pq^2)$			✓	✓
$(p^4, q^4, r^4, p^3q, p^2q^2, pq^3)$				✓

that progressively increase the degree of the basis functions, and with multiplicative factors related to pq, which is often referred in the literature [Carastan-Santos et al. 2019] as the area of the jobs.

Our proposed method leverages weighted multiple linear regression to minimize the weighted sum of squared loss function, as shown in equation 5. The weighting term, represented by $(p_j q_j)$, places emphasis on accurate estimation of $score$ values for jobs with large p and q values. These jobs can have a significant impact on overall scheduling performance by blocking the execution of smaller jobs.

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

Once we have obtained the coefficients $\hat{\boldsymbol{\theta}}$ through multiple linear regression for all functions $f(\boldsymbol{\theta}, \mathbf{x}) \in \mathcal{F}$, we can assess the quality of the fitting using the Mean Absolute Error (MAE) function, as shown in equation 6.

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

4. Results and discussion

4.1. Gathering scheduling knowledge through simulation

In our simulations, we considered an HPC platform comprising of 256 homogeneous processors. To generate the distribution $score(p, q, r)$, we employed sets of 16 and 32 jobs for the sets S and Q, respectively. Job characteristics for both sets were derived using the Lublin and Feitelson [Lublin and Feitelson 2003] workload model, which is based on a generalized workload derived from real HPC workload logs. This model can represent geometry of jobs (p_j and q_j), as well as their release date (r_j), including peak periods. All simulations were conducted using SimGrid [Casanova et al. 2014].

Additionally, enumerating and simulating the execution of all possible permutations of a job set Q with a size of 32 is computationally infeasible. Therefore, it was necessary to determine an appropriate number of permutations that can generate accurate trial score distributions. For that, we selected a pair (S, Q), calculated the trial distributions with incrementally increasing numbers of trials, and measured the standard deviation of the estimated scores ten times over. We opted to use 256,000 trials, as it offered a good balance between quick simulations and a low standard deviation. The simulations produced a distribution of scores containing 14,081 job characteristics and their respective scores.

4.2. Scheduling effectiveness and polynomial size relationship

We examine how the effectiveness of regression-based scheduling heuristics is influenced by polynomial size. Table 2 presents the entire list of coefficients $\boldsymbol{\theta}$ obtained from the regression analysis. We observed instabilities in the features' coefficients between Lin, Qdr, Cub, and Qua across the four functions. Such instability is manifested by a change in the sign of the coefficients within a given feature, indicating a potential impact on the resulting schedule. For instance, the Lin function gives higher priority to jobs with smaller q values due to its positive coefficient for q, while the Qdr function assigns higher priority to jobs with larger q values owing to its negative coefficient for q.

Table 2. 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

To evaluate the scheduling performance of our functions, we performed online job scheduling simulations. For these simulations, a task queue was generated with characteristics obtained from the Lublin and Feitelson workload model, and a 256-core HPC platform was considered. These settings are the same as those used in the scheduling knowledge collection simulations.

The online scheduling algorithm functions as follows: jobs are received in a central waiting queue, and the scheduler, which operates using a scheduling heuristic, reschedules these jobs in the queue at two distinct events: (i) when a new job is added to the queue, and (ii) when a set of cores becomes available due to the completion of a job.

We define a dynamic scheduling experiment as the simulation of the online scheduling algorithm for different job sequences extracted from a single workload trace and using the same scheduling policy. Each sequence consists of all job submissions over a fifteen-day period, with no overlap between them.

In Figure 1, we present the average bounded slowdown for fifty dynamic scheduling experiments using the proposed heuristics. Among the four functions that were an-

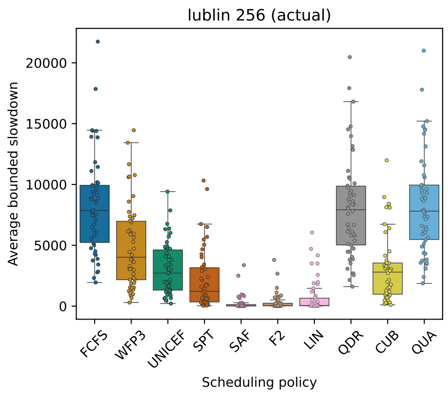

Figure 1. Scheduling performance comparison with jobs generated from Lublin & Feitelson workload model and using the actual processing time.

alyzed, the Lin function showed one of the best results, comparable to the SAF and F2 heuristics, which has proven to be reliably efficient [Carastan-Santos et al. 2019, Carastan-Santos and de Camargo 2017]. Conversely, the most complex functions, Qdr, Cub, and Qua, did not perform as well as Lin, producing one the worst results. Although the MAE values remained close for all functions ($\mathrm{Lin} = 4.48 \times 10^{-3}$, $\mathrm{Qdr} = 4.66 \times 10^{-3}$, $\mathrm{Cub} = 10.5 \times 10^{-3}$, and $\mathrm{Qua} = 6.42 \times 10^{-3}$), the performance of the most complex functions proved to be as inefficient as FCFS. These results emphasize the negative impact of the coefficients stability.

4.3. The negative impact of multicollinearity on scheduling heuristics

The instability of the coefficients suggests that the functions suffer from the phenomenon of multicollinearity [Alin 2010], which occurs when the features are highly correlated with each other. To test this hypothesis, we computed the Variance Inflation Factor [García et al. 2022] (VIF) for all features in the Lin, Qdr, Cub, and Qua functions (see Table 2). Moderate correlation (VIF > 5) was observed for Qdr function with q, r, and q^2. The higher degree functions (Cub, and Qua) showed extremely high correlation (VIF $\gg 10$), with VIFs exceeding 10,000 for q^3 in the Qua function. We conjecture that introducing more derivative features will increase VIFs and exacerbate multicollinearity.

Table 3. Scheduling policies used for comparison. Detailed information regarding WFP3, UNICEF, and F2 policies can be found in [Tang et al. 2009] and [Carastan-Santos and de Camargo 2017].

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

4.4. Resilience evaluation of regression-derived scheduling heuristics

The following experiments were designed to evaluate the generalizability of regression-based scheduling heuristics to different workloads. To achieve this, we compared them with the policies listed in table 3, using workload traces obtained from large-scale HPC platforms. The Parallel Workloads Archive [Feitelson et al. 2014] and the ALAS Repository [Amvrosiadis et al. 2018] were the sources of the traces, which were selected to represent 19 years of development in HPC platforms and workloads, ranging from an old IBM SP2 with a few hundred CPUs to a modern supercomputer with hundreds of thousands of CPUs.

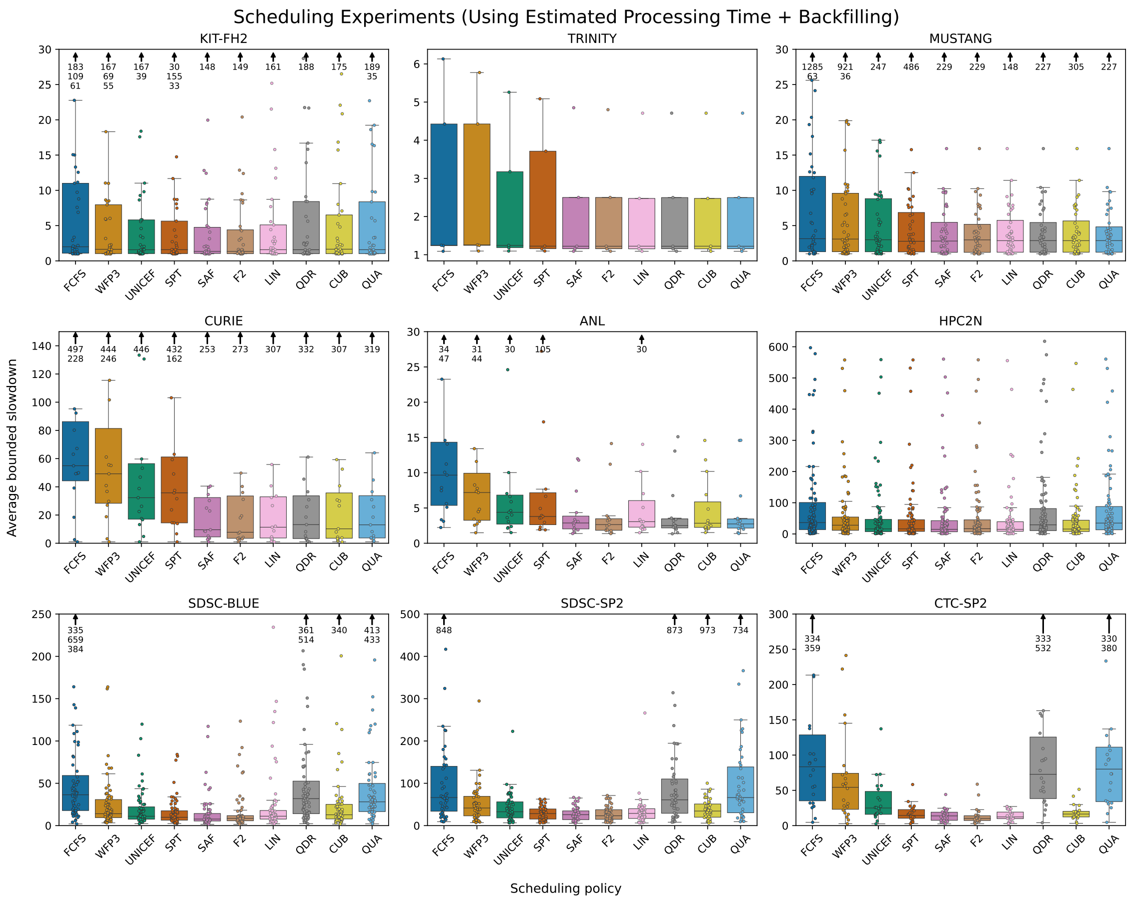

Figure 2. Computed average bounded slowdown for different scheduling policies. Experiments based on user-estimated processing times, with aggressive backfilling algorithm.

Two different online scheduling experiments were conducted: (i) scheduling based on job processing time estimates, and (ii) scheduling by combining processing time estimates with an aggressive backfilling technique [Mu'alem and Feitelson 2001] – which helps reduce platform idle time. The second scenario is particularly relevant to a real-world HPC platform. However, due to the limited scope of this paper, we omit the results of the first scenario, as they were found to be similar to those observed in the second scenario which is more realistic.

The results shown in Figure 2 indicate that backfilling has a significant impact on the performance of poorly-performing heuristics such as FCFS. However, it is insufficient to surpass the queue sorting methods used by SAF, F2, and Lin, which consistently demonstrate superior performance. On the other hand, the Lin function had lower performance compared to F2 and SAF policies, but it often resulted in lower average bounded slowdowns and smaller differences between extreme quartiles for most workloads. This shows that despite its simplicity, it has good scheduling performance.

The results demonstrate that regression-based scheduling heuristics, specifically F2 and Lin, can maintain stable and efficient performance across a wide range of HPC platforms and workloads. For instance, the Mustang trace spans a 5-year workload evolution (from 2011 to 2016), and F2 and Lin showed good performance on all Mustang workload samples. Furthermore, both F2 and Lin were designed once and did not require any adjustments over time to maintain their efficient scheduling performance.

5. Conclusion

In conclusion, this scientific initiation work explored the relationship between managing resources on high-performance computing platforms and using regression-derived scheduling heuristics to optimize performance. The use of polynomial functions to generate scheduling heuristics was proposed, and the simplest polynomial was found to be particularly efficient.

In addition, our study demonstrated that the proposed regression-derived heuristics can provide stable and efficient scheduling performance across a wide range of HPC platforms and workloads without the need for adjustments over time. The findings of this research have been published in two peer-reviewed national and international workshops. Moving forward, we plan to further extend this work by considering job power demand as a feature, with the objective of achieving optimized scheduling efficiency with the lowest possible overall power consumption of the platform.

References

Alin, A. (2010). Multicollinearity. *Wiley Interdisciplinary Reviews: Computational Statistics*, 2(3):370–374.

Amvrosiadis, G., Kuchnik, M., Park, J. W., Cranor, C., Ganger, G. R., Moore, E., and DeBardeleben, N. (2018). The atlas cluster trace repository. *Usenix Mag*, 43(4).

Brucker, P. (2007). *Scheduling Algorithms*. Springer, hardcover edition.

Carastan-Santos, D., Camargo, R. Y. D., Trystram, D., and Zrigui, S. (2019). One can only gain by replacing EASY backfilling: A simple scheduling policies case study. In *2019 19th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGRID)*. IEEE.

Carastan-Santos, D. and de Camargo, R. Y. (2017). Obtaining dynamic scheduling policies with simulation and machine learning. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*. ACM.

Casanova, H., Giersch, A., Legrand, A., Quinson, M., and Suter, F. (2014). Versatile, scalable, and accurate simulation of distributed applications and platforms. *Journal of Parallel and Distributed Computing*, 74(10):2899–2917.

Fan, Y., Lan, Z., Childers, T., Rich, P., Allcock, W., and Papka, M. E. (2021). Deep reinforcement agent for scheduling in hpc. In *2021 IEEE International Parallel and Distributed Processing Symposium (IPDPS)*, pages 807–816.

Feitelson, D. G. (2001). Metrics for parallel job scheduling and their convergence. In *Job Scheduling Strategies for Parallel Processing*, pages 188–205. Springer Berlin Heidelberg.

Feitelson, D. G., Rudolph, L., Schwiegelshohn, U., Sevcik, K. C., and Wong, P. (1997). Theory and practice in parallel job scheduling. In *Job Scheduling Strategies for Parallel Processing: IPPS'97 Processing Workshop Geneva, Switzerland, April 5, 1997 Proceedings 3*, pages 1–34. Springer.

Feitelson, D. G., Tsafrir, D., and Krakov, D. (2014). Experience with using the parallel workloads archive. *Journal of Parallel and Distributed Computing*, 74(10):2967–2982.

García, C. G., Gómez, R. S., and Pérez, J. G. (2022). A review of ridge parameter selection: minimization of the mean squared error vs. mitigation of multicollinearity. *Communications in Statistics - Simulation and Computation*, pages 1–13.

Legrand, A., Trystram, D., and Zrigui, S. (2019). Adapting batch scheduling to workload characteristics: What can we expect from online learning? In *2019 IEEE International Parallel and Distributed Processing Symposium (IPDPS)*. IEEE.

Li, J., Zhang, X., Han, L., Ji, Z., Dong, X., and Hu, C. (2021). Okcm: improving parallel task scheduling in high-performance computing systems using online learning. *The Journal of Supercomputing*, 77(6):5960–5983.

Lublin, U. and Feitelson, D. G. (2003). The workload on parallel supercomputers: modeling the characteristics of rigid jobs. *Journal of Parallel and Distributed Computing*, 63(11):1105–1122.

Mu'alem, A. and Feitelson, D. (2001). 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.

Rosa, L., Carastan-Santos, D., and Goldman, A. (2023). An experimental analysis of regression-obtained hpc scheduling heuristics. In *Job Scheduling Strategies for Parallel Processing*. Springer-Verlag. To be published.

Rosa, L. and Goldman, A. (2022). In search of efficient scheduling heuristics from simulations and machine learning. In *Anais Estendidos do XXIII Simpósio em Sistemas Computacionais de Alto Desempenho*, pages 17–24, Porto Alegre, RS, Brasil. SBC.

Shalf, J. (2020). The future of computing beyond moore's law. *Philosophical Transactions of the Royal Society A: Mathematical, Physical and Engineering Sciences*, 378(2166):20190061.

Tang, W., Lan, Z., Desai, N., and Buettner, D. (2009). Fault-aware, utility-based job scheduling on blue, gene/p systems. In *2009 IEEE International Conference on Cluster Computing and Workshops*. IEEE.

Zhang, D., Dai, D., He, Y., Bao, F. S., and Xie, B. (2020). Rlscheduler: An automated hpc batch job scheduler using reinforcement learning. In *SC20: International Conference for High Performance Computing, Networking, Storage and Analysis*, pages 1–15.

Zrigui, S., de Camargo, R. Y., Legrand, A., and Trystram, D. (2022). Improving the performance of batch schedulers using online job runtime classification. *Journal of Parallel and Distributed Computing*, 164:83–95.