SAND2021-9246C

Backfilling HPC Jobs with a Multimodal-Aware Predictor

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

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

***Abstract*—Job scheduling aims to minimize the turnaround time on the submitted jobs while catering to the resource constraints of High Performance Computing (HPC) systems. The challenge with scheduling is that it must honor job requirements and priorities while actual job run times are unknown. Although approaches have been proposed that use classification techniques or machine learning to predict job run times for scheduling purposes, these approaches do not provide a technique for reducing underprediction, which has a negative impact on scheduling quality. A common cause of underprediction is that the distribution of the duration for a job class is multimodal, causing the average job duration to fall below the expected duration of longer jobs. In this work, we propose the Top Percent predictor, which uses a hierarchical classification scheme to provide better accuracy for job run time predictions than the user-requested time. Our predictor addresses multimodal job distributions by making a prediction that is higher than a specified percentage of the observed job run times. We integrate the Top Percent predictor into scheduling algorithms and evaluate the performance using schedule quality metrics found in literature. To accommodate the user policies of HPC systems, we propose priority metrics that account for job flow time, job resource requirements, and job priority. The experiments demonstrate that the Top Percent predictor outperforms the related approaches when evaluated using our proposed priority metrics.**

***Index Terms*—High Performance Computing, Running Time Prediction, Schedule Quality**

I. Introduction

High Performance Computing (HPC) systems deliver computing power to applications through job schedulers that manage compute nodes. The job scheduler typically accepts a user-requested time which serves as the upper bound for each job's running time. The scheduling performance is evaluated using a metric referred to as *scheduling quality*, where better performance is generally characterized by a lower average wait time for the scheduled jobs. *Backfilling* is a common approach to fill in scheduling gaps with lower priority jobs based on user-requested time, improving scheduling quality though increased node utilization. Since jobs that exceed the user-requested time are terminated, it is in the user's best interest to significantly overpredict the job running time. The dual-purpose use of user-requested time both as an upper bound on run time and as a backfill predictor has a negative impact on resource utilization; constantly overpredicting job running times negatively impacts schedule quality since this leads to fewer jobs that can be backfilled. As a result, there is a demand for prediction techniques that can accurately predict the running time of a job for more aggressive backfilling without burdening users for extra information.

Previous approaches have identified that jobs with common attributes are likely to exhibit similar run times in comparison to jobs with no common attributes [1]–[4]. Other approaches use machine learning to predict the job run time [5], [6]. Although these job duration prediction techniques are effective at approximating the job run time, they do not address underestimates [7], a crucial factor given that the performance penalty of underprediction often produces a schedule quality that is worse than the simple approach of using the overestimated user-requested time from the start.

A culprit of frequent underprediction is that the distribution of the duration for each job class is often multimodal – the job duration is either very short or long. Since a significant number of jobs have a short duration, the average job duration is shifted below the common case duration for the longer jobs. This complicates the straightforward solution of adding the standard deviation multiplied by a factor to achieve the desired confidence level. In this work, we propose a job duration predictor that uses a hierarchical classification scheme while overcoming the multimodal job distribution by basing the prediction on the outliers with the longest duration. Our predictor handles the possibility of the observed duration of the jobs changing midway by assigning weights to the observed results to adapt to the change in job duration.

We evaluate our proposed predictor by running workloads from the Parallel Workload Archive [8] on a scheduling simulator [5] and using our predictor to predict the job duration which is provided to the scheduling algorithm. We measure the schedule quality of the job schedule produced

Sandia National Laboratories is a multi-mission laboratory managed and operated by National Technology and Engineering Solutions of Sandia, LLC., a wholly owned subsidiary of Honeywell International, Inc., for the U.S. Department of Energy's National Nuclear Security Administration under contract DE-NA-0003525. SAND #: SAND2020-8522 C

by the scheduling algorithm to assess the effectiveness of our predictor. Schedule quality metrics that are used in literature include *average bounded slowdown* (AVEbsld) [9], [10] and *average flow time* (AF) [11]. AVEbsld and AF trend favorably for a schedule that prioritizes shorter jobs since more short jobs can be executed per time unit to minimize wait time, encouraging the development of algorithms solely targeting shortest job first scheduling. These existing metrics are useful for perceived system responsiveness for users, but they ignore constraints such as job resource requirements and job priority that are important to HPC administrators and ultimately improve overall scheduling quality. To address these concerns, we propose new schedule quality metrics, referred to as *priority metrics*, that account for flow time, resource requirements, and job priority. We demonstrate that predicting job duration in the scheduling algorithm using our proposed predictor yields better schedule quality based on our metrics than the baseline approach of scheduling jobs using user-requested times.

II. Proposed Predictor Design

We propose the Top Percent predictor, a novel prediction approach which limits underprediction while accounting for multimodal job distribution. First, each job is classified in a similar fashion as Smith et al. [1] using data known before the job starts, including the executable name, the name of the user that submitted the job, the user-requested run time, and the number of required processors. This allows grouping of related jobs, which typically have similar run times. The Top Percent predictor tracks a threshold that separates the longest run time outliers (the longest 3% of jobs, for example) from the rest of the jobs, then predicting along that threshold to ensure with high confidence that our prediction will not underpredict. This works well when each job class runs for consistent amounts of time with few outliers, but the average run time often fluctuates for each class over time. To accommodate this, we assign weights to each observed run time, using exponential decay to ensure more recent run times hold more weight than older run times. The decay rate can be tuned between responsiveness to, and stability against, job duration changes. In the presence of a multimodal distribution, weights allow the shorter run times to drag down our prediction far below the longer run times. To limit these effects, weights are also proportional to job length, with longer jobs having more weight. This design allows for more aggressive backfilling than traditional user-requested time while still minimizing underprediction to avoid the associated performance hit. The behavior of this predictor is visualized in Fig. 1. In this example, the job group has a trimodal distribution. The predicted run time (grey line) maintains a cautious overprediction even in the presence of many short jobs, yet reduces the predicted run time when long jobs are no longer being queued.

III. Proposed Priority Metrics

The schedule quality metrics in literature, including AVEbsld [9], [10] and AF [11], do not account for constraints such as job resource requirements and job priority. We address this

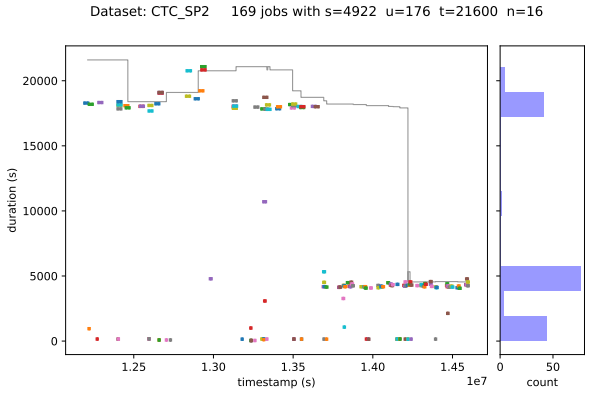

Fig. 1. **Predictor behavior for an example job group with a multimodal distribution.** Job start time is on the x-axis and run time is on the y-axis. Boxes are individual jobs and the grey line is the predicted run time. The histogram on the right shows the frequency of jobs with a particular run time.

issue by proposing *priority metrics* that are inclusive of flow time, job resource requirements, and job priority. Our proposed priority metrics are centered around computing a Weighted Average Flow (WAF). We select the weights based on the resource constraints of HPC systems. The *area* a_j of a job, also referred to as the *squashed area* [12], is the product of the running time T_r of a job and the number of resources n. Since the area of a job impacts the availability of resources, we select the area as a weight for our proposed WAF. We also select job priority p as a weight for our proposed WAF.

In HPC systems, it is a common policy to assign a higher priority to jobs that waited longer. In this case, the behavior of a job's priority over time should be considered instead of treating the job's priority as a static value. We assume that the job priority is modeled as a function of time $p_j(t)$, referred to as a *priority function*. The priority function may also account for a user priority or a project priority. It is also possible for the number of resources required by a job to change over time, where the number of resources is modeled by a function of time $\sigma_j(t)$, referred to as a *resource function*. The job "area" can now be considered a volume, where time is the x-axis, the number of required resources is the y-axis, and the priority is the z-axis. The job *volume* is computed by integrating the product of the resource function and the priority function between the limits of the job-start-to-run-at-time R_j and the job completion time C_j, yielding $\int_{R_j}^{C_j} \sigma_j(t) \cdot p_j(t)\ dt$ as a weight for flow time F_j in the WAF expressed in Equation 1.

$$WAF = \frac{\sum_{j=1}^{n} \left(\int_{R_j}^{C_j} \sigma_j(t) \cdot p_j(t)\ dt \right) \cdot F_j}{\sum_{j=1}^{n} \int_{R_j}^{C_j} \sigma_j(t) \cdot p_j(t)\ dt} \tag{1}$$

A. Applied WAF

We apply Equation 1 to measure schedule quality when evaluating the Top Percent predictor. We choose the number of required nodes to serve as our resource requirements. Since the number of resources (nodes) required by a job is held constant in the jobs documented in the workload logs from the Parallel

Workload Archive [8], we set $\sigma_j(t) = n_j$ in Equation 1. We assume that $p_j(t)$ is a function of time relative to the submission time S_j of a job, so we set $p_j(t) = (t - S_j)^\alpha$ in Equation 1 yielding a weight of $n_j \cdot \int_{R_j}^{C_j} (t - S_j)^\alpha \, dt$, where α is an amplification factor. The wait time T_{wj} is R_j - S_j. We set R_j to $S_j + T_{wj}$ and C_j to $S_j + F_j$, and perform the integration to obtain Equation 2, which we refer to as *Area * Priority Weighted Average Flow* (AP$^\alpha$WAF).

$$AP^\alpha WAF = \frac{\sum_{j=1}^{n} n_j \cdot \left(F_j^{\alpha+1} - {T_w}_j^{\alpha+1}\right) \cdot F_j}{\sum_{j=1}^{n} n_j \cdot \left(F_j^{\alpha+1} - {T_w}_j^{\alpha+1}\right)} \quad (2)$$

B. Prediction Metrics

The coefficient of determination (R^2) is a standard statistic used to measure how well a prediction model matches the actual values. This makes R^2 a suitable metric to measure prediction quality independently from scheduling quality. R^2 is computed by Equation 3, where y_i is an observed run time, $\bar{y}$ is the mean of the observed run times which is the baseline, and f_i is the predicted run time. Using this equation, $R^2 = 1$ if prediction is perfect; 0 if prediction is equivalent to a baseline; and a negative value if the prediction is worse than the baseline.

$$R^2 = 1 - \frac{\sum_{i=1}(y_i - f_i)^2}{\sum_{i=1}(y_i - \bar{y})^2} \quad (3)$$

IV. Experimental Results

Experiments ran on an expanded version of Predictsim [5], a parallel job scheduler simulator. This simulator is simplified to consider the number of nodes as the only limited resource and to assume a job's length is fixed, unaffected by external factors such as I/O. Predictsim simulates the scheduling of jobs provided by standard workload format (swf) logs. The Cornell Theory Center SP2 '96 (CTC-SP2), Swedish Royal Institute of Technology SP2 '96 (KTH-SP2), and San Diego Supercomputer Center SP2 '98 (SDSC-SP2) workloads were sourced from the Parallel Workload Archive [8] because they have been compared in related work [13]. We added a workload from the NCSA Blue Waters supercomputer (BW), which was sourced by converting publicly available TORQUE logs from December 2019 to swf.

We evaluate our proposed Top Percent predictor at 2-4% (Top02, Top03, Top04). We compare our predictor against our Complete predictor (Complete), which does job group classification like Smith et al. [1], then makes a prediction based on average historic run times associated with the group. We compare against Tsafrir's predictor [14] (Tsafrir) that predicts using the average run times of the last two jobs and CVH [5] (CVH) that predicts using machine learning techniques. The user-requested time predictor (Reqtime) uses the user-requested time, provided at job submit time, to predict actual run time; this serves as a baseline lower-bound on prediction quality. The clairvoyant predictor (Clairvoyant) takes advantage of the simulated environment to use the actual run time as the predicted run time; this is impossible in practice but establishes an upper bound on maximum predictor improvement. We tested the behavior of several backfill schedulers, including Conservative backfill (Cons) [13], EASY backfill (EASY) [13], and Largest Area First backfill (LAF). Results were relatively close, so only EASY, the most widely used backfilling approach, is reported.

A. Metrics

Each backfill algorithm was evaluated using conventional scheduling quality metrics: average bounded slowdown (AVEbsld) [9], [10] and average flow time (AF) [11]. We also compare using our newly proposed priority metric, Area * Priority Weighted Average Flow (AP$^\alpha$WAF). The α of each metric is tested at 0 and 1, where 0 only considers flow time and job packing efficiency while 1 additionally considers priorities. Finally, we report R^2 to measure prediction quality and evaluate its relationship with scheduling quality.

B. Results

Our full results are listed in Table I. As Reqtime is intended to be a minimum baseline for improvement, all reported metrics except R^2 are normalized to Reqtime by dividing each by the Reqtime metric. Any value over 1 is worse than the baseline of using user-requested run time while under 1 is an improvement. Clairvoyant is included as a baseline reference of maximum possible improvement.

Complete offers the best prediction quality (R^2) in all cases and provides competitive and in some cases superior AVEbsld and AF. Its poor performance using our new metrics is a result of treating overprediction and underprediction as equally bad, even though underprediction more harshly affects scheduling performance, as reflected by our new metrics.

CVH [5] typically offers the best AVEbsld and AF, but the worst results for our new metrics. This is likely because CVH's machine learning approach explicitly targets these metrics while ignoring the properties we value in AP$^\alpha$WAF. R^2 is poor because accurate prediction was not a goal of the CVH design; it is a good illustration of how prediction quality, perhaps unintuitively, is not inherently well-correlated with scheduling quality. Tsafrir [14] behaves similarly to CVH, targeting the traditional AVEbsld and AF metric, but Tsafrir always underperforms when compared against the other predictors.

Our proposed Top Percent predictor offers tolerable AVEbsld and AF. It is the only design in our tests that can consistently outperform Reqtime for our AP$^\alpha$WAF metrics and consistently offers the best priority-sensitive results. We believe Reqtime does well because it is the only practical approach that cannot underpredict; if the job tries to run longer than Reqtime, then the scheduler terminates the job for exceeding the length provisioned by the scheduler. This advantage means that, although our improvements over Reqtime appear minor, any predictor that can outperform Reqtime by any margin is a meaningful improvement to the state-of-the-art. None of the other related works have managed to accomplish this in

BW	AVEbsld	AF	AP^0WAF	AP^1WAF	R^2	**KTH-SP2**	AVEbsld	AF	AP^0WAF	AP^1WAF	R^2
Reqtime	1.000	1.000	1.000	1.000	-0.234	Reqtime	1.000	1.000	1.000	1.000	-0.183
Clairvoyant	0.646	0.917	0.940	0.932	1.000	Clairvoyant	0.772	0.967	0.980	0.976	1.000
Complete	**0.192**	0.706	1.078	1.317	**0.106**	Complete	0.796	0.960	1.041	1.096	**0.641**
CVH	0.243	**0.612**	**0.985**	1.208	-0.157	CVH	**0.745**	**0.915**	1.052	1.399	0.008
Tsafrir	0.277	0.626	1.310	2.113	-0.157	Tsafrir	0.771	0.944	1.016	1.084	0.008
Top02	1.102	1.043	0.998	**0.996**	-4.791	Top02	0.996	0.996	**0.994**	0.987	0.601
Top03	2.121	1.260	1.032	1.039	-4.825	Top03	1.001	0.997	0.994	**0.987**	0.603
Top04	1.755	1.144	1.024	1.032	-4.803	Top04	1.000	0.997	0.994	0.987	0.605
CTC-SP2	AVEbsld	AF	AP^0WAF	AP^1WAF	R^2	**SDSC-SP2**	AVEbsld	AF	AP^0WAF	AP^1WAF	R^2
Reqtime	1.000	1.000	1.000	1.000	-0.380	Reqtime	1.000	1.000	1.000	1.000	-0.232
Clairvoyant	0.761	0.986	0.870	0.828	1.000	Clairvoyant	0.802	0.884	0.875	0.873	1.000
Complete	0.964	1.002	1.590	3.018	**0.508**	Complete	**0.858**	**0.930**	1.026	1.122	**0.637**
CVH	**0.634**	**0.794**	1.149	4.165	-0.219	CVH	0.930	0.960	1.067	2.236	-0.197
Tsafrir	0.789	0.926	1.593	4.728	-0.219	Tsafrir	0.901	0.934	1.036	1.179	-0.197
Top02	1.000	0.989	1.002	1.007	-0.416	Top02	1.024	1.004	0.999	1.008	-0.335
Top03	1.041	1.001	**0.998**	**0.999**	-0.385	Top03	1.029	1.002	0.992	0.998	-0.330
Top04	1.041	1.016	1.008	1.005	-0.364	Top04	1.042	1.006	**0.990**	**0.995**	-0.325

TABLE I

EASY BACKFILL METRICS. REQTIME AND CLAIRVOYANT ARE BASELINE PREDICTORS. ALL BUT R^2 ARE NORMALIZED TO THE REQTIME BASELINE, WITH LOWER METRICS BEING BETTER. THE BEST NON-BASELINE RESULT FOR EACH METRIC AND TEST IS HIGHLIGHTED IN **BOLD**.

our testing. We attribute our results to a design that favors overprediction. It will never predict more time than Reqtime, but with historic knowledge, it can reduce the predicted time under Reqtime for more aggressive backfilling with minimum underprediction and support for multimodal job distributions.

V. RELATED WORK

One of the primary approaches for predicting job run times is the application of statistical methods on historical data [1], [4], [13]–[16]. Gibbons [4] predicts job running times from historical data using a desired confidence level. Smith et al. [1] predict job running time by categorizing the jobs according to the historical run time information. Tsafrir et al. [14] propose a job running time predictor that simply averages the last two jobs by the same user. Sfiligoi et al. [17] establish a correlation between users and their job runtimes based on the historical data from the Compact Muon Solenoid (CMS) experiment at the Large Hadron Collider. Another effective approach for predicting job running times is machine learning [5], [7], [18]–[20]. Gaussier et al. [5] use machine learning to predict job running times that will produce an improved schedule quality. Fan et al. [7] make job run time predictions using the Tobit model which is a censored regression model that censors the latent variables, enabling an approach that reduces the problem of underestimation in job run time prediction.

Our proposed Top Percent predictor is most similar to the approach of Smith et al. [1] in that it classifies jobs according to the provided job data. The characteristic that distinguishes our predictor from the related statistical method approaches is that it alleviates underprediction by making a prediction based on the jobs that fall in the top percentage threshold for job duration. The reduction in underprediction avoids the schedule quality hit incurred due to severe disruption of scheduling resulting from jobs that do not finish in their allocated time.

The schedule quality metrics commonly used in practice are centered around job delay [9], [11], [21]. Horn [11] uses the notion of *flow time*, ie. the total delay of the jobs, as the criterion to minimize when assigning jobs to parallel machines. Feitelson et al. [9] propose *bounded slowdown* that is computed by dividing the flow time (wait time + run time) by the maximum between the run time and a threshold for the wait time. Schedule quality metrics have also been proposed that account for job resource requirements [12], [22]–[24]. Ernemann et al. [12] introduce the *squash area* as the product of the number of requested resources and the run time for a job. The authors compute the Average Weighted Wait Time (AWWT) as the wait time weighted by the squash area. Unlike the other schedule quality metrics, our proposed priority metrics are the first to account for job priority when computing the weighted average flow time.

VI. CONCLUSION

We proposed Top Percent, a new predictor that adapts to multimodal job distributions and minimizes underpredictions. We also proposed AP$^\alpha$WAF, a new scheduling quality metric that, unlike existing works, evaluates flow time alongside resource utilization and job priorities. Our new predictor offers some of the best scheduling quality results for our metric, while remaining competitive with existing metrics, via more aggressive backfilling with minimum underprediction and support for multimodal job distributions. Our new metric addresses important gaps in scheduling quality valued by HPC administrators. In future work, we plan to improve prediction quality by considering job inputs and parameters in addition to what is already provided by swf logs. We also plan to extend this work to predict factors such as I/O and other unknown levels of resource utilization for improved scheduling.

VII. ACKNOWLEDGMENTS

The authors thank Benjamin Schwaller and Omar Aaziz for their valuable discussions. The works at the University of Central Florida were supported by contracts with Sandia National Laboratories.

REFERENCES

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

[2] A. B. Downey, "Predicting queue times on space-sharing parallel computers," in *Proceedings 11th International Parallel Processing Symposium*. IEEE, 1997, pp. 209–218.

[3] D. G. Feitelson and B. Nitzberg, "Job characteristics of a production parallel scientific workload on the nasa ames ipsc/860," in *workshop on job scheduling strategies for parallel processing*. Springer, 1995, pp. 337–360.

[4] R. Gibbons, "A historical application profiler for use by parallel schedulers," in *Workshop on Job Scheduling Strategies for Parallel Processing*. Springer, 1997, pp. 58–77.

[5] E. Gaussier, D. Glesser, V. Reis, and D. Trystram, "Improving backfilling by using machine learning to predict running times," in *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, ser. SC '15. New York, NY, USA: Association for Computing Machinery, 2015. [Online]. Available: https://doi.org/10.1145/2807591.2807646

[6] J.-W. Park and E. Kim, "Runtime prediction of parallel applications with workload-aware clustering," *The Journal of Supercomputing*, vol. 73, no. 11, pp. 4635–4651, 2017.

[7] Y. Fan, P. Rich, W. E. Allcock, M. E. Papka, and Z. Lan, "Trade-off between prediction accuracy and underestimation rate in job runtime estimates," in *2017 IEEE International Conference on Cluster Computing (CLUSTER)*. IEEE, 2017, pp. 530–540.

[8] D. G. Feitelson, D. Tsafrir, and D. Krakov, "Experience with using the parallel workloads archive," *Journal of Parallel and Distributed Computing*, vol. 74, no. 10, pp. 2967–2982, 2014.

[9] D. G. Feitelson, L. Rudolph, U. Schwiegelshohn, K. C. Sevcik, and P. Wong, "Theory and practice in parallel job scheduling," in *Workshop on Job Scheduling Strategies for Parallel Processing*. Springer, 1997, pp. 1–34.

[10] D. G. Feitelson, "Metrics for parallel job scheduling and their convergence," in *Workshop on Job Scheduling Strategies for Parallel Processing*. Springer, 2001, pp. 188–205.

[11] W. Horn, "Minimizing average flow time with parallel machines," *Operations Research*, vol. 21, no. 3, pp. 846–847, 1973.

[12] C. Ernemann, V. Hamscher, and R. Yahyapour, "Benefits of global grid computing for job scheduling," in *Fifth IEEE/ACM International Workshop on Grid Computing*. IEEE, 2004, pp. 374–379.

[13] A. Mu'alem and D. Feitelson, "Utilization, predictability, workloads, and user runtime estimates in scheduling the ibm sp2 with backfilling," *IEEE Transactions on Parallel and Distributed Systems*, vol. 12, no. 6, pp. 529–543, 2001.

[14] D. Tsafrir, Y. Etsion, and D. G. Feitelson, "Backfilling using system-generated predictions rather than user runtime estimates," *IEEE Transactions on Parallel and Distributed Systems*, vol. 18, no. 6, pp. 789–803, 2007.

[15] W. Tang, N. Desai, D. Buettner, and Z. Lan, "Analyzing and adjusting user runtime estimates to improve job scheduling on the blue gene/p," in *2010 IEEE International Symposium on Parallel & Distributed Processing (IPDPS)*. IEEE, 2010, pp. 1–11.

[16] S. Zrigui, R. de Camargo, A. Legrand, and D. Trystram, "Improving the performance of batch schedulers using online job runtime classification," *IEEE Transactions on Parallel and Distributed Systems*, 2020.

[17] I. Sfiligoi, "Estimating job runtime for cms analysis jobs," in *Journal of Physics: Conference Series*, vol. 513, no. 3. IOP Publishing, 2014, p. 032087.

[18] M. Tanash, B. Dunn, D. Andresen, W. Hsu, H. Yang, and A. Okanlawon, "Improving hpc system performance by predicting job resources via supervised machine learning," in *Proceedings of the Practice and Experience in Advanced Research Computing on Rise of the Machines (learning)*, 2019, pp. 1–8.

[19] Q. Wang, J. Li, S. Wang, and G. Wu, "A novel two-step job runtime estimation method based on input parameters in hpc system," in *2019 IEEE 4th International Conference on Cloud Computing and Big Data Analysis (ICCCBDA)*. IEEE, 2019, pp. 311–316.

[20] A. Matsunaga and J. A. Fortes, "On the use of machine learning to predict the time and resources consumed by applications," in *2010 10th IEEE/ACM International Conference on Cluster, Cloud and Grid Computing*. IEEE, 2010, pp. 495–504.

[21] G. Sabin, G. Kochhar, and P. Sadayappan, "Job fairness in non-preemptive job scheduling," in *International Conference on Parallel Processing, 2004. ICPP 2004.* IEEE, 2004, pp. 186–194.

[22] H. Lee, D. Lee, and R. S. Ramakrishna, "An enhanced grid scheduling with job priority and equitable interval job distribution," in *International Conference on Grid and Pervasive Computing*. Springer, 2006, pp. 53–62.

[23] D. Klusácek and H. Rudová, "Performance and fairness for users in parallel job scheduling," in *Workshop on Job Scheduling Strategies for Parallel Processing*. Springer, 2012, pp. 235–252.

[24] Š. Tóth and D. Klusáček, "User-aware metrics for measuring quality of parallel job schedules," in *Workshop on Job Scheduling Strategies for Parallel Processing*. Springer, 2014, pp. 90–107.