AMPRO-HPCC: A Machine-Learning Tool for Predicting Resources on Slurm HPC Clusters

Mohammed Tanash
Computer Science Department
Kansas State University
Manhattan, United States
e-mail: tanash@ksu.edu

Daniel Andresen
Computer Science Department
Kansas State University
Manhattan, United States
e-mail: dan@ksu.edu

William Hsu
Computer Science Department
Kansas State University
Manhattan, United States
e-mail: bhsu@ksu.edu

***Abstract*—Determining resource allocations (memory and time) for submitted jobs in High Performance Computing (HPC) systems is a challenging process even for computer scientists. HPC users are highly encouraged to overestimate resource allocation for their submitted jobs, so their jobs will not be killed due to insufficient resources. Overestimating resource allocations occurs because of the wide variety of HPC applications and environment configuration options, and the lack of knowledge of the complex structure of HPC systems. This causes a waste of HPC resources, a decreased utilization of HPC systems, and increased waiting and turnaround time for submitted jobs. In this paper, we introduce our first ever implemented fully-offline, fully-automated, stand-alone, and open-source Machine Learning (ML) tool to help users predict memory and time requirements for their submitted jobs on the cluster. Our tool involves implementing six ML discriminative models from the scikit-learn and Microsoft LightGBM applied on the historical data (sacct data) from Simple Linux Utility for Resource Management (Slurm). We have tested our tool using historical data (saact data) using HPC resources of Kansas State University (Beocat), which covers the years from January 2019 - March 2021, and contains around 17.6 million jobs. Our results show that our tool achieves high predictive accuracy R^2 (0.72 using LightGBM for predicting the memory and 0.74 using Random Forest for predicting the time), helps dramatically reduce computational average waiting-time and turnaround time for the submitted jobs, and increases utilization of the HPC resources. Hence, our tool decreases the power consumption of the HPC resources.**

***Keywords*—HPC; Scheduling; Supervised Machine Learning; Slurm; Performance.**

I. Introduction

High Performance Computing (HPC) resources have become more available to users to run their extensive computations and simulations. One of the most important parts of the HPC system is the batch scheduler. The batch scheduler manages resources and queues of all submitted jobs in the cluster. Hence, it is the part that decides where and when jobs will run in the cluster. On the other hand, batch scheduler performance depends on the resource requirements from the user such as the amount of memory, requested time, and the number of cores [1]. While these resource requirements are the responsibility of HPC users to determine, it is a fact that users may determine resource needs inaccurately [2]. Also, users are highly encouraged to overestimate these resources in order to satisfy job requirements, so their jobs will not be killed during the run time due to insufficient resources [3]. Overestimating job resource requirements negatively impacts the performance and the utilization of the HPC system. Moreover, over-estimating job resource process will increase average turn-around time and average waiting time for submitted jobs.

In this paper, we introduce the first-ever open-source, stand-alone, highly-accurate, fully-offline, and fully-automated tool called AMPRO-HPCC, which stands for "A Machine-Learning-Tool for Predicting Resources On Slurm HPC Clusters". AMPRO-HPCC aims to help HPC users predict and estimate the required job resource allocations (memory and time) for their submitted jobs. Our tool uses Simple Linux Utility for Resource Management (Slurm) historical logs-data (sacct) and involves implementation of six Machine Learning (ML) discriminative models from the scikit-learn [4] and Microsoft LightGBM (LGBM) [5]. Our ML tool is invoked through Command Line Interface (CLI), and it consists of two parts: **i)** System administrator part, which is responsible for preparing data and all the required models for building the final models and tool; **ii)** HPC user side, which will automatically read the submission job script provided from the HPC user and recommend the required job allocation resources (memory and time) for the associated submitted job.

We have extended our previous work [6]–[8], and designed the AMPRO-HPCC tool to help HPC users determine the allocation of HPC resource needs (memory and time) using supervised ML over historical data (sacct). Our open-source tool can be found on GitHub [9].

The rest of this paper is organized as follows: Section 2 discusses the related work. Section 3 describes our prediction tool, AMPRO-HPCC, which includes the workflow model, data preparation, evaluation and building of our Mixed Account Regression Model (MARM), and the job resource prediction. Section 4 shows our promising results. Finally, Section 5 presents our conclusion.

II. Related Work

Simple Linux Utility for Resource Management (Slurm) is a resource manager, which enables HPC resources to execute parallel jobs efficiently [10]. Slurm turns a set of hundreds or tens of thousands of computers into a single unit that you can

 ISBN: 978-1-61208-887-7

run jobs on. So Slurm makes parallel computers easy to use. Slurm allocates resources within a cluster, manages the nodes, and keeps track of architecture within a node such as sockets, NUMA boards, cores, hyper threads, memory, interconnect, generic resources, and managing licenses. Slurm manages jobs through varieties of scheduling algorithms (fair share, gang, advanced reservation, etc.) [11].

While there are many kinds of resource management scheduler such as Sun Grid Engine (SGE) [12], Tera-scale Open-source Resource and Queue manager (TORQUE) [13], [14], and Portable Batch System (PBS) [15], [16], Slurm is the most popular and most used among them. Hence, we implemented our tool based on Slurm workload manager HPC systems.

There are many studies and research focusing on predicting the running time and the time required for running application on the HPC systems or the cloud [17]–[29], while there are quite a lot of research that focuses on predicting the amount of memory required for the submitted jobs [30], [31].

Our work differs by the methodology used and the ability to predict both memory and time required for submitted jobs on the HPC systems. We conclude "there does not yet exist software that can help to fully automate the allocation of HPC resources or to anticipate resource needs reliably by generalizing over historical data, such as determining the number of processor cores and the amount of memory needed." [6]. Hence, we are introducing the first-ever open-source ML tool for predicting job resources (memory and time) for submitted jobs on the HPC systems.

III. Prediction Tool AMPRO-HPCC

Figure 1 illustrates the use-case diagram of our ML tool. We have two types of users: i) system administrators (referred to as admin henceforth) and ii) HPC users (referred to as users henceforth). Modules `PreProcess`, `BuildPerAccountModels`, `BuildMixedAccountModels` and `TrainSelectedMARM` are available to admins, while the `Ampro-hpcc` module is available to both admins and users. The main objective of our tool is to build Mixed Account Regression Models (MARM), which are regression models built on a subset of slurm *Accounts* with the best overall predictive performance, containing a reasonable percentage of jobs. Here, we provide descriptions of each module along with its inputs and outputs.

A. AMPRO-HPCC Workflow Model

Figure 2 describes the workflow model of our work as follows: i) The user prepares and creates a new job, which includes the requested amount of memory, time limit, quality of service (QoS), and partition name for the proposed job. ii) The HPC user will submit their job and passes it through our ML model in order to predict the amount of the required memory and the amount of time needed for the job to run. iii) Our ML model will process the submitted job by parsing all of the parameters needed, then predicting required memory and time for the specific job. iv) The HPC user will get feedback

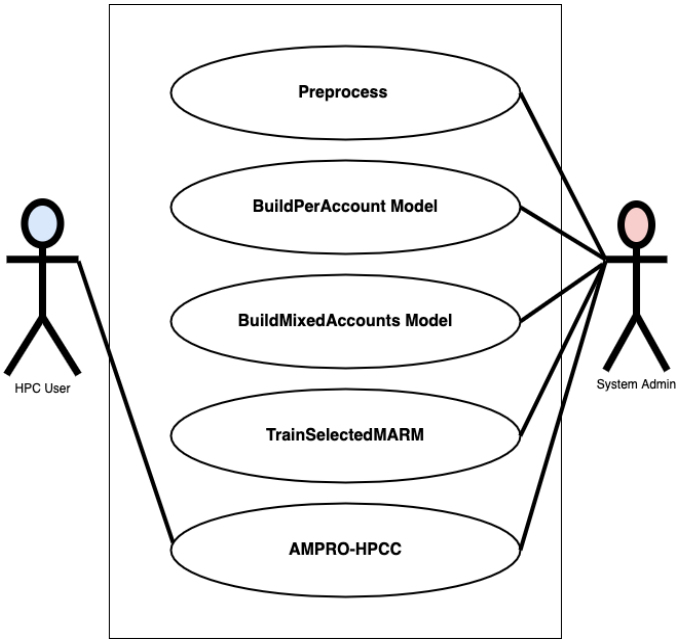

Fig. 1. Use-Case Diagram for AMPRO-HPCC

from our model regarding the needed amount of memory and time for their submitted jobs. v) The user will have the option to confirm or deny to use the predicted values for the required memory and time. vi) If the user confirms the use of the predicted amounts for either the required memory or the required time or both, then our ML model will update the amounts of memory and time as needed for the submitted job. If not, then the submitted job will remain the same. vii) The user will be notified about the changes to their jobs. viii) Finally, either an updated job or the original job will be scheduled for running on the cluster.

B. Data Preparation

The data preparation or `Preprocess` module takes the path (`path_to_data`) to logs of slurm jobs accounting information (sacct) to extract *Account, ReqMem, Timelimt, ReqNodes, ReqCPUS, QoS, Partition, MaxRSS, CPUTimeRAW, and State* from the dataset. A description of these features can be found at [32]. The module also asks the admin to provide default time-limit (`def_time`), default quality of service (`def_qos`), and default partition assignment (`def_partition`) to deal with some of the missing values in the data. Finally, the admin also has the ability to specify a set of QoS (`sel_qos`) and partitions (`sel_partition`) that they want to select over the entire data. In addition, the `Pre-processing` module does its own filtration by only selecting jobs with *State* equals to *'COMPLETED'*, and having non-zero *MaxRSS* and *CPUTimeRAW*. Next, this module standardizes *Timelimit* to numeric hours, *MaxRSS* and *ReqMem* to gigabytes (GB), and *Account* and *QoS* to numeric factors. Finally, *Account*, *ReqMem*, *ReqNodes*, *Timelimit*, *QoS*, *MaxRSS*, and *CPUTimeRAW* are normalized us-

 ISBN: 978-1-61208-887-7

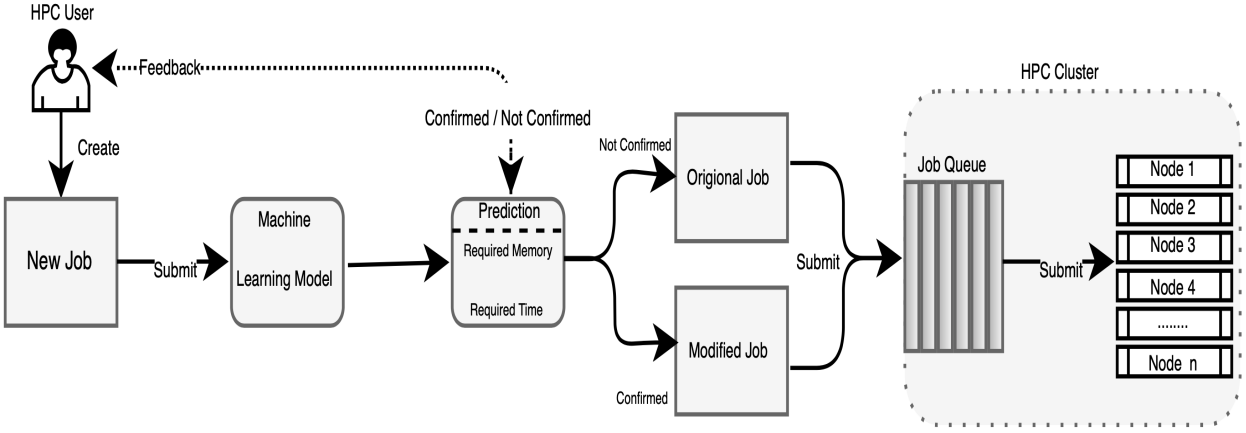

Fig. 2. AMPRO-HPCC Work-Flow Diagram.

ing the `StandardScaler` transform in Scikit-learn Python package [4].

C. Evaluating Individual Regression Models

Before building the *Mixed Account Regression Models* (MARM), the admin can evaluate individual regression models to note what may be most suited to their dataset. Although optional, the `BuildPerAccModels` module can provide initial insights on the quality of data and can significantly speed up MARM building time by nominating promising regression models for MARM overall possibilities. The `BuildPerAccModels` module requires the admin to provide the path to processed data (`path_to_data`), independent variables or features (`indep_vars`), and a dependent variable (`dep_var`) to train and evaluate seven popular regression models on all data-subsets containing individual *Account*. At this point, the admin can specify the minimum number of jobs an individual *Account* should have in order to be considered (`min_num_jobs`). The seven regression models include: i) Lasso Least Angle Regression (LL) [33], [34], ii) Linear Regression (LR) [34], iii) Ridge Regression (RG) [34], iv) Elastic Net Regression (EN) [34], v) Classification and Regression Trees (DTR) [35], vi) Random Forest Regression, (RFR) [36], and vii) LightGBM (LGBM) [5]. The regression models are evaluated by means of the Coefficient of determination (R^2), and root mean squared error (RMSE) [34]. We used scikit-learn's [4] implementation for all models and performance metrics.

D. Evaluating Mixed Account Regression Models

Once the individual regression models have been evaluated, the admin can select what models should be considered for MARM. The admin can also decide to select all seven regression models for MARM. Our `BuildMixedAccountModels` module requires a path to processed data (`path_to_data`), independent variables (`indep_vars`), dependent variable (`dep_var`), the minimum number of jobs (`min_num_jobs`), and the names of the regression models to be considered for MARM (`methodnames`). A mixed account regression model $MARM(N, M, X, Y)$ is constructed by finding N accounts with the best performance score for a given regression model M in predicting a dependent variable Y using independent variables X. MARM is constructed iteratively and can be summarized as follows:

$$MARM(N, M, X, Y) = \begin{cases} N^{'} & N = 1 \\ MARM(N-1, M, X, Y) \cup N^{'} & \text{otherwise} \end{cases}$$

where $N^{'} \in N$ is the *Account* that results in the best overall aggregate score in terms of R^2 on training ($R2_{tr}$) and testing ($R2_{te}$) datasets and number of jobs ($S_{N'}$), given by:

$$N^{'} = \arg\max_{n \in N}(R2_{tr}(M, X_{A[n]}, Y_{A[n]}), R2_{te}(M, X_{A[n]}, Y_{A[n]}), S_{A[n]})$$

where $X_{A[n]}$ and $Y_{A[n]}$ correspond to independent and dependent variables respectively for an unique *Account* $A[n]$. Thus, the MARM of N accounts depends upon the MARM of $N-1$ accounts appended with the best overall *Account* $N^{'}$ that results in the best overall performance. R^2 scores $R2_{tr}$ and $R2_{te}$ are calculated by randomly splitting the data into 80% (training) / 20% (testing), five times (5-fold) modeling using the regression model M, and averaging the R^2 scores on training and testing data subsets over the five runs. A comprehensive explanation of the Mixed Account Regression Model (MARM) can be found in our publication [7].

E. Building MARM for Prediction

The `BuildMixedAccountModels` module generates R^2 score distributions over $1, 2, \cdots, N$ for each regression model M specified by the admin in `methodnames`. Thus, the admin can determine which regression model performs the best along with the best number of accounts $\hat{n} \leq N$ to use. Thus, our `TrainSelectedMARM` module takes the selected regression model (`sel_model`), path to processed data (`path_to_data`), path to the intermediate results produced by `BuildMixedAccountModels` module (`path_to_marm_res`) independent variables

 ISBN: 978-1-61208-887-7

(indep_vars), dependent variable (dep_var) and number of accounts (num_acc) to build the final MARM for resource prediction.

F. Job Resource Prediction

Finally, the users of the slurm system can use Ampro-hpcc module by providing a path to their Slurm job submission script (path_to_script), a path to selected MARM model (path_to_model), a path to system default (path_to_defaults), and a path to the normalization transform (standard Scalar inverse transform) (path_to_stdscale) to obtain the recommended values of time and memory. To be conservative and prevent failure due to time and memory requirements that may underestimate of the actual memory and time utilization, our recommended values are increased by 10%.

IV. Results and Discussion

A. Preprocessing and PerAccount Models

We applied our ML tool using the HPC resources at Kansas State University, called Beocat. The data side has 17.6 million instances and covers the years 2018 - 2021 of the usage. After using PreProcessing module only selecting *'normal'* QoS, the dataset contained 7.8 million jobs spread across 21 unique accounts. Employing BuildPerAccountModels, we evaluated all seven regression models across 21 accounts, resulting in Figure 3 for predicting time (CPUTimeRAW) and memory (MaxRSS) that shows boxplots of R^2 and negative RMSE score distributions. We found LGBM, DTR, and RFR to be clear winners. Thus, we decided to only utilize LGBM, DTR, and RFR to build MARM.

B. MARM Models in Beocat

Utilizing BuildMixedAccountModels, we constructed MARMs to predict memory and time in Beocat using 17 out of 21 accounts (80% of the total accounts) in Beocat. Figure 4 shows the mean R^2 score distribution of DTR, RFR, and LGBM on training and testing datasets versus the number of best account combinations in predicting time. It can be seen that the R^2 decreases as the number of accounts (and jobs) increases. We found RFR was the best performer in predicting time, while LGBM was the best performer in predicting memory. Thus, we finalized the memory and time MARM using TrainSelectedMARM to be i) best five account combination (spanning across 1.8 million jobs) with an average R^2 of 0.74, for building an RFR based time model and ii) best thirteen accounts combination (spanning across 1.4 million jobs) with average R^2 of 0.72, for building an LGBM based memory model as shown in Figure 4. Using the finalized MARMs, we randomly sampled 5000 jobs from Beocat and ran them on a Slurm simulator with requested, actual, and predicted time and memory values.

C. Evaluating Our Model

We assessed our model using the Slurm simulator [37], [38], which was developed by the Center for Computational Research, SUNY Buffalo. The Slurm simulator was chosen because it is implemented from a modification of the actual Slurm code while disabling some unnecessary functions, which do not affect the functionality of the real Slurm [37].

Figure 5 shows submission and execution time, which indicates the difference between the job submission time (timestamp that represents when the job was submitted) and the execution time (difference between the start and end execution time) for five thousand jobs. Our results indicate that we have achieved almost identical running time compared to the actual running time.

Figure 6 measures and compares system utilization using requested jobs resources versus actual job resources versus predicted job resources using the AMPRO-HPCC tool. Our results show that our tool reached almost similar utilization compared to the utilization of the HPC system that used actual job resources because of the high prediction accuracy of our ML tool.

Figure 7 compares and assesses the backfill-sched algorithm's performance. The graph shows more efficient performance on the backfill-sched algorithm on the Beocat testbeds that used our ML module than the ones that did not. The graph shows fewer density results when using predicted values since using our AMPRO-HPCC model decreases the number of resources required by the user for the submitted jobs in most cases. This situation results in helping Slurm fit more jobs on the cluster. It also reduces the need to use the backfill-sched algorithm and resulting in more overall system efficiency by using these available resources.

Table 1 provides the calculated average waiting time, and average turn-around time for Beocat jobs for requested, actual, and predicted job resources allocation. Our results show that our tool was able to reduce the average waiting time for submitted jobs from 680 hours to 8.0 hours and the average turnaround time from 692 hours to 16.4 hours.

V. Conclusion

Determining the allocation of HPC resources for submitted jobs is a difficult process for HPC users. It is still an open question how many resources the user should specify (memory and time) for their submitted jobs on the cluster. HPC users are encouraged to overestimate job resources for their submitted jobs. In this paper, we have developed a novel and the first-ever open-source, stand-alone, fully-automated, highly-accurate, and fully-offline ML tool to help HPC users to determine the amount of required resources (memory and time) for their submitted jobs on the HPC clusters. Our tool was built using supervised ML algorithms. Our tool consists of two parts: i) the system admin part, which is responsible for preparing and building the ML model based on Slurm historical data and providing it to the users; ii) the user part, which uses the ML model provided from the system admin part, reads the submitted job script, and predicts the required

 ISBN: 978-1-61208-887-7

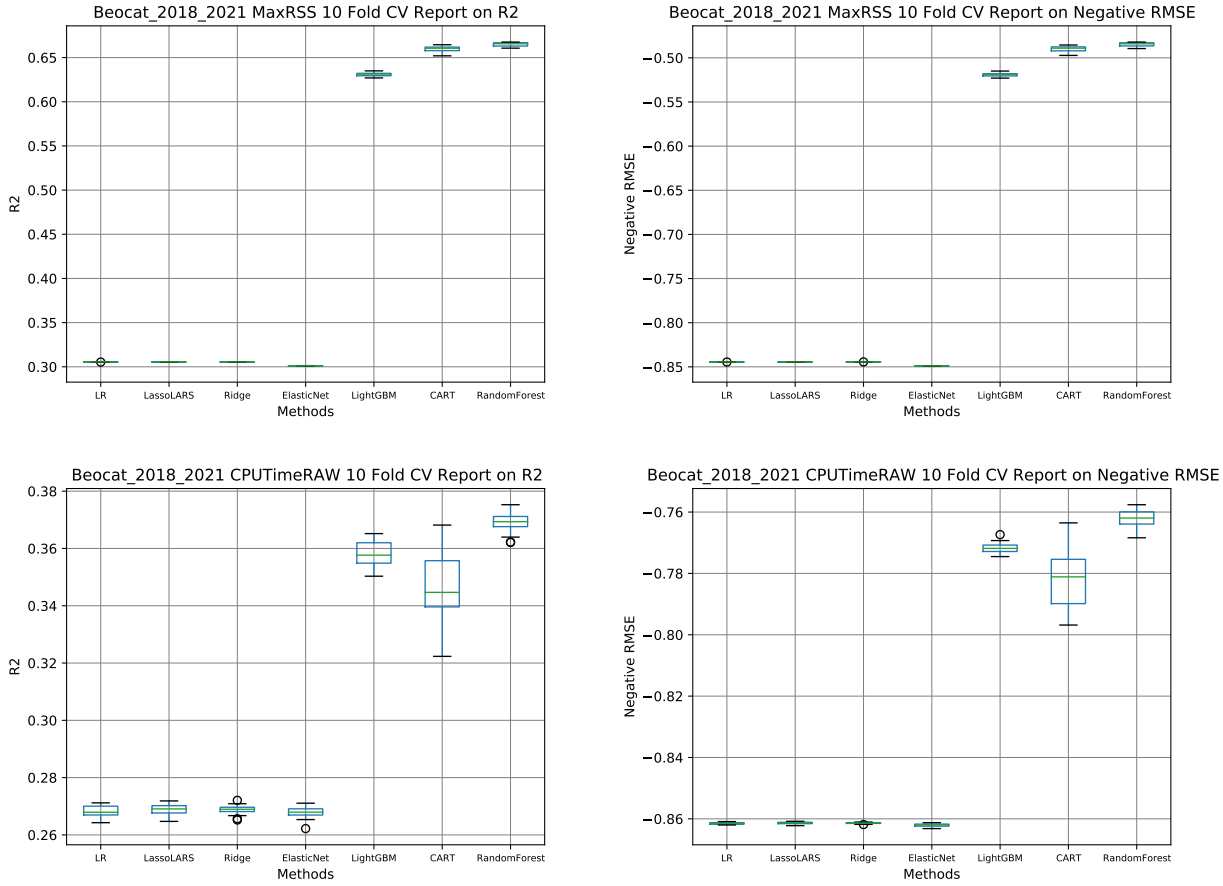

Fig. 3. R^2 and Negative RMSE of Seven Methods Across 21 Accounts in Beocat.

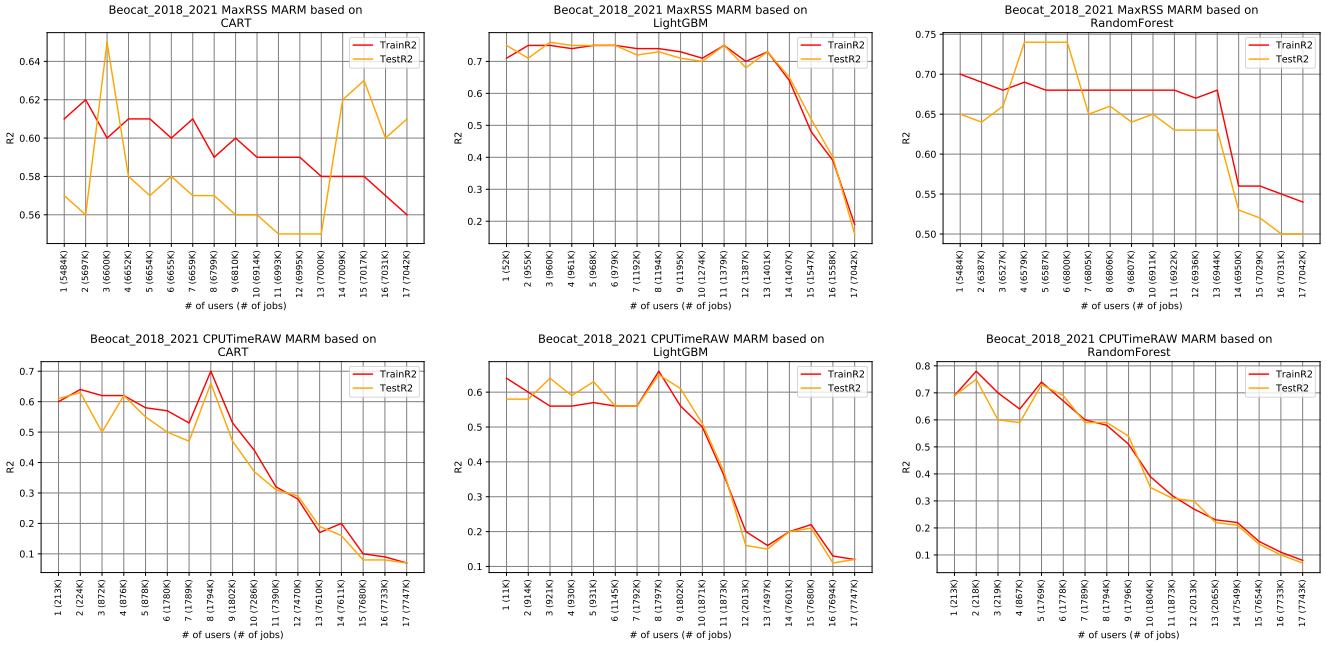

Fig. 4. R^2 Versus Number of Accounts in Predicting Memory and Time Using MARM Across Beocat

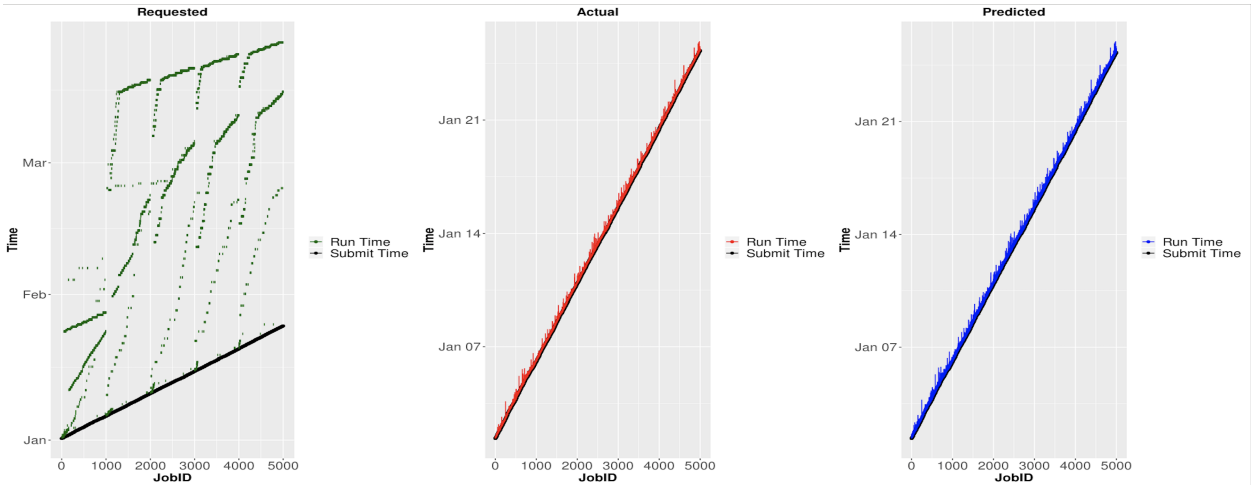

Fig. 5. Jobs Submission and Running Time. (Note Dramatic Improvement of Y Axis Range.)

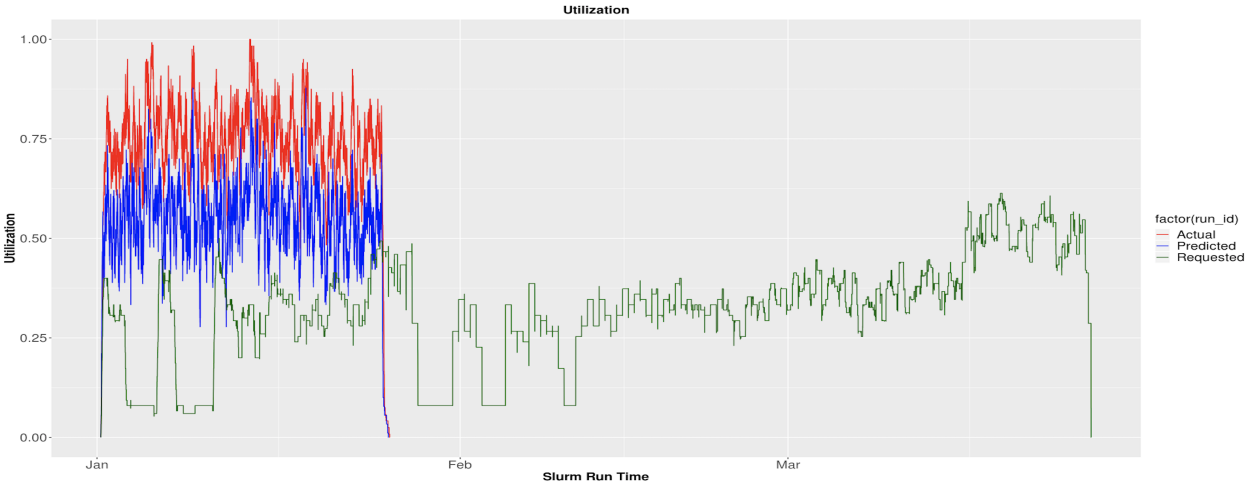

Fig. 6. Utilization (Requested vs Actual vs Predicted) for Beocat Jobs.

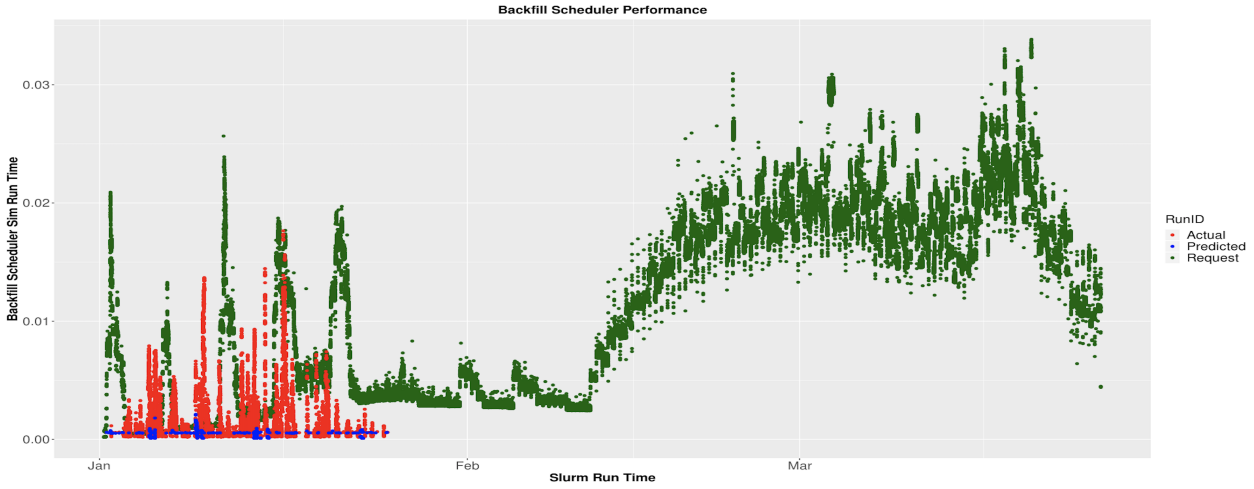

Fig. 7. Backfill-Sched Algorithm Performance (Requested vs Actual vs Predicted) for Beocat Jobs.

 ISBN: 978-1-61208-887-7

TABLE I
AVERAGE WAITING AND TURNAROUND TIME (REQUESTED VS ACTUAL VS PREDICTED) FOR BEOCAT

	Avg Wait Time (Hour)	Avg TA Time (Hour)	Median Wait Time (Hour)	Median TA Time (Hour)
Requested	680 ±128	692.8 ±130	713.6	715.6
Actual	0.4 ±0.08	3.62 ±1.8	0	3.09
Predicted	8.0±1.1	6.36 ±1.9	1.4	5.9

amount of the resources (memory and time). Our tool achieves high accuracy and can significantly increase the performance and utilization of the HPC systems. Moreover, our ML tool can dramatically decrease the average turnaround and waiting time for the submitted jobs. Hence, our tool increases the efficiency and decreases the power consumption of the Slurm-based HPC resources.

ACKNOWLEDGMENT

We thank the HPC staff at KSU, including Adam Tygart and Kyle Hutson, for their help and technical support. We also thank the authors of the Slurm simulator at SUNY Buffalo for releasing their work. This research was supported by NSF awards CHE-1726332, ACI-1440548, CNS-1429316, NIH award P20GM113109, and KSU.

REFERENCES

[1] 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.

[2] C. B. Lee, Y. Schwartzman, J. Hardy, and A. Snavely, "Are user runtime estimates inherently inaccurate?" in *Workshop on Job Scheduling Strategies for Parallel Processing*. Springer, 2004, pp. 253–263.

[3] M. Hovestadt, O. Kao, A. Keller, and A. Streit, "Scheduling in hpc resource management systems: Queuing vs. planning," in *Workshop on Job Scheduling Strategies for Parallel Processing*. Springer, 2003, pp. 1–20.

[4] F. Pedregosa *et al.*, "Scikit-learn: Machine learning in Python," *Journal of Machine Learning Research*, vol. 12, pp. 2825–2830, 2011.

[5] G. Ke *et al.*, "Lightgbm: A highly efficient gradient boosting decision tree," *Advances in neural information processing systems*, vol. 30, pp. 3146–3154, 2017.

[6] D. Andresen, W. Hsu, H. Yang, and A. Okanlawon, "Machine learning for predictive analytics of compute cluster jobs," *arXiv preprint arXiv:1806.01116*, 2018.

[7] M. Tanash, H. Yang, D. Andresen, and W. Hsu, "Ensemble prediction of job resources to improve system performance for slurm-based hpc systems," in *Practice and Experience in Advanced Research Computing*, 2021, pp. 1–8.

[8] M. Tanash *et al.*, "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.

[9] tanash1983, "Tanash1983/ampro-hpcc: A machine-learning-tool for predicting job resources on hpc clusters." [Online]. Available: https://github.com/tanash1983/AMPRO-HPCC

[10] A. B. Yoo, M. A. Jette, and M. Grondona, "Slurm: Simple linux utility for resource management," in *Workshop on job scheduling strategies for parallel processing*. Springer, 2003, pp. 44–60.

[11] "Slurm workload manager - documentation," https://slurm.schedmd.com/, retrieved: 06, 2021.

[12] W. Gentzsch, "Sun grid engine: towards creating a compute power grid," in *Proceedings First IEEE/ACM International Symposium on Cluster Computing and the Grid*. IEEE Comput. Soc. [Online]. Available: https://doi.org/10.1109/ccgrid.2001.923173

[13] G. Staples, "Torque resource manager," in *Proceedings of the 2006 ACM/IEEE conference on Supercomputing*, 2006, pp. 8–es.

[14] "Torque resource manager," http://www.adaptivecomputing.com/products/torque/, retrieved: 06, 2021.

[15] B. Nitzberg, J. M. Schopf, and J. P. Jones, "Pbs pro: Grid computing and scheduling attributes," in *Grid resource management*. Springer, 2004, pp. 183–190.

[16] "Pbs professional open source project," https://www.pbspro.org/, retrieved: 05, 2021.

[17] 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.

[18] T.-P. Pham, J. J. Durillo, and T. Fahringer, "Predicting workflow task execution time in the cloud using a two-stage machine learning approach," *IEEE Transactions on Cloud Computing*, vol. 8, no. 1, pp. 256–268, 2017.

[19] S. Kim, Y.-K. Suh, and J. Kim, "Extes: An execution-time estimation scheme for efficient computational science and engineering simulation via machine learning," *IEEE Access*, vol. 7, pp. 98 993–99 002, 2019.

[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. [Online]. Available: https://doi.org/10.1109/ccgrid.2010.98

[21] A. Tyryshkina, N. Coraor, and A. Nekrutenko, "Predicting runtimes of bioinformatics tools based on historical data: five years of galaxy usage," *Bioinformatics*, vol. 35, no. 18, pp. 3453–3460, 2019.

[22] M. Naghshnejad and M. Singhal, "Adaptive online runtime prediction to improve hpc applications latency in cloud," in *2018 IEEE 11th International Conference on Cloud Computing (CLOUD)*. IEEE, 2018, pp. 762–769.

[23] 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.

[24] F. Nadeem, D. Alghazzawi, A. Mashat, K. Faqeeh, and A. Almalaise, "Using machine learning ensemble methods to predict execution time of e-science workflows in heterogeneous distributed systems," *IEEE Access*, vol. 7, pp. 25 138–25 149, 2019.

[25] M. H. Hilman, M. A. Rodriguez, and R. Buyya, "Task runtime prediction in scientific workflows using an online incremental learning approach," in *2018 IEEE/ACM 11th International Conference on Utility and Cloud Computing (UCC)*. IEEE, 2018, pp. 93–102.

[26] D. Ardagna *et al.*, "Predicting the performance of big data applications on the cloud," *The Journal of Supercomputing*, pp. 1–33, 2020.

[27] Y.-K. Suh, S. Kim, and J. Kim, "Clutch: A clustering-driven runtime estimation scheme for scientific simulations," *IEEE Access*, vol. 8, pp. 220 710–220 722, 2020.

[28] O. Aaziz, J. Cook, and M. Tanash, "Modeling expected application runtime for characterizing and assessing job performance," in *2018 IEEE International Conference on Cluster Computing (CLUSTER)*. IEEE, 2018, pp. 543–551.

[29] T. Saillant, J.-C. Weill, and M. Mougeot, "Predicting job power consumption based on rjms submission data in hpc systems," in *International Conference on High Performance Computing*. Springer, 2020, pp. 63–82.

[30] T. Taghavi, M. Lupetini, and Y. Kretchmer, "Compute job memory recommender system using machine learning," in *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 2016, pp. 609–616.

[31] E. R. Rodrigues, R. L. Cunha, M. A. Netto, and M. Spriggs, "Helping hpc users specify job memory requirements via machine learning," in *2016 Third International Workshop on HPC User Support Tools (HUST)*. IEEE, 2016, pp. 6–13.

 ISBN: 978-1-61208-887-7

[32] "Slurm workload manager," retrieved: 04, 2021. [Online]. Available: https://slurm.schedmd.com/sacct.html
[33] B. Efron *et al.*, "Least angle regression," *Annals of statistics*, vol. 32, no. 2, pp. 407–499, 2004.
[34] G. Bonaccorso, *Machine learning algorithms*. Packt Publishing Ltd, 2017.
[35] W.-Y. Loh, "Classification and regression trees," *Wiley interdisciplinary reviews: data mining and knowledge discovery*, vol. 1, no. 1, pp. 14–23, 2011.
[36] L. Breiman, "Random forests," *Machine learning*, vol. 45, no. 1, pp. 5–32, 2001.
[37] N. A. Simakov *et al.*, "A slurm simulator: Implementation and parametric analysis," in *International Workshop on Performance Modeling, Benchmarking and Simulation of High Performance Computer Systems*. Springer, 2017, pp. 197–217.
[38] "Github - ubccr-slurm-simulator/slurm_simulator: Slurm simulator: Slurm modification to enable its simulation," https://github.com/ubccr-slurm-simulator/slurm$_simulator, retrieved : 05, 2021.

 ISBN: 978-1-61208-887-7