Improving HPC System Performance by Predicting Job Resources via Supervised Machine Learning

Mohammed Tanash
Kansas State University
Manhattan, Kansas
tanash@ksu.edu

Brandon Dunn
Kansas State University
Manhattan, Kansas
brdunn@ksu.edu

Daniel Andresen
Kansas State University
Manhattan, Kansas
dan@ksu.edu

William Hsu
Kansas State University
Manhattan, Kansas
bhsu@ksu.edu

Huichen Yang
Kansas State University
Manhattan, Kansas
huichen@ksu.edu

Adedolapo Okanlawon
Kansas State University
Manhattan, Kansas
arokanlawon@ksu.edu

ABSTRACT

High-Performance Computing (HPC) systems are resources utilized for data capture, sharing, and analysis. The majority of our HPC users come from other disciplines than Computer Science. HPC users including computer scientists have difficulties and do not feel proficient enough to decide the required amount of resources for their submitted jobs on the cluster. Consequently, users are encouraged to over-estimate resources for their submitted jobs, so their jobs will not be killing due insufficient resources. This process will waste and devours HPC resources; hence, will lead to inefficient cluster utilization. We created a supervised machine learning model and integrated it into the Slurm resource manager simulator to predict the amount of required memory resources (Memory) and the required amount of time to run the computation. Our model involved using different machine learning algorithms. Our goal is to integrate and test the proposed supervised machine learning model on the Slurm. We used over 10000 tasks selected from our HPC log files to evaluate the performance and the accuracy of our integrated model. The purpose of our work is to increase the performance of the Slurm by predicting the amount of the required jobs memory resources and the time required for each particular job in order to improve the utilization of the HPC system using our integrated supervised machine learning model.

Our results indicate that for larger jobs our model helps dramatically reduce computational turnaround time (from five days to ten hours for large jobs), substantially increased utilization of the HPC system, and decreased the average waiting time for the submitted jobs.

CCS CONCEPTS

• **Computing methodologies** → **Supervised learning**; *Artificial intelligence*; • **Software and its engineering** → **Scheduling**; • **Hardware** → *Testing with distributed and parallel systems*;

KEYWORDS

HPC, Scheduling, Supervised Machine Learning, Slurm, Performance, User Modeling

PEARC'19, July 2019, Chicago, Illinois USA

ACM ISBN 123-4567-24-567/08/06.
https://doi.org/10.1145/nnnnnnn.nnnnnnn

1 INTRODUCTION

HPC systems have become more well-known and available to users among the universities and research centers, to name a few. Users rely on running their extensive computations on these machines. One of the most critical parts of the HPC system is the scheduler, which is a piece of software on a high-performance computing cluster which decides and controls what calculations to run next and wherein the HPC systems [22]. Schedulers can become a bottleneck for HPC systems through handling vast numbers of submitted jobs that are requesting an extensive amount of cluster resources (CPUs and memory). Users of the HPC systems come from different disciplines. Particular fields in science and engineering such as atmospheric sciences, chemical separations, astrophysics, geoinformation science, and evolutionary biology rely on and demand HPC resources through simulations, experiments, and dealing with a tremendous amount of data [11] [21]. These users are usually not familiar and do not have a good knowledge and experience to estimate what exactly their jobs need, and the scheduler does not know any better. Calculating the resource needs for a particular job is a hard thing even for computer scientists. On the other hand, HPC users are implicitly encouraged to overestimate predictions in terms of memory, CPUs, and time so they will avoid severe consequences and their jobs will not be killed due to an insufficient amount of resources. Overestimate job resources will negatively impact the performance of the scheduler by wasting infrastructure resources; lower throughput; and leads to longer user response time.

1.1 Slurm Workload Manager

There are different varieties of job schedulers such as SGE (Sun Grid Engine) [14], Maui Cluster Scheduler [2], TORQUE (Tera-scale Open-source Resource and Queue manager) [6], and PBS (Portable Batch System) [4]. Slurm (Simple Linux Utility for Resource Management) which is one of the most popular among all of them [22]. Slurm is an open source; fault tolerant; secure; highly configurable; highly scalable, and support most of Unix variants .Slurm role is both workload manager and a job scheduler, which makes Slurm

more convenient to use. Resource manager role is allocating resources such as nodes, sockets, cores, hyper-threads, memory, interconnect, and other generic resources within the HPC environment. While the scheduler role is managing the queue of work jobs including different scheduling algorithms such as fair share scheduling, preemption, gang scheduling, advanced reservation, etc. [5].

1.2 Slurm Simulator

In order to test our module, we implemented a machine learning module and testing it using the *Slurm simulator* developed by Center for Computational Research, SUNY University at Buffalo. The Slurm simulator is located in the Github [3]. The Slurm simulator was developed to help the administrators to choose the best Slurm configuration while avoiding impacting the existing production system. We used this Slurm simulator because it is implemented from a modification of the actual Slurm code while disabling some unnecessary functionalities which does not affect the functionality of the real Slurm, and it can simulate up to 17 days of work in an hour [19]. hence, we can test our models accurately and quickly.

Slurm is a vital component of supercomputers but using it is hard, and this leads to inefficiencies. Hence, we are trying to use supervised machine learning to address these efficiencies. This entails first defining inference tasks: regression-based estimation of the probability of a job being killed given its runtime parameters and given a user's historical track record to date; a classification-based prediction of the outcome of the current run, computed by estimating the odds of specific outcomes (or log odds, in the case of logistic regression), and finally an expected utility based on probability distribution over outcomes. While the first two use cases are purely predictive and solvable by supervised or semisupervised inductive learning, the third presents an opportunity for sequential problem solving, towards reinforcement learning-based automation (learning to act).

We are focused on developing a predictive analytics capability for Slurm so it can predict needed amount of memory resources and required running time for each particular submitted jobs (regression). We hope to improve the efficiency of Slurm and the HPC systems itself by increase system throughput; increase system utilization; decrease turnaround time, and decrease average job waiting time. To do so, we train different models with different machine learning algorithms described in Section 3. In Section 4 we present the results of our experiments, and conclude in Section 5.

2 RELATED WORK

The primary research conducted in a related field of study focused on predicting the length of time of the jobs temporarily waiting in the queue. Besides, the previous research either predicted memory usage of the jobs or predicted the execution time of the jobs running on the cluster. The central point and novel contribution of our study is to predict and determine the resources needed to accomplish the jobs submitted on the cluster and determine which is more harmful for the HPC system, overestimate the memory or the time for the jobs running on the cluster?

Matsunaga and Fortes [18] introduced an extended machine learning tree algorithm called Predicting Query Runtime 2 (PQR2). This method is a modified implementation of an existing classification tree algorithm (PQR). PQR2 focused on the two bioinformatics applications, BLAST, and RAxML. Their method increased the accuracy of predicting the job execution time, memory and space usage, and decreased the average percentage error for those applications.

Warren Smith [20] introduced a lower prediction error rate machine learning method based on instance-based learning (IBL) techniques to predict job execution times, queue wait time, and file transfer time.

Kumar and Vadhiyar [16] developed a prediction system called Predicting Quick Starters (PQStar) for identified and prediction of quick starters jobs (jobs who has waiting time < 1 hour). PQStar prediction based on jobs request size and estimated run-time time, queue and processor occupancy states.

García [12] study and found that automatically collecting and combining real performance running job data specifically "memory bandwidth usage of applications", and scheduling data that extracted from the hardware counters during jobs execution and used it again in the future for scheduling purposes can improve HPC scheduling performance and reduce the amount of waste resources and decrease the number of killed jobs due to reaching their execution time limit.

Gaussier et al. found that using a more limited approach to machine learning on HPC log data to predict jobs running time is an effective method for helping and improving scheduling algorithms and reduced the average bounded slowdown [13].

Other works focused on predict and maximize power consumption for scientific applications and maximize performance using machine learning techniques [9] [10] .

3 IMPLEMENTATION

In this section, we will explain the workflow for our model, our machine learning algorithms used in our model, the data and the experimental testbeds used, and the features used for our machine learning modeling.

3.1 Workflow Model

The workflow model of our work described in Figure 1 as follows. 1) The user submits their job which is including the amount of memory and requested time limit for the proposed job. 2) The submitted job will be passed through our machine learning model to predict the amount of the required memory and the amount of time needed for the job to run. 3) Our model will update the amount of memory resources and update the amount of time required for the submitted job. 4) The user will be notified about the changes to their jobs. 5) Finally, The updated job will be scheduled for running on the cluster.

3.2 Data Preparation and Feature Analysis

For training our machine learning model, we used **fourteen million** instances which cover approximately eight years of log history data between the years 2009 to 2017 from our local HPC cluster, "Beocat." Each instance on the log file has forty-five features. We chose eight features as described in Table 1 in each instance of the fourteen million total instances used for training the machine learning model. Beocat is no cost educational system, and the most

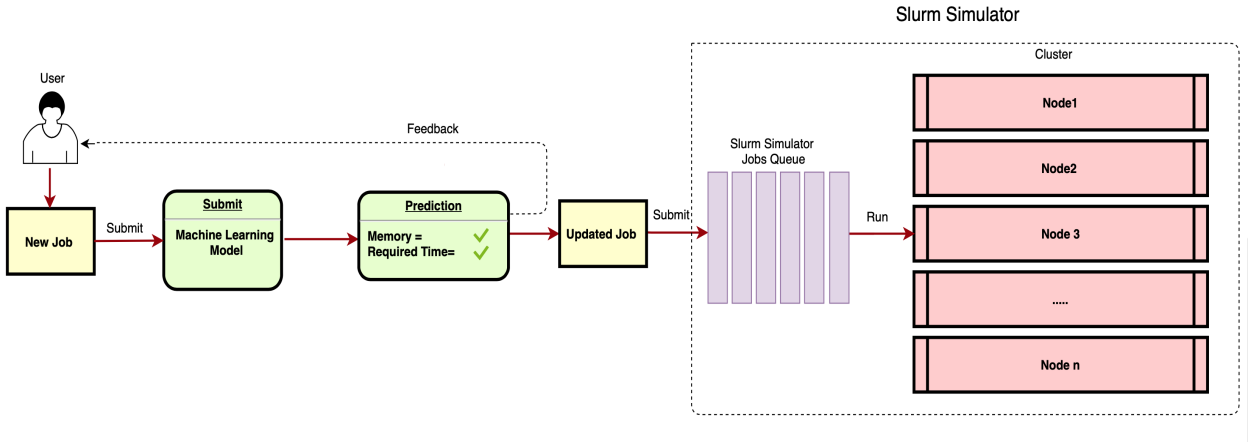

Figure 1: Work Flow Diagram for our Model.

significant cluster in the state of Kansas.. It is located at Kansas State University and operated by the Computer Science department [1].

3.3 Machine Learning Algorithms

Several discriminative models from the **scikit-learn** machine learning library [7] [17] were trained to implement predictive functionality in our experiments. Data preparation steps included data cleaning by means of validating the data model for logged data and applying transformations to normalize the data, reduce redundancies, and otherwise standardize the coalesced data model. For the baseline predictive task, we specified a classification target: specifically, learning the concept of a job that is more likely than not to be killed given historical and runtime variables. This admits the use of a logistic regression or logit model, support vector machines, or k-nearest neighbor, whereas for the planned expected utility estimation task, estimating the actual probability of a job being killed is a genera regression task [15] that admits linear, distance-weighted, or support vector regression, as well as probit and generative models.

For the regression task, we used several supervised models, including linear regression, LassoLarsIC (L1 regularization), ridge regression (L2 regularization), ElasticNetCV (L1/L2 ratio), and a decision tree regressor. For the linear discriminants and their use on this task, we refer the interested reader to [8]. Using these flexible representations admits a balance of generalization quality (via overfitting control) and explainability.

4 RESULTS AND DISCUSSION

In this section, we describe, discuss and evaluate our machine learning algorithms results, and the strategy used for our experiment by presenting results and graphs consisting of quantitative metrics.

4.1 Machine Learning Techniques

There are various machine learning algorithms available, and it is difficult to decide which supervised machine learning algorithm provided the best results for our module. Hence, we implemented our model using five supervised machine learning algorithms and trained them using our 14 million instances to predict the required time and memory. The statistical measures of the coefficient of determination of the machine learning algorithms shown in Table 2 and Table 3 respectively. Based on our results we chose **DecisionTreeRegressor** algorithm in our model since it has the most significant R-squared value which means the most fitted data to the regression line.

The legend for **Table 2** and **table 2** described as follows:

- **LR**: Linear Regression
- **LLIC**: LassoLarsIC Regression
- **ENCV**: ElasticNetCV Regression
- **RG**: Ridge Regression
- **DTR**: Decision Tree Regression

4.2 Evaluating Our Model

In this subsection, we show results and evaluate our model. To do so, we test our model using two testbeds *(Testbed-1)* and *(Testbed-2)*. Each testbed is evaluated based on three metrics as follows:

- **Submission and Execution Time**
- **System Utilization**
- **Backfill-Sched Performance**

Submission and Execution Time shows the difference between the job submission time and the execution time (when the job is submitted, start and duration of the run). **System Utilization** measure how efficiently the system is utilizing the resources, while the *Backfill-Sched Performance* shows the performance of the backfill-sched algorithm helping the main scheduler to fit more jobs within the cluster to increase resource utilization.

We used the Slurm Simulator to examine each metric above by comparing the results of the following:

- Running each testbed using user **requested** memory and run time.
- Running each testbed using the **actual** memory usage and duration.
- Running each testbed using **Predicted** memory and predicted run time.

Table 1: Feature Selected

Feature	A Type	Description
job _id	Numeric	Id of submitted job
username	Text	User name of submitted job
submit	Date	Date and time to submit job
wclimit	Numeric	Requested time in minutes (predicted variable)
duration	Numeric	Actual running wall time for the job in seconds
cpu_per_task	Numeric	Number of requested CPU's per task
req_mem	Numeric	Requested memory for job at submission time in MB (predicted variable)
req _mem _per _cpu	Numeric	Required memory per CPU

Table 2: Wall Clock Time Limit Prediction Algorithms Results

Model	R^2 (%)	Time (Second)
LLIC	0.0677	0.30
LLIC	0.0677	0.44
ENCV	0.0677	4.32
RG	0.0677	0.18
DTR	0.611	7.53

Table 3: Memory Required Prediction Algorithms Results

Model	R^2 (%)	Time (Second)
LR	0.174	0.39
LLIC	0.174	0.46
ENCV	0.174	4.98
RG	0.174	0.12
DTR	0.638	8.28

Table 4: Average Waiting and Turnaround Time (Requested vs Actual vs Predicted) For Jobs in Testbed-1

	Avg Wait Time (Hour)	Avg TA Time (Hour)
Requested	45.37	46.29
Actual	3.90	4.82
Predicted	4.00	4.94

Table 5: Average Waiting and Turnaround Time (Requested vs Actual vs Predicted) For Jobs in Testbed-2

	Avg Wait Time (Hour)	Avg TA Time (Hour)
Requested	0.08	3.90
Actual	0.05	3.23
Predicted	0.06	3.54

4.2.1 ***Testbed-1***. Testbed-1 contains larger jobs (jobs which are requesting at least 4GB of memory and four cores per task). *Testbed-1* includes a set of a thousand jobs. Figure 2 shows submission and execution time metric based on the job_id, start time, and the execution time for **(Requested vs. Actual vs. Predicted)** for the jobs included in Testbed-1. The graph shows that it takes around **five days** to complete the execution for all of the jobs using user requested memory and time, while it takes only around **ten hours** to complete the running for the jobs using the actual and predicted time and memory for the jobs. Based on the results, our model predicted the values for the required time and memory accurately.

Figure 3 shows that using our module helped the HPC system achieved higher utilization compared to the utilization of the HPC system that used unmodified user requested resources. Figure 4 indicates that the backfill-sched algorithm has achieved more efficiency on the testbed that used our module compared to the ones that did not.

These results were achieved because using our model in most cases reduces the amount of resources required by the user submitted jobs. Hence, the HPC system has more available resources to fit more jobs in the system. Thus, the backfill schedule becomes less needed and the overall system more efficient by using these available resources.

Table 4 provides the calculated **average waiting time** and **average turn-around time** for the jobs in *Testbed-1* for each requested, actual, and predicted runs. Using our model significantly reduced the average waiting time from **45.37** hours to **3.9** hours and average turnaround time from **46.29** hours to **4.94**. Both predicted average waiting time and turn-around time is almost exactly the same as the actual average waiting time and turnaround time for jobs in *testbed-1*.

4.2.2 ***Testbed-2***. Testbed-2 contains smaller jobs (jobs which are requesting less than 4GB of memory and four cores per task). *Testbed-2* includes a set of **ten thousand jobs**.

While the results were less impressive than Testbed-2, Figure 5 and 6 shows that our predicted model achieved better utilization and better backfilling performance. Moreover, Table 5 shows that our predicted model incrementally reduced the average waiting and turnaround time from **(0.08 to 0.06 hours)** and from **(3.90 to 3.54 hours)** respectively.

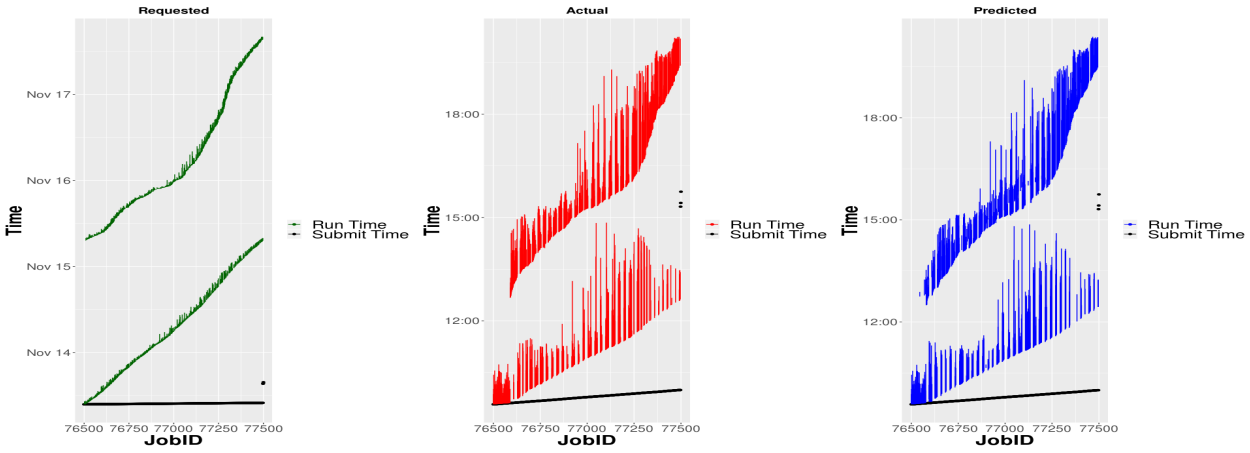

Figure 2: Jobs Submission and Running time (Requested vs Actual vs Predicted) for Jobs in Testbed-1. Note dramatic improvement of Y axis range between graphs.

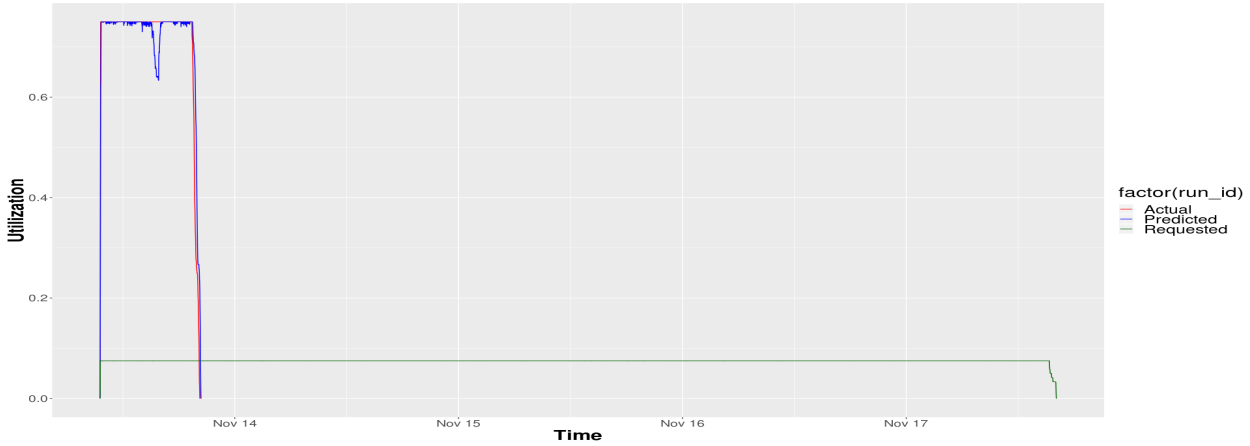

Figure 3: Utilization (Requested vs Actual vs Predicted) for Jobs in Testbed-1.

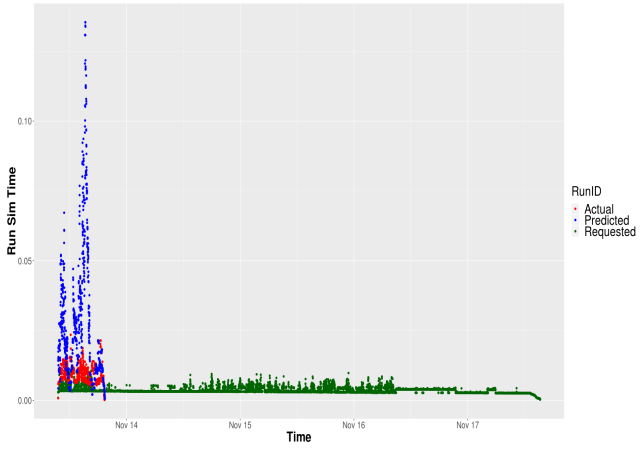

Figure 4: Backfill-Sched Performance for Jobs in Testbed-1

4.3 Predicting Memory Required vs. Predicting Time Required

In this results subsection, we will discuss and show the results that answer a question **"Which is more important to predict? Required memory or required time?"**

Figure 7 shows the submission and running times for two runs of Testbed-1. One run is using our model where we are predicting only the required memory (Red) and the other one predicting the required time (Blue). This is mostly caused by inaccurate estimation of the time and memory equally by jobs submitted by the users.

Figure 8 and 9 shows the comparison of the utilization and the performance of backfill-sched for the system by running jobs in Testbed-1 on the Slurm Simulator using ((Requested vs Actual vs Required Time Predicted vs Memory Predicted vs Required Time and Memory Predicted).

Our results indicates that both memory prediction and time requested prediction are highly valuable and are almost equally important because they achieved similar performance as shown

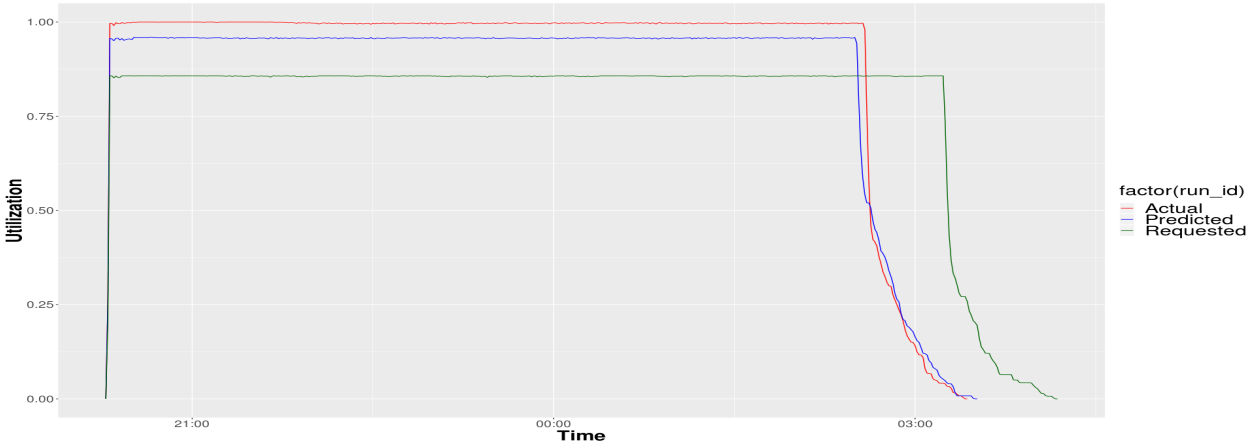

Figure 5: Utilization (Requested vs Actual vs Predicted) For Testbed-2.

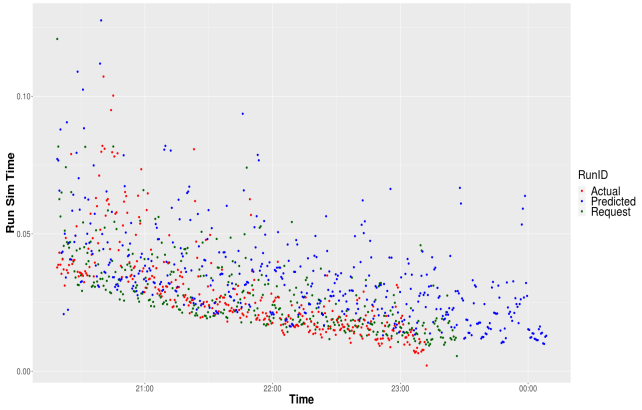

Figure 6: Backfill-Sched Performance for Testbed-2)

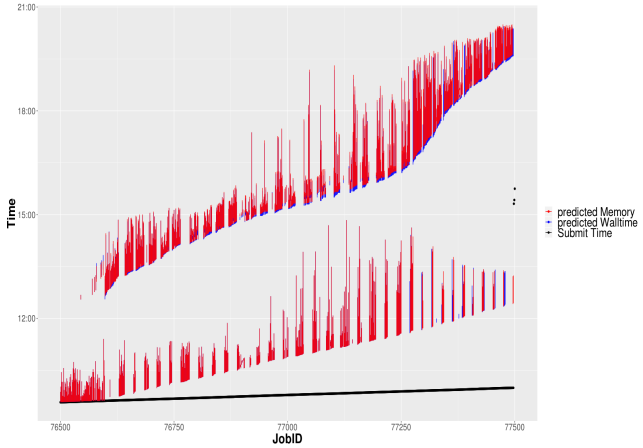

Figure 7: Jobs Submission and Running time (Predicted Time Required vs Memory)

in the graphs. We achieved peak performance and utilization by combining both of them in one model.

5 CONCLUSIONS

Our model is an important link between HPC users, scheduler, and HPC resources. The rule of our model is predicting the amount of memory and time required for any submitted jobs using supervised machine learning algorithms. Our model helps to reduce computational time, increase utilization of the HPC system, decrease average waiting time, and decrease the average turn-around time for the submitted jobs. As a result, our analysis indicates that our model helps maximize efficiency, increase the capability, and decrease the power consumption of the cluster.

6 FUTURE WORK

Our future work will include continuing improving our model by applying additional machine learning algorithms, testing our module in a real HPC system and including a bigger testbed to achieve more accurate results.

In addition, our machine learning approach will incorporate both the classification (logit and other discriminative modes) and regression (probit estimation and other maximum likelihood estimation) into a decision support system. This system will provide a test bed for personalized recommendations and experimental evaluation of the pros, cons, and effectiveness of off-loading large jobs to the cloud service. As a use case of data science it will also facilitate exploration of variables that are exogenous to a single job, such as a user's history of job submission and rates of success or failure by mode (memory vs. CPU). This can also potentially provide insights into the effectiveness of training and the skill acquisition curve of a user as related to self-efficacy (as indicated on surveys) and as discovered automatically by clustering of users.

ACKNOWLEDGMENTS

We are greatly appreciate the HPC staff at Kansas State University, including Adam Tygart and Kyle Hutson, for their help and technical support. We also thank the authors of the Slurm simulator at

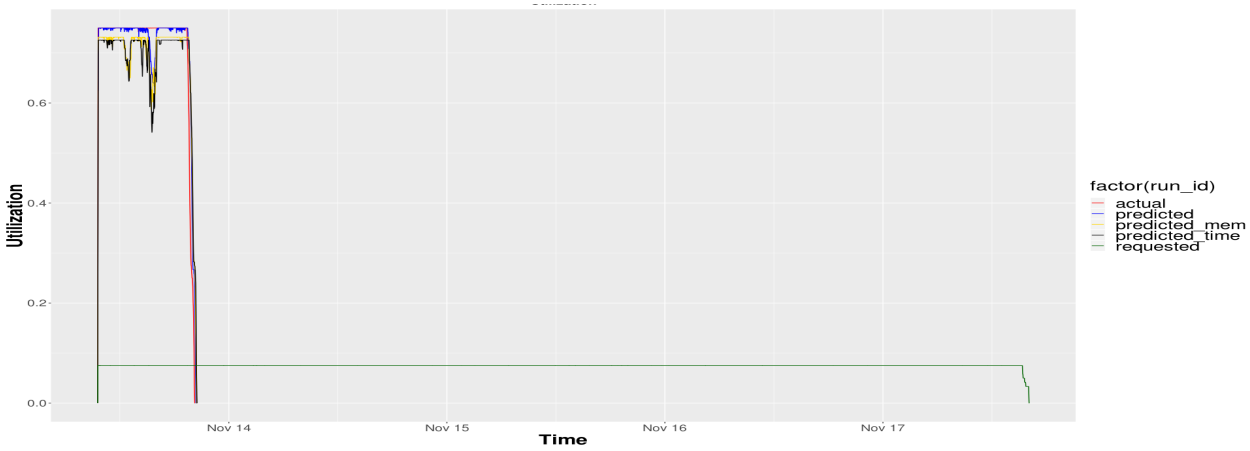

Figure 8: Utilization (Requested vs Actual vs Required Time Predicted vs Memory Predicted vs Required Time and Memory Predicted).

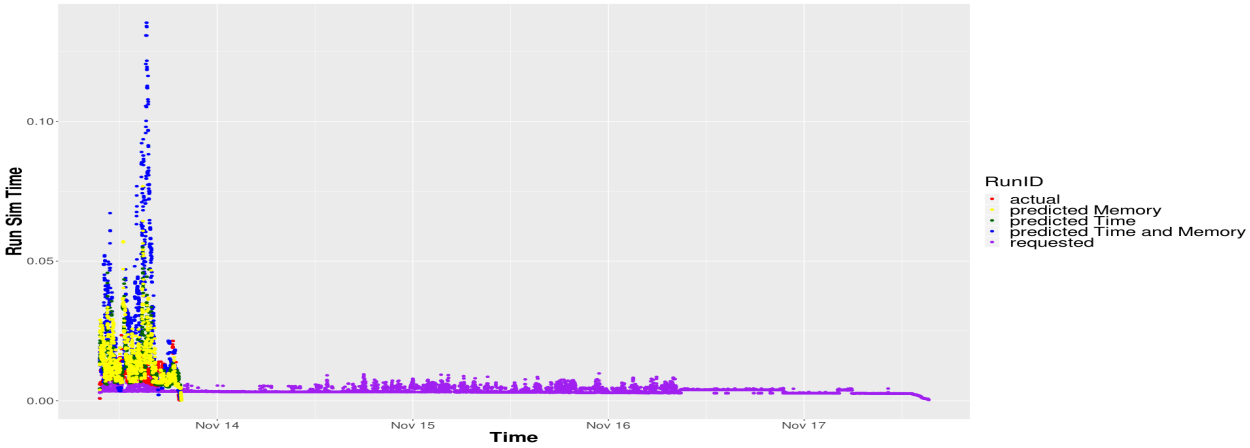

Figure 9: Backfill-Sched Performance for (Requested vs Actual vs Required Time Predicted vs Memory Predicted vs Required Time and Memory Predicted)

SUNY U. of Buffalo for releasing their work. This research was supported by NSF awards CHE-1726332, ACI-1440548, CNS-1429316, NIH award P20GM113109, and Kansas State University.

REFERENCES

[1] [n. d.]. Beocat. https://support.beocat.ksu.edu/BeocatDocs/index.php/Main_Page. (Accessed on 03/013/2019).
[2] [n. d.]. Documentation Index. http://www.adaptivecomputing.com/support/documentation-index/. (Accessed on 02/011/2019).
[3] [n. d.]. GitHub - ubccr-slurm-simulator/slurm_simulator: Slurm Simulator: Slurm Modification to Enable its Simulation. https://github.com/ubccr-slurm-simulator/slurm_simulator. (Accessed on 01/03/2019).
[4] [n. d.]. PBS Professional Open Source Project. https://www.pbspro.org/. (Accessed on 02/03/2019).
[5] [n. d.]. Slurm Workload Manager - Documentation. https://slurm.schedmd.com/. (Accessed on 01/07/2019).
[6] [n. d.]. TORQUE Resource Manager. http://www.adaptivecomputing.com/products/torque/. (Accessed on 02/02/2019).
[7] 2019. Getting Started with Scikit-learn for Machine Learning. In *Python® Machine Learning*. John Wiley & Sons, Inc., 93–117. https://doi.org/10.1002/9781119557500.ch5
[8] Dan Andresen, William Hsu, Huichen Yang, and Adedolapo Okanlawon. 2018. Machine Learning for Predictive Analytics of Compute Cluster Jobs. *CoRR* abs/1806.01116 (2018). arXiv:1806.01116 http://arxiv.org/abs/1806.01116
[9] Josep Ll. Berral, Íñigo Goiri, Ramón Nou, Ferran Julià, Jordi Guitart, Ricard Gavaldà, and Jordi Torres. 2010. Towards energy-aware scheduling in data centers using machine learning. In *Proceedings of the 1st International Conference on Energy-Efficient Computing and Networking - e-Energy '10*. ACM Press. https://doi.org/10.1145/1791314.1791349
[10] Bruce Bugbee, Caleb Phillips, Hilary Egan, Ryan Elmore, Kenny Gruchalla, and Avi Purkayastha. 2017. Prediction and characterization of application power use in a high-performance computing environment. *Statistical Analysis and Data Mining: The ASA Data Science Journal* 10, 3 (Feb. 2017), 155–165. https://doi.org/10.1002/sam.11339
[11] N.R. Council, D.E.L. Studies, D.E.P. Sciences, and C.P.I.H.E.C.I.F.S. Engineering. 2008. *The Potential Impact of High-End Capability Computing on Four Illustrative Fields of Science and Engineering*. National Academies Press. https://books.google.com/books?id=2XadAgAAQBAJ
[12] Fenoy GarcÃıa and Carlos. 2014. Improving HPC applications scheduling with predictions based on automatically collected historical data. https://upcommons.upc.edu/handle/2099.1/23049
[13] Eric Gaussier, David Glesser, Valentin Reis, and Denis Trystram. 2015. Improving backfilling by using machine learning to predict running times. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis on - SC '15*. ACM Press. https://doi.org/10.1145/2807591.2807646

[14] W. Gentzsch. [n. d.]. 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. https://doi.org/10.1109/ccgrid.2001.923173

[15] S. B. Kotsiantis, I. D. Zaharakis, and P. E. Pintelas. 2007. Machine learning: a review of classification and combining techniques. https://link.springer.com/article/10.1007/s10462-007-9052-3

[16] Rajath Kumar and Sathish Vadhiyar. 2013. Identifying Quick Starters: Towards an Integrated Framework for Efficient Predictions of Queue Waiting Times of Batch Parallel Jobs. In *Job Scheduling Strategies for Parallel Processing*, Walfredo Cirne, Narayan Desai, Eitan Frachtenberg, and Uwe Schwiegelshohn (Eds.). Springer Berlin Heidelberg, Berlin, Heidelberg, 196–215.

[17] L. Massaron and A. Boschetti. 2016. *Regression Analysis with Python*. Packt Publishing. https://books.google.com/books?id=d2tLDAAAQBAJ

[18] Andréa Matsunaga and José A.B. Fortes. 2010. 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. https://doi.org/10.1109/ccgrid.2010.98

[19] Nikolay A. Simakov, Martins D. Innus, Matthew D. Jones, Robert L. DeLeon, Joseph P. White, Steven M. Gallo, Abani K. Patra, and Thomas R. Furlani. 2018. A Slurm Simulator: Implementation and Parametric Analysis. In *High Performance Computing Systems. Performance Modeling, Benchmarking, and Simulation*, Stephen Jarvis, Steven Wright, and Simon Hammond (Eds.). Springer International Publishing, Cham, 197–217.

[20] Warren Smith. 2007. Prediction Services for Distributed Computing. In *2007 IEEE International Parallel and Distributed Processing Symposium*. IEEE. https://doi.org/10.1109/ipdps.2007.370276

[21] Chaowei Yang, David Wong, Qianjun Miao, and Ruixin Yang (Eds.). 2010. *Advanced Geoinformation Science*. CRC Press. https://doi.org/10.1201/b10280

[22] Andy B. Yoo, Morris A. Jette, and Mark Grondona. 2003. SLURM: Simple Linux Utility for Resource Management. In *Job Scheduling Strategies for Parallel Processing*, Dror Feitelson, Larry Rudolph, and Uwe Schwiegelshohn (Eds.). Springer Berlin Heidelberg, Berlin, Heidelberg, 44–60.