Ensemble Prediction of Job Resources to Improve System Performance for Slurm-Based HPC Systems

MOHAMMED TANASH, HUICHEN YANG, DANIEL ANDRESEN, WILLIAM HSU, Kansas State University

In this paper, we present a novel methodology for predicting job resources (memory and time) for submitted jobs on HPC systems based on historical jobs data (saccount data) provided from the Slurm workload manager using supervised machine learning. This Machine Learning (ML) prediction model is effective and useful for both HPC administrators and HPC users. Moreover, our ML model increases the efficiency and utilization for HPC systems, thus reduce power consumption as well. Our model involves using Several supervised machine learning discriminative models from the scikit-learn machine learning library and LightGBM applied on historical data from Slurm.

Our tool helps HPC users to determine the required amount of resources for their submitted jobs and make it easier for them to use HPC resources efficiently. This work provides the second step towards implementing our general open source tool towards HPC service providers. For this work, our tool has been implemented and tested using two HPC providers, an XSEDE service provider (University of Colorado-Boulder (RMACC Summit) and Kansas State University (Beocat)).

We used more than two hundred thousand jobs: one-hundred thousand jobs from SUMMIT and one-hundred thousand jobs from Beocat, to model and assess our ML model performance. In particular we measured the improvement of running time, turnaround time, average waiting time for the submitted jobs; and measured utilization of the HPC clusters.

Our model achieved over **85%** accuracy in predicting the amount of time and the amount of memory for both SUMMIT and Beocat HPC resources. Our results show that our model helps dramatically reduce computational average waiting time **(from 380 to 4 hours in RMACC Summit and from 662 hours to 28 hours in Beocat)**; reduced turnaround time **(from 403 to 6 hours in RMACC Summit and from 673 hours to 35 hours in Beocat)**; and acheived up to **100%** utilization for both HPC resources.

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

Additional Key Words and Phrases: HPC, Scheduling, Supervised Machine Learning, Slurm, Performance, User Modeling

1 INTRODUCTION

HPC systems have become more well-known and available to users among 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 where in the HPC systems [42]. 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). Most of the High Performance Computing (HPC) users come from disciplines different

Author's address: Mohammed Tanash, Huichen Yang, Daniel Andresen, William Hsu, Kansas State University, 2184 Engineering Hall 1701D Platt St. Manhattan, Kansas, 66506, [tanash,huichen,dan,bhsu]@ksu.edu.

from computer science, such as atmospheric sciences, chemical modeling, astrophysics, geo-information science, and evolutionary biology, etc [13] [41]. Determining the allocation of HPC resources such as determining the amount of memory and the number of processor cores for submitted jobs is a difficult process, because of a lack knowledge about the structure and implementation of HPC systems, running applications, and size of submitted jobs. Moreover, there is no software that can help to predict and determine the needed resources required for submitted jobs. 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. The overestimation of job resources negatively impacts the performance of the scheduler by wasting infrastructure resources; lower throughput; and leads to longer user response times.

We extended our earlier work [9], improving and expanding our previous work [37] [27] by designing a predictive ML model for Slurm based HPC resources; improving predictive accuracy; and achieving better results for our ML predictive model. Our work focuses on accurately predicting and determining the required amount of resources (time and memory) for submitted jobs and making it easier for users to use HPC resources efficiently. Hence, increasing efficiency, decreasing waiting and turnaround time for submitted jobs, and decreasing power consumption for HPC systems. This work provides a big step towards implementing our open source software tool to predict and determine the needed resources required for submitted jobs.

Our ML tool involves implementing different machine learning algorithms (five discriminative models from the scikit-learn and Microsoft LightGBM) applied on the historical data (sacct data) from Simple Linux Utility for Resource Management (Slurm). Our tool will increase the utilization and help to decrease the power consumption of the HPC resources.

In this work, our tool has been implemented for two HPC resources (An XSEDE service provider, University of Colorado-Boulder (SUMMIT), and Kansas State University(BEOCAT)). In this paper, we answered several research questions: Can we enhance the performance of HPC resources by applying supervised machine learning algorithms on Slurm-based HPC historical data? How accurate is our proposed model in terms of prediction of the resource needed for submitted jobs? Can we create a tool that helps HPC users decide the amount of resources needed for their submitted jobs? Our work focuses on making the process of deciding the amount of the required resources (memory and time) accurate and easy for the submitted jobs.

1.1 Why the Slurm Workload Manager and Slurm Simulator?

Simple Linux Utility for Resource Management (Slurm) is a resource manager which makes HPC resources executes parallel jobs efficiently [43]. Slurm turns a set of hundreds or tens of thousands of computers into a single unit that you can run jobs on. So Slurm makes parallel computers easy to use. Slurm allocates resources within a cluster, and manages the nodes and keeps track of architecture within a node such as sockets, NUMA boards, core, hyper threads, memory, interconnect, generic resources, managing licences. It manages jobs through varieties of scheduling algorithms (fair share, gang, advanced reservation, etc.) [5]. Hence, improving the performance of Slurm will increase the efficiency of the HPC systems. Slurm can become a bottleneck of the HPC system through handling the big amount of these requested resources. The scheduler decides and controls what calculations to run next, and where in the cluster [42]. Slurm is the most popular job scheduler over all of other schedulers such as SGE (Sun Grid Engine) [17], TORQUE (Tera-scale Open-source Resource and Queue manager) [34] [6], PBS (Portable Batch System) [26] [4], and the Maui Cluster Scheduler [2]. Testing our model on a real cluster is a hard process due to security and reliability issues, impacting real cluster and the time needed to run all of jobs needed for testing. Hence, The Slurm simulator developed

by Center for Computational Research, SUNY University at Buffalo was the best way in order to be able to test our model accurately and quickly. On the other hand the Slurm simulator was chosen for testing because it is implemented from a modification of the actual Slurm code while disabling some unnecessary functionalities which do not affect the functionality of the real Slurm, and it can simulate up to 17 days of work in an hour [33]. The Slurm simulator is located in the Github [3]

2 RELATED WORK

Job scheduling in HPC systems affected by several factors such as job submission time, job duration time, Requested Processors, Requested Time, Requested Memory, Group ID, etc. [16]. While there are some other main scheduling strategies that researchers focuses on the past by using different priority functions such as Shortest Job First (SJF), First Come First Serve (FCFS), Priority Scheduling, etc. [31] which don't achieve maximum performance and utilization in multiple nodes clusters. Previous studies show that batch job scheduling is NP-complete problem [39].

There has been many studies focused in predicting the run time for running application on the HPC systems or on the cloud [28] [30] [20] [22] [38] [25] [40] [24] [18] [10] [35]

While there are a very few studies focuses in predicting memory for running jobs on the cluster such as Taghavi et al. who introduced a machine learning recommender system for predicting the amount of memory for jobs submitted to Load Sharing Facility (LSF ®). [36].

Fan et al. proposed another HPC scheduling techniques using deep reinforcement learning (DARS), which uses neural networks for resource reservation and backfilling. [15].

Similarly Zhang et al. proposed a deep reinforcement learning-based job scheduler called RLScheduler which is capable of learning high quality scheduling policy. [44]. Some other work focused on measuring power consumption based on submitted jobs on a cluster. [32].

Tyryshkina et al. used three machine learning algorithms: extra trees regressor, the gradient boosting regressor and the random forest regressor for predicting run time for bioinformatics tools and found best performance is by using random forests. [32]

Other different method presented by Aaziz et al. for measuring the performance of a running job. The method uses historical application data (job parameters and hardware counter metrics) of a specific applications during the running time. This method increases the application overhead by 5% [7].

Our work combines both predicting memory and time required for submitted jobs on HPC system. In addition we could not find any work that tested their work in a real or simulated resource workload manager as we did for Slurm.

Our work propose a zero overhead stand alone ML model which dramatically improves the efficiency, and utilization. In addition, our model will decreases turnaround time, waiting time for the submission jobs.

3 METHODOLOGY

In this section we will explain the workflow model; Data Preparation and Feature Analysis; Regression Models; and multi-technique prediction: Mixed Account Regression Models.

3.1 Workflow Model

Figure 1 described the workflow model of our work as follows. **i)** The user prepare and create a new job which is including 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 passed it through our machine learning model in order to predict the

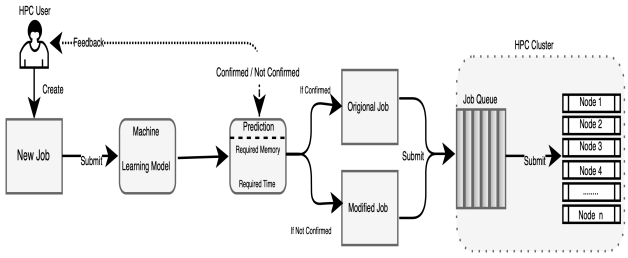


Fig. 1. Work-Flow Diagram for our Model.

Table 1. Feature Selected

Feature	Type	Description
Account	Text	Account the job ran under.
ReqMem	Text	Minimum required memory for the job (in MB per CPU or MB per node).
Timelimit	Text	Timelimit set for the job in [DD-[HH:]]MM:SS format.
ReqNodes	Numeric	Requested number minimum Node count.
ReqCPUS	Numeric	Number of requested CPUs.
QOS	Text	Name of Quality of Service.
Partition	Text	The partition on which the job ran.
MaxRSS	Numeric	Maximum resident set size of all tasks in job (in MB).
CPUTimeRAW	Numeric	Time used (Elapsed time * CPU count) by a job (in seconds).
State	Text	The job status.

amount of the required memory and the amount of time needed for the job to run. **iii)** Our Machine learning model will process the submitted job through parsing all of the parameters needed, then predict the required memory and time for the specific job. **iv)** The HPC user will get a feedback 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 machine Learning model will update the amounts 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 updated job or the original job will be scheduled for running on the cluster.

3.2 Data Preparation and Feature Analysis

Two data sets (sacct data) were collected from Slurm database. The first data set was collected from the HPC resources of the XSEDE service provider at the University of Colorado Boulder (RMACC Summit) [8]. The data set has **7.8 million** instances and cover the years from 2016 – 2019 of the usage. The second data set was collected from the HPC resources of Kansas State University (Beocat) [1]. The data side has **10.9 million** instances and cover the years 2018-2019.

Given logs of accounting information for jobs invoked with Slurm and HPC system specific requirements such as default time-limit, memory-limit, quality of service (QOS), partition, etc., we began by extracting useful features from the data, as shown in **table 1**. Among the features, State and Partition were used for filtering, while others were used for data modeling. We selected CPUTimeRAW over Elapsed time as the 'time to predict' because it naturally incorporates the number of requested CPUs into actual runtime. It is well known that using more CPUs does not

necessarily translate to reduced runtime [23] due to the limited bandwidth of memory access, overhead in resource management and protection, etc., thus, we considered CPUTimeRAW as a relatively robust and conservative estimate of runtime over Elapsed time. The selected features were processed in three stages: **i)** data treatment and filtration **ii)** feature standardization and **iii)** data normalization.

Data Treatment and Filtration. : At this stage, we dealt with missing values (*NaN*) in the data and removed certain types of jobs. Based on the respective HPC system policies, missing Timelimit, Partition and QOS were replaced with default values. Jobs missing values for either MaxRSS or CPUTimeRAW were removed. Jobs belonging to premium Partition / QOS were removed; so were jobs with incomplete State ('Cancelled', 'Failed', 'Deadline', etc.), or 'Unlimited' Timelimit. Post filtration, we had **4.45 million** jobs left in Beocat, while RMACC Summit had **2.8 million** jobs left.

Feature Standardization. : Timelimit was parsed to numeric hours. MaxRSS was standardized to gigabytes (GB). Account, QOS and Partitions were factorized to unique integer codes. ReqMem was converted from MB per CPU (suffix c) or MB per node (suffix n) to a numeric total MB, and was subsequently standardized to GB.

Data Normalization. : Only Account, ReqMem, ReqNodes, Timelimit, QOS, MaxRSS and CPUTimeRAW were selected for further processing and analysis. All seven features except QOS and Account were normalized by shifting to their respective means and scaling by their respective standard deviation, using the *StandardScalarTransform()* in Scikit-learn Python package [29].

3.3 Regression Models

Our objective was to model time (CPUTimeRAW) and memory (MaxRSS) as a function of requested parameters Account, Timelimit, ReqNodes, ReqMem, ReqCPUS and QOS. Thus, we considered the following seven popular regression models for the task: **i)** Lasso Least Angle Regression (LL) [11, 14], **ii)** Linear Regression (LR) [11], **iii)** Ridge Regression (RG) [11], **iv)** Elastic Net Regression (EN) [11], **v)** Classification and Regression Trees (DTR) [21], **vi)** Random Forest Regression (RFR) [12], and **vii)** LightGBM (LGBM) [19]. We used the Coefficient of determination (R^2) [11] and root mean squared error (RMSE) [11] to evaluate the regression models. We used scikit-learn's [29] implementation for all models and performance metrics.

3.4 Multi-Technique prediction: Mixed Account Regression Models

Using Timelimit, ReqNodes, ReqMem, ReqCPUS and QOS to predict CPUTimeRAW and MaxRSS, we found that it was challenging to faithfully model all job accounting information for large HPC systems. This happened due to a plethora of job requests having identical amount of requested resources, but leading to substantially different actual resource allocation. Such jobs, invariant in independent variables and highly variant in the dependent variable, resulted in sub-par performance across all regression models.

Nevertheless, we found that partitioning the data by 'Account' led to significant improvements in performance in certain slices of data, suggesting that some accounts had better job request specifications, predictive of the actual parameters. Thus, instead of modeling the entire data, we deliberately built a mixed account regression model by iteratively adding best performing accounts to an account pool until peak performance is reached with reasonable data utilization. Algorithm 1 shows the Mixed Account Regression Model (MARM) that takes a dependent variable Y, independent variables X, account ids *acc* for each observation in X and Y, number of accounts to consider *num_acc*, and a regression model m.

Algorithm 1 MARM(Y, X, *acc*, *num_acc m*)

```
unacc = unique(acc)
acc_pool = {}
Repeat num_acc times:
    for i ∈ unacc
        tac = append(acc_pool, i), if not i ∈ acc_pool
        indices = which tac ∈ acc
        X_a, Y_a = X[indices], Y[indices]
        Repeat 20 times:
            Split X_a, Y_a into 80% training and 20% testing.
            RM = Build model using m.
            Calculate training and testing R^2 of RM.
        R2_tr[i] = mean R^2 of RM using training data.
        R2_te[i] = mean R^2 of RM using testing data.
    best_aid = Choose i with best mean R2_tr and R2_te ranks.
    best_r2_tr = R2_tr[best_aid]
    best_r2_te = R2_te[best_aid]
    acc_pool = append(acc_pool, best_aid)
Return best_r2_tr, best_r2_te, acc_pool
```

The algorithm begins with an empty account pool (acc_pool). Accounts are added to a temporary account pool (tac) using the current account pool (acc_pool) and an account i among unique accounts *unacc*, only if i does not already exist in acc_pool, next we find a data subset X_a, Y_a corresponding to accounts in tac. The data subset is then randomly split 20 times into 80% (training) / 20% (testing) ratio and modeled using the regression model m. In each run R^2 scores on training and testing data are computed, that are averaged and stored in $R2_{tr}[i]$ for training data and in $R2_{te}[i]$ for testing data. Finally, after all unique accounts in *unacc* have been evaluated, we select the best account id (best_aid) based on $R2_{tr}$, $R2_{te}$ to be added to the current account pool. The algorithm continues until acc_pool contains *num_acc* accounts.

In principle, MARM is a greedy-strategy to find best the 'n' account combination to maximize performance. The growing account pool (acc_pool) finds the best account combinations starting from one to 'n', assuming that the best i account combination is constructed by the union of the best $i-1$ account combination and an account that results in the best R^2 performance. Thus, a Slurm administrator can set *num_acc* to the total number of accounts N in the HPC system and use MARM to generate a R^2 score distribution along with all best account combinations (one to N) and the number of jobs covered by these accounts. The administrator can then choose the number of account combination that offers the best performance across reasonable number of jobs.

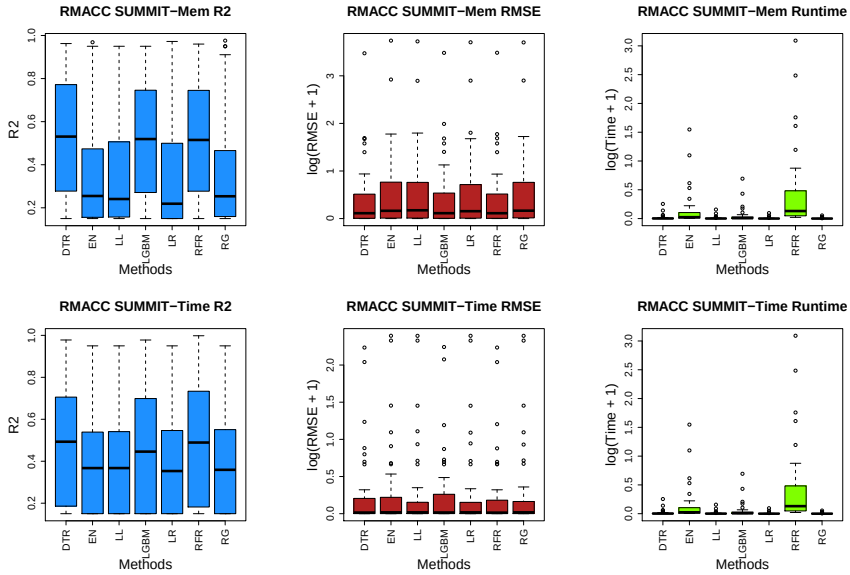
4 RESULTS AND DISCUSSION

In this section we will explain the enchmarking predictive performance of regression models;he MARM models for Beocat and RMACC Summit; and Evaluating Our Model.

4.1 Benchmarking predictive performance of regression models

Instead of directly using all seven regression models for building MARM, we benchmarked these models to select ones that offered superior empirical evidence of effectiveness. We evaluated all seven methods based on their performance

Fig. 2. R2, RMSE and Runtime of seven methods across 50 accounts in RMACC SUMMIT and 20 accounts in BEOCAT.



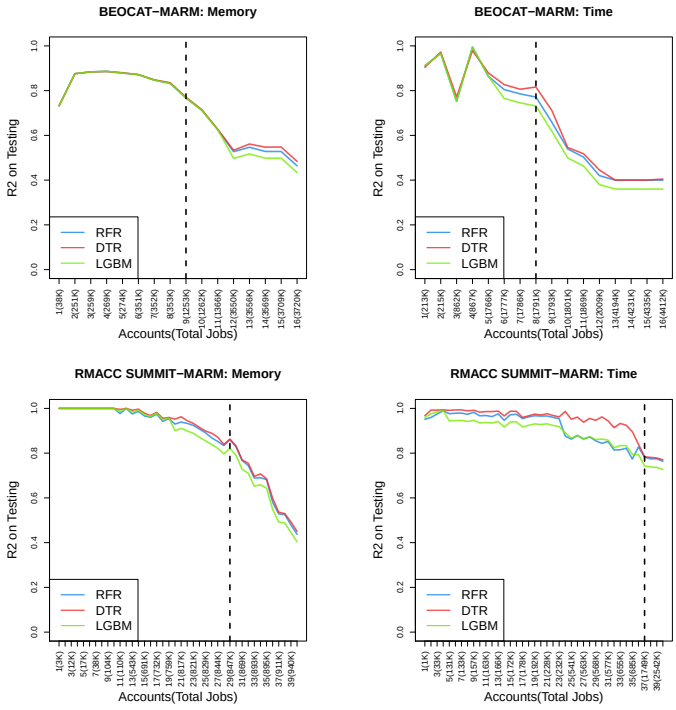
across data slices corresponding to single accounts in predicting CPUTimeRAW (Time) and MaxRSS (Memory). Beocat contained 4.45 million jobs spread across 20 unique accounts, while RMACC Summit contained 2.8 million jobs spread across 50 unique accounts. We employed 5-fold cross validation and used average R^2 and RMSE obtained on testing data, as well as average time to build the model, as our performance metrics. Figure 2 shows boxplots illustrating the distribution of average R^2, log (RMSE+1) and log (runtime+1) in predicting memory and time among 50 accounts in RMACC Summit for each regression model. We found similar trends among 20 accounts in Beocat.

LGBM, DTR and RFR are similarly outstanding in R^2 performance in both memory and time. RMSE does not show significant variation among the seven methods. In terms of runtime, RFR is consistently the slowest of all methods followed by EN. Based on these results, we decided to move forward with LGBM, RFR and DTR for building MARMs for Beocat and RMACC Summit.

4.2 The MARM models for Beocat and RMACC Summit

We constructed MARMs to predict memory and time in Beocat and RMACC Summit using **80%** of the total accounts, **40 out of 50 accounts** in RMACC Summit and **16 out of 20 accounts** in Beocat. **Figure 3** shows the mean R^2 score distribution of DTR, RFR and LGBM on testing data versus number of best account combinations. It can be seen that the R^2 decreases as number of accounts (and jobs) increase. While all three methods DTR, RFR and LBGM perform similarly, DTR does slightly better across all cases. The dotted lines show the best number of account combinations we choose to build memory and time models across the two datasets. In particular we choose, **i)** best **nine accounts** combination (spanning across 1.25 million jobs) with average R^2 of **0.77**, for building a DTR based memory model in BEOCAT; **ii)** best **eight accounts** combination (spanning across 1.8 million jobs) with average R^2 of **0.81**, for building a DTR based time model in BEOCAT; **iii)** best **29 accounts** combination (spanning 847,000 jobs) with average R^2 of **0.86**,

Fig. 3. R^2 versus Number of Accounts in predicting memory and time using MARM across BEOCAT and RMACC Summit



for building a DTR based memory model in RMACC Summit; and **iv)** best **37 accounts** combinations (spanning across 1.74 million jobs) with average R^2 of **0.78**, for building a DTR based time model in RMACC Summit.

4.3 Evaluating Our Model

In order to assess our model, we have examined our model using two testbeds (RMACC Summit testbed) and (Beocat testbed). Each testbed contains one hundred thousand jobs. Each testbed was assessed based on three metrics **i)** Submission and Execution Time, which indicate the difference between the job submission time (timestamps that represent when the job was submitted) and the execution time (the difference between the start and end execution time). **ii)** System Utilization which measure how effective and efficient the system utilizing its resources. **iii)** Backfill-Sched Performance, shows the performance of the backfill-sched algorithm assisting the main scheduler to schedule more jobs within the cluster to enhance resource utilization. We used the Slurm Simulator to assess each metric above by comparing the results of running each testbed using users requested memory and run time; using actual memory usage and duration; and using our ML model predicted memory and run time. Each RMACC Summit and Beocat testbeds contains one hundred thousand jobs. **Figure 4** and **Figure 7** shows submission and execution time metric based on the job-id, start time, and the execution time for (Requested vs. Actual vs. Predicted) for **five thousand jobs** included in RMACC Summit Testbed and **five thousand jobs** included in Beocat testbed, respectively. The graphs show it takes around **fifty-five days** for RMACC Summit and **eighty-two days** for Beocat to complete the execution for all of the jobs using user requested memory and time, while it takes only **twenty-four** days for both RMACC Summit and Beocat

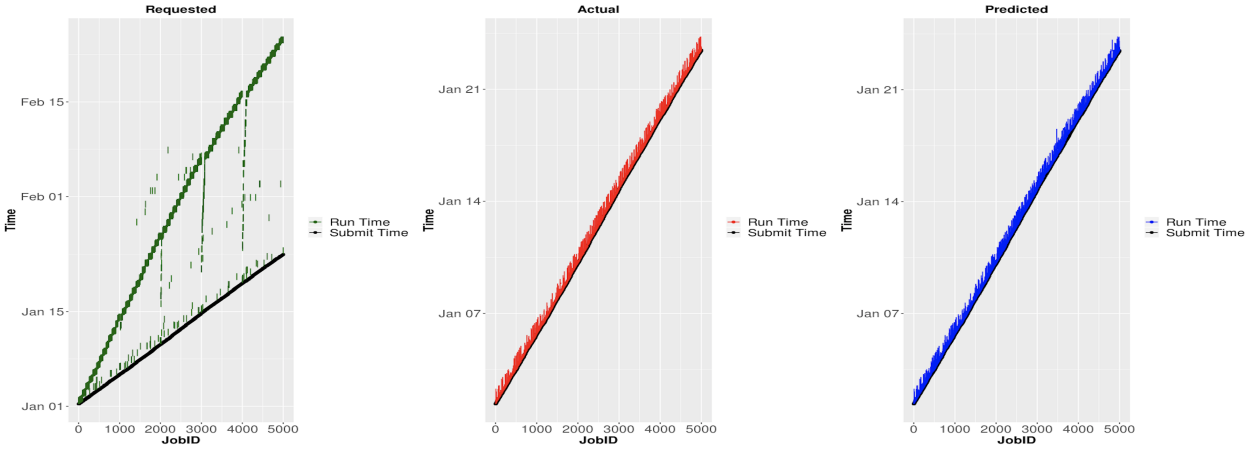


Fig. 4. Jobs Submission and Running time (Requested vs Actual vs Predicted) for RMACC Summit Jobs. Note dramatic improvement of Y axis range between graphs.

Table 2. Average Waiting and Turnaround Time (Requested vs Actual vs Predicted) For RMACC Summit

	Avg Wait Time (Hour)	Avg TA Time (Hour)	Median Wait Time (Hour)	Median TA Time (Hour)
Requested	380.6 ±241.2	403.14 ±243.3	401.2	425.7
Actual	1.3 ±0.7	2.9 ±3.2	0.5	1.1
Predicted	3.7 ±1.1	5.5 ±4.8	1.3	4.5

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 5** and **Figure 8** show that using our module to facilitate the RMACC Summit and Beocat HPC systems allows them to reach similar utilization **(up to 100%)** compared to the utilization of the HPC system that used actual job resources. **Figure 6** and **Figure 9** indicates that the backfill-sched algorithm has achieved more efficiency on the testbed that used our module compared to the ones that did not based on measuring the density of jobs attempts to schedule over the time. 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 2 and **Table 3** provides the calculated average waiting time with the median, average turn-around (TA) time, median Avg Wait Time (Hour), and median TA Time for the jobs in RMACC Summit and Beocat HPC resources for each requested, actual, and predicted runs. Using our model drastically reduced the average waiting time from **380 hours** to **4** hours and average turnaround time from **403 hours** to **6 hours** for RMACC Summit. And reduced the average waiting time from **662 hours** to **28 hours** and average turnaround time from **673 hours** to **35 hours** for Beocat.

5 CONCLUSIONS

Our machine learning model is valuable for both HPC users and administrators. Our model helps HPC users to estimate and recommend the amount of resources (Time and Memory) that required for their submitted jobs on HPC cluster. Our ML model is built based on implementation of different machine learning algorithms (Six discriminative models

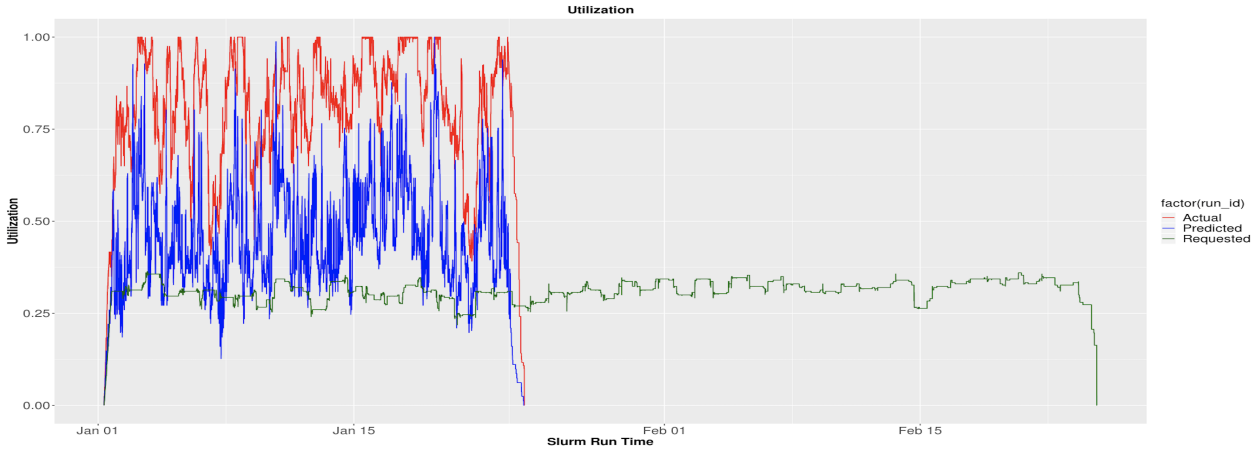


Fig. 5. Utilization (Requested vs Actual vs Predicted) for RMACC Summit Jobs.

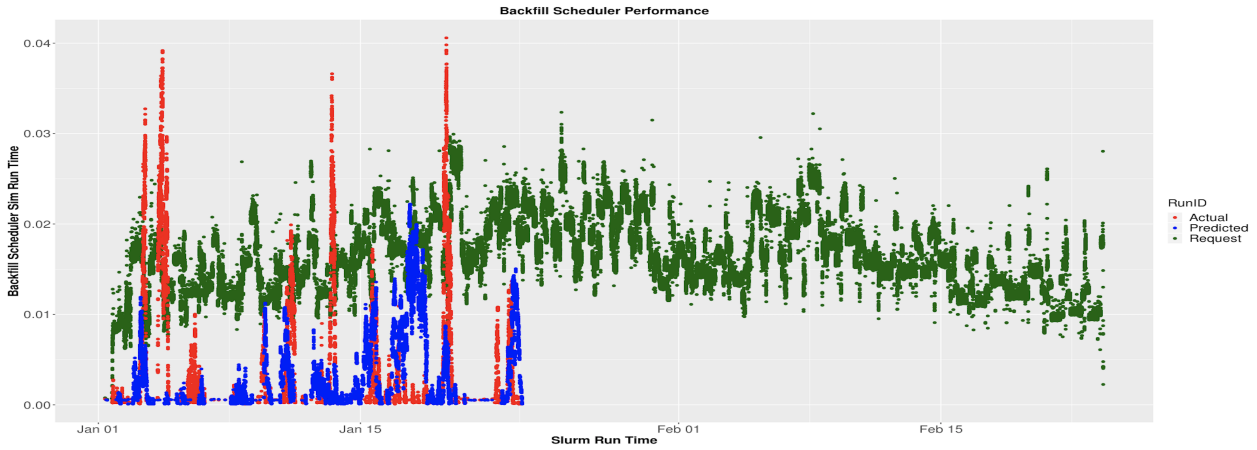


Fig. 6. Backfill-Sched Performance for RMACC Summit Jobs.

Table 3. 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	662.9 ±193.6	673.5±196.6	681.6	652.2
Actual	1.5 ±1.1	4.1 ±2.2	0.9	3.2
Predicted	27.7±25.3	34.8 ±27.1	6.2	13.9

from the scikit-learn and Microsoft LightGBM) applied on the historical data (sacct data) from Slurm for one XSEDE service provider (The University of Colorado Boulder (SUMMIT) and Kansas State University (Beocat)) HPC resources. We tested our ML model using one-hundred thousand job for each testbed.

Our results shows dramatically increased utilization of up to **100%**, decreased average waiting time **(from 380 to 4 hours in RMACC Summit and from 662 hours to 28 hours in Beocat)**, and decreased the average turn-around

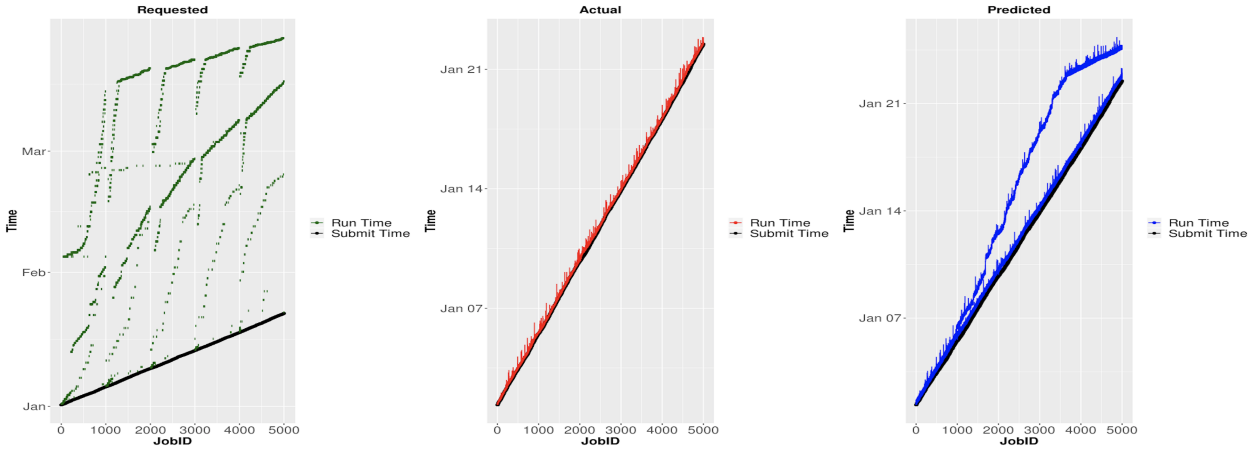


Fig. 7. Jobs Submission and Running time (Requested vs Actual vs Predicted) for Beocat Jobs. Note dramatic improvement of Y axis range between graphs.

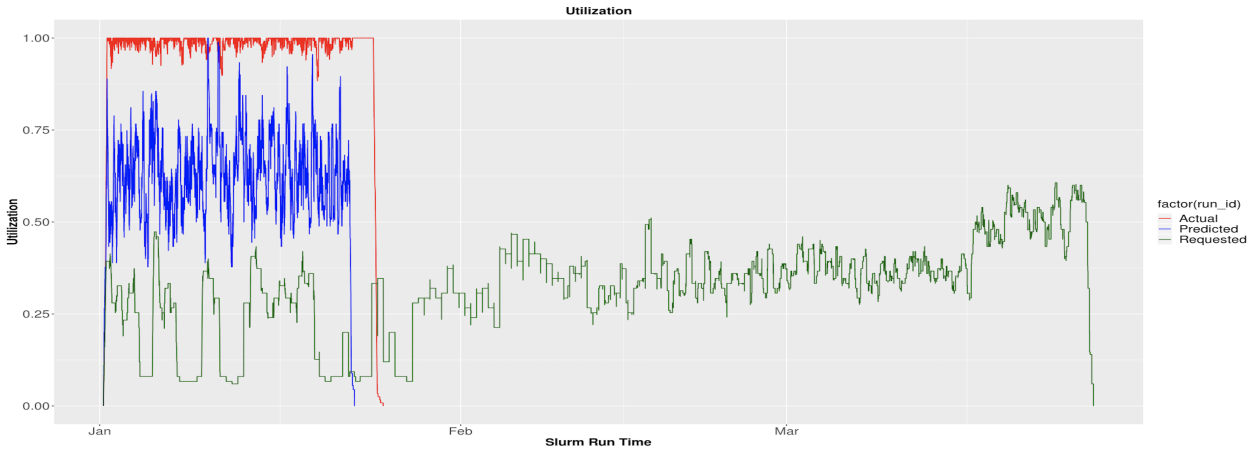


Fig. 8. Utilization (Requested vs Actual vs Predicted) for Beocat Jobs.

time for the submitted jobs **(from 403 to 6 hours in RMACC Summit and from 673 hours to 35 hours in Beocat)**. This implies a dramatic increase the efficiency and decreased power consumption for Slurm-based HPC resources.

6 FUTURE WORK

Our future work includes implementing an open-source software tool to predict and recommend the needed resources required for submitted jobs. In addition, our machine learning approach will incorporate many different machine learning algorithms, and build tools for both Slurm and SGE based resources so that can be beneficial for most of the HPC users and service providers, including XSEDE service providers.

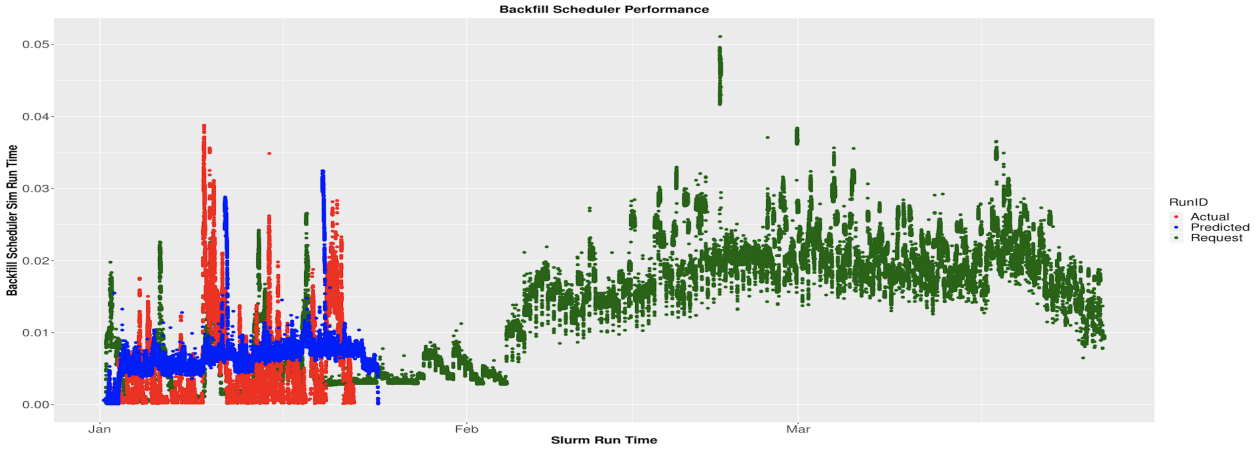


Fig. 9. Backfill-Sched Performance for Beocat Jobs.

ACKNOWLEDGMENTS

We are greatly appreciate the HPC staff at Kansas State University and University of Colorado Boulder, including Adam Tygart, Kyle Hutson, and Jonathon Anderson for their help and technical support. We also thank the authors of the Slurm simulator at SUNY U. of Buffalo for releasing their work. Special thanks to Brandon Dunn for his contribution. This research was supported by NSF awards CHE-1726332, ACI-1440548, CNS-1429316, NIH award P20GM113109, and Kansas State University.

This work used the Extreme Science and Engineering Discovery Environment (XSEDE), which is supported by National Science Foundation grant number ACI-1548562.

This work utilized resources from the University of Colorado Boulder Research Computing Group, which is supported by the National Science Foundation (awards ACI-1532235 and ACI-1532236), the University of Colorado Boulder, and Colorado 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] Omar Aaziz, Jonathan Cook, and Mohammed Tanash. 2018. Modeling Expected Application Runtime for Characterizing and Assessing Job Performance. In *2018 IEEE International Conference on Cluster Computing (CLUSTER)*. IEEE, 543–551.

[8] Jonathon Anderson, Patrick J Burns, Daniel Milroy, Peter Ruprecht, Thomas Hauser, and Howard Jay Siegel. 2017. Deploying RMACC Summit: an HPC resource for the Rocky Mountain region. In *Proceedings of the Practice and Experience in Advanced Research Computing 2017 on Sustainability, Success and Impact*. 1–7.

[9] Dan Andresen, William Hsu, Huichen Yang, and Adedolapo Okanlawon. 2018. Machine learning for predictive analytics of compute cluster jobs. *arXiv preprint arXiv:1806.01116* (2018).

[10] D Ardagna, E Barbierato, E Gianniti, M Gribaudo, TBM Pinto, APC da Silva, and JM Almeida. 2020. Predicting the performance of big data applications on the cloud. *The Journal of Supercomputing* (2020), 1–33.

[11] Giuseppe Bonaccorso. 2017. *Machine learning algorithms*. Packt Publishing Ltd.

[12] Leo Breiman. 2001. Random forests. *Machine learning* 45, 1 (2001), 5–32.

[13] 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

[14] Bradley Efron, Trevor Hastie, Iain Johnstone, Robert Tibshirani, et al. 2004. Least angle regression. *Annals of statistics* 32, 2 (2004), 407–499.

[15] Yuping Fan, Zhiling Lan, Taylor Childers, Paul Rich, William Allcock, and Michael E Papka. 2021. Deep Reinforcement Agent for Scheduling in HPC. *arXiv preprint arXiv:2102.06243* (2021).

[16] Dror G Feitelson, Dan Tsafrir, and David Krakov. 2014. Experience with using the parallel workloads archive. *J. Parallel and Distrib. Comput.* 74, 10 (2014), 2967–2982.

[17] 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

[18] Muhammad Hafizhuddin Hilman, Maria Alejandra Rodriguez, and Rajkumar Buyya. 2018. 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, 93–102.

[19] Guolin Ke, Qi Meng, Thomas Finley, Taifeng Wang, Wei Chen, Weidong Ma, Qiwei Ye, and Tie-Yan Liu. 2017. Lightgbm: A highly efficient gradient boosting decision tree. *Advances in neural information processing systems* 30 (2017), 3146–3154.

[20] Seounghyeon Kim, Young-Kyoon Suh, and Jeeyoung Kim. 2019. Extes: An execution-time estimation scheme for efficient computational science and engineering simulation via machine learning. *IEEE Access* 7 (2019), 98993–99002.

[21] Wei-Yin Loh. 2011. Classification and regression trees. *Wiley interdisciplinary reviews: data mining and knowledge discovery* 1, 1 (2011), 14–23.

[22] 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

[23] Chao Mei, Gengbin Zheng, Filippo Gioachin, and Laxmikant V Kalé. 2010. Optimizing a parallel runtime system for multicore clusters: a case study. In *Proceedings of the 2010 TeraGrid Conference*. 1–8.

[24] Farrukh Nadeem, Daniyal Alghazzawi, Abdulfattah Mashat, Khalid Faqeeh, and Abdullah Almalaise. 2019. Using machine learning ensemble methods to predict execution time of e-science workflows in heterogeneous distributed systems. *IEEE Access* 7 (2019), 25138–25149.

[25] Mina Naghshnejad and Mukesh Singhal. 2018. Adaptive online runtime prediction to improve HPC applications latency in cloud. In *2018 IEEE 11th International Conference on Cloud Computing (CLOUD)*. IEEE, 762–769.

[26] Bill Nitzberg, Jennifer M Schopf, and James Patton Jones. 2004. PBS Pro: Grid computing and scheduling attributes. In *Grid resource management*. Springer, 183–190.

[27] Adedolapo Okanlawon, Huichen Yang, Avishek Bose, William Hsu, Dan Andresen, and Mohammed Tanash. 2020. Feature Selection for Learning to Predict Outcomes of Compute Cluster Jobs with Application to Decision Support. *arXiv preprint arXiv:2012.07982* (2020).

[28] Ju-Won Park and Eunhye Kim. 2017. Runtime prediction of parallel applications with workload-aware clustering. *The Journal of Supercomputing* 73, 11 (2017), 4635–4651.

[29] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. 2011. Scikit-learn: Machine Learning in Python. *Journal of Machine Learning Research* 12 (2011), 2825–2830.

[30] Thanh-Phuong Pham, Juan J Durillo, and Thomas Fahringer. 2017. Predicting workflow task execution time in the cloud using a two-stage machine learning approach. *IEEE Transactions on Cloud Computing* 8, 1 (2017), 256–268.

[31] Michael Pinedo. 2012. *Scheduling*. Vol. 29. Springer.

[32] Théo Saillant, Jean-Christophe Weill, and Mathilde Mougeot. 2020. Predicting Job Power Consumption Based on RJMS Submission Data in HPC Systems. In *International Conference on High Performance Computing*. Springer, 63–82.

[33] 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.

[34] Garrick Staples. 2006. Torque resource manager. In *Proceedings of the 2006 ACM/IEEE conference on Supercomputing*. 8–es.

[35] Young-Kyoon Suh, Seounghyeon Kim, and Jeeyoung Kim. 2020. CLUTCH: A Clustering-Driven Runtime Estimation Scheme for Scientific Simulations. *IEEE Access* 8 (2020), 220710–220722.

[36] Taraneh Taghavi, Maria Lupetini, and Yaron Kretchmer. 2016. Compute job memory recommender system using machine learning. In *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. 609–616.

[37] Mohammed Tanash, Brandon Dunn, Daniel Andresen, William Hsu, Huichen Yang, and Adedolapo Okanlawon. 2019. 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)*. 1–8.

[38] Anastasia Tyryshkina, Nate Coraor, and Anton Nekrutenko. 2019. Predicting runtimes of bioinformatics tools based on historical data: five years of Galaxy usage. *Bioinformatics* 35, 18 (2019), 3453–3460.

[39] Jeffrey D. Ullman. 1975. NP-complete scheduling problems. *Journal of Computer and System sciences* 10, 3 (1975), 384–393.

[40] Qiqi Wang, Jing Li, Shuo Wang, and Guibao Wu. 2019. 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, 311–316.

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

[42] 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.

[43] Andy B Yoo, Morris A Jette, and Mark Grondona. 2003. Slurm: Simple linux utility for resource management. In *Workshop on job scheduling strategies for parallel processing*. Springer, 44–60.

[44] Di Zhang, Dong Dai, Youbiao He, Forrest Sheng Bao, and Bing Xie. 2020. RLScheduler: an automated HPC batch job scheduler using reinforcement learning. In *SC20: International Conference for High Performance Computing, Networking, Storage and Analysis*. IEEE, 1–15.