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

Mohammed Tanash,  
Kansas State University, Manhattan, Kansas  

Daniel Andresen,  
Kansas State University, Manhattan, Kansas  

Huichen Yang,  
Kansas State University, Manhattan, Kansas  

William Hsu  
Kansas State University, Manhattan, Kansas

Abstract

In this paper, we present a novel methodology for predicting job resources (memory and time) for submitted jobs on HPC systems. Our methodology 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 model 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 Machine learning model 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 up to 86% 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 achieved up to 100% utilization for both HPC resources.

Keywords
HPC; Scheduling; Machine Learning; Slurm; Performance

1 INTRODUCTION
High Performance Computing (HPC) users rely on running their extensive computations on HPC clusters. 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 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 [8], improving and expanding our previous work [28] [20] 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 methodology 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 method has been implemented for two HPC resources (An XSEDE service provider, University of Colorado-Boulder (RMACC SUMMIT), and Kansas State University (BEOCAT)). In this paper, we answered several research questions as the following: Can we enhance the performance of HPC resources by applying supervised machine learning algorithms using 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?

Improving the performance of Simple Linux Utility for Resource Management (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. Slurm is the most popular job scheduler over all of other schedulers such as SGE (Sun Grid Engine), TORQUE (Tera-scale Open-source Resource and Queue manager), PBS (Portable Batch System), and the Maui Cluster Scheduler. 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. The Slurm simulator is located in the Github.

2 RELATED WORK

Job scheduling in HPC systems affected by several factors such as job submission time, job duration time, number of requested processors, amount of requested time and memory, group ID, etc. 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. which don’t achieve maximum performance and utilization in multiple nodes clusters. Previous studies show that batch job scheduling is NP-complete problem.

There has been many studies focused in predicting the run time for running application on the HPC systems or on the cloud, 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). Fan et al. proposed another HPC scheduling techniques using deep reinforcement learning (DARS), which uses neural networks for resource reservation and backfilling.

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

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.

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 around 5% [6].

Our work combines both predicting memory and time required for submitted jobs on HPC systems. 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 stand alone ML model which dramatically improves the efficiency, and utilization. In addition, our model will decreases turnaround time, waiting time for the submitted jobs.

3 METHODOLOGY

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

3.1 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) [7]. 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 [19] 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 [22].

### 3.2 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) [10, 12], ii) Linear Regression (LR) [10], iii) Ridge Regression (RG) [10], iv) Elastic Net Regression (EN) [10], v) Classification and Regression Trees (DTR) [18], vi) Random Forest Regression (RFR) [11], and vii) LightGBM (LGBM) [16]. We used the Coefficient of determination ($R^2$) [10] and root mean squared error (RMSE) [10] to evaluate the regression models. We used scikit-learn’s [22] implementation for all models and performance metrics.

### 3.3 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**: Mixed Account Regression Model (MARM) Algorithm 1

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 $R^2_{tr} [i]$ for training data and in $R^2_{te} [i]$ for testing data. Finally, after all unique
accounts in $unacc$ have been evaluated, we select the best account id ($best_aid$) based on $R^2_{tp}, R^2_{te}$ 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.

**Algorithm 1**

\[
\text{MARM}(Y, X, acc, num\_acc, m)
\]

1. $unacc = \text{unique}(acc)$
2. $acc\_pool = \{\}$
3. Repeat $num\_acc$ times:
   
   for $i \in unacc$
   
   tac = append($acc\_pool, i$), if not $i \in acc\_pool$
   
   indices = which $tac \in acc$
   
   $X_a, Y_a = X[\text{indices}], Y[\text{indices}]$
   
   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.
   
   $R^2_{tp}[i] = \text{mean } R^2 \text{ of RM using training data.}$
   
   $R^2_{te}[i] = \text{mean } R^2 \text{ of RM using testing data.}$
   
   $best_aid = \text{Choose } i \text{ with best mean } R^2_{tp} \text{ and } R^2_{te} \text{ ranks.}$
   
   $best_r2\_tr = R^2_{tp}[best_aid]$
   
   $best_r2\_te = R^2_{te}[best_aid]$
   
   $acc\_pool = \text{append}(acc\_pool, best_aid)$
   
   Return $best_r2\_tr, best_r2\_te, acc\_pool$

4 RESULTS AND DISCUSSION

In this section we will explain the benchmarking predictive performance of regression models; the 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 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 1 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 2 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, 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 3 and Figure 6 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 that it takes around **fifty-five days** for
RMACC Summit and **eighty-two days** for Beocat to complete the execution for all of the
submitted jobs using user requested memory and time, while it takes less than **twenty-four
days** for both RMACC Summit and Beocat to complete the running of the submitted jobs
using the actual and predicted time and memory. Based on the results, our model predicted
the values for the required time and memory accurately. Figure 4 and Figure 7 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 5 indicates that the backfill-sched algorithm has achieved
more efficiency on the Beocat 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. While our
model achieved similar BEOCAT results for RMACC SUMMIT testbed. 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 from
the scikit-learn and Microsoft Light-GBM) applied on the historical data (sacct data) from
Slurm for one XSEDE service provider (The University of Colorado Boulder (RMACC
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 to 28 hours in Beocat**),
and decreased the average turn-around 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 a Slurm open-source software tool to predict and recommend the needed resources for HPC submitted jobs. Our tool can be beneficial for most of the HPC users and service providers, including XSEDE service providers. In addition, our machine learning approach will incorporate additional machine learning algorithms.

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.]. TORQUE Resource Manager. http://www.adaptivecomputing.com/products/torque/. (Accessed on 02/02/2019).
[6]. Aaziz Omar, Cook Jonathan, and Tanash Mohammed. 2018. Modeling Expected Application Runtime for Characterizing and Assessing Job Performance. In 2018 IEEE International Conference on Cluster Computing (CLUSTER). IEEE, 543–551.
[7]. Anderson Jonathon, Burns Patrick J, Milroy Daniel, Ruprecht Peter, Hauser Thomas, and Siegel Howard Jay. 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.
[8]. Andresen Dan, Hsu William, Yang Huichen, and Okanlawon Adedolapo. 2018. Machine learning for predictive analytics of compute cluster jobs. arXiv preprint arXiv:1806.01116 (2018).
[9]. Ardagna D, Barbierato E, Gianniti E, Gribaudo M, Pinto TBM, da Silva APC, and Almeida JM. 2020. Predicting the performance of big data applications on the cloud. The Journal of Supercomputing (2020), 1–33.
[10]. Bonaccorso Giuseppe. 2017. Machine learning algorithms. Packt Publishing Ltd.
[11]. Breiman Leo. 2001. Random forests. Machine learning 45, 1 (2001), 5–32.
[12]. Efron Bradley, Hastie Trevor, Johnstone Iain, Tibshirani Robert, et al. 2004. Least angle regression. Annals of statistics 32, 2 (2004), 407–499.
[13]. Fan Yuping, Lan Zhiling, Childers Taylor, Rich Paul, Allcock William, and Papka Michael E. 2021. Deep Reinforcement Agent for Scheduling in HPC. arXiv preprint arXiv:2102.06243 (2021).

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

[15]. Gentzsch W. [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. 10.1109/ccgrid.2001.923173

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

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

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

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

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

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

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

[23]. Pinedo Michael. 2012. Scheduling. Vol. 29. Springer.

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

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

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

[27]. Taghavi Taraneh, Lupetini Maria, and Kretchmer Yaron. 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.

[28]. Tanash Mohammed, Dunn Brandon, Andresen Daniel, Hsu William, Yang Huichen, and Okanlawon Adedolapo. 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.

[29]. Tsyryshkina Anastasia, Coraor Nate, and Nekrutenko Anton. 2019. Predicting runtimes of bioinformatics tools based on historical data: five years of Galaxy usage. Bioinformatics 35, 18 (2019), 3453–3460. [PubMed: 30698642]

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

[31]. Wang Qiqi, Li Jing, Wang Shuo, and Wu Guibao. 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.
[32]. Yoo Andy B., Jette Morris A., and Grondona Mark. 2003. Slurm: Simple linux utility for resource management. In Workshop on job scheduling strategies for parallel processing. Springer, 44–60.

[33]. Yoo Andy B., Jette Morris A., and Grondona Mark. 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.

[34]. Zhang Di, Dai Dong, He Youbiao, Bao Forrest Sheng, and Xie Bing. 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.
CCS CONCEPTS

- Computing methodologies → Supervised learning; Artificial intelligence; Software and its engineering → Scheduling; Hardware → Testing with distributed and parallel systems.
Figure 1:
R2, RMSE and Runtime of seven methods across 50 accounts in RMACC SUMMIT and 20 accounts in BEOCAT.
Figure 2:
$R^2$ versus Number of Accounts in predicting memory and time using MARM across BEOCAT and RMACC Summit
Figure 3:
Jobs Submission and Running time (Requested vs Actual vs Predicted) for RMACC Summit Jobs. Note dramatic improvement of Y axis range between graphs.
Figure 4:
Utilization (Requested vs Actual vs Predicted) for RMACC Summit Jobs.
Figure 5:
Backfill-Sched Performance for RMACC Summit Jobs.
Figure 6:
Jobs Submission and Running time (Requested vs Actual vs Predicted) for Beocat Jobs.
Note dramatic improvement of Y axis range between graphs.
Figure 7:
Utilization (Requested vs Actual vs Predicted) for Beocat Jobs.
Table 1:

| 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.                                      |
Table 2:

|                  | 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                  |
### 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                  |