Improving HPC system performance by predicting job resources for submitted jobs using machine learning techniques

by

Mohammed Tanash

B.Sc., Irbid National University, Jordan, 2005

M.Sc., University Utara Malaysia, Malaysia, 2008

M.Sc., New Mexico State University, USA, 2014

AN ABSTRACT OF A DISSERTATION

submitted in partial fulfillment of the
requirements for the degree

DOCTOR OF PHILOSOPHY

Department of Computer Science
Carl R. Ice College of Engineering

KANSAS STATE UNIVERSITY
Manhattan, Kansas

2021

Abstract

Overestimation of High Performance Computing (HPC) job resources allocation typically happens because of the wide variety of HPC applications, environment configuration options, and the lack of knowledge of the complex structure of HPC systems. This overestimation of resources will waste and devour HPC resources; hence, this will lead to inefficient cluster utilization, increased wait times, and increased turnaround time for submitted jobs.

With this background, this dissertation aims to investigate the benefits, effects, and challenges of using machine learning techniques for predicting job resources on HPC systems from different perspectives.

First, we have developed a machine learning model based on using several supervised ML discriminative models from the scikit-learn machine learning library applied on historical data from SunGrid Engine (SGE) provided by an HPC service provider at the Kansas State University called Beocat. Our methodology achieved high accuracy in predicting the amount of required time and the amount of required memory for Beocat HPC resources.

Second, we have designed a machine learning methodology called Mixed Account Regression Model (MARM) built based on several supervised machines learning discriminative models from the scikit-learn machine learning library and LightGBM. Our work has been implemented and tested using historical data (sacct data) provided from two HPC providers, an XSEDE service provider at the University of Colorado-Boulder (RMACC Summit) and the Kansas State University (Beocat). Our models help dramatically reduce computational average waiting time, reduce turnaround time. Moreover, our models help achieve higher utilization, throughput, and efficiency for HPC resources.

Third, we introduced our first-ever implemented, fully-offline, fully-automated, standalone, and open-source Machine Learning tool called AMPRO-HPCC. Our tool aims to help

HPC admins and HPC users predict memory and time requirements for their submitted jobs on HPC Clusters.

Finally, we study and investigate the impact of our machine learning models in running jobs on the cloud by comparing the cost of running the jobs with and without using our machine learning models on most popular cloud computing resources, including Amazon Web Services such as (AWS), Microsoft Azure, Google Cloud Platform, Digital Ocean, IBM Cloud, and using the local resources of Holland Computing Center at the University of Nebraska - Lincoln.

In summary, in this work, we present and develop novel methodologies for predicting job resources (memory and time) for submitted jobs on HPC systems based on historical jobs data provided by the HPC systems scheduler. The outcomes are expected to dramatically reduce computational average waiting time, reduce turnaround time for submitted jobs. Moreover, increased utilization, increased throughput, improved efficiency, and decreased power consumption for the HPC systems.

Improving HPC system performance by predicting job resources for submitted jobs using machine learning techniques

by

Mohammed Tanash

B.Sc., Irbid National University, Jordan, 2005

M.Sc., University Utara Malaysia, Malaysia, 2008

M.Sc., New Mexico State University, USA, 2014

A DISSERTATION

submitted in partial fulfillment of the
requirements for the degree

DOCTOR OF PHILOSOPHY

Department of Computer Science
Carl R. Ice College of Engineering

KANSAS STATE UNIVERSITY
Manhattan, Kansas

2021

Approved by:

Major Professor
Daniel Andresen

Copyright

Abstract

Overestimation of High Performance Computing (HPC) job resources allocation typically happens because of the wide variety of HPC applications, environment configuration options, and the lack of knowledge of the complex structure of HPC systems. This overestimation of resources will waste and devour HPC resources; hence, this will lead to inefficient cluster utilization, increased wait times, and increased turnaround time for submitted jobs.

With this background, this dissertation aims to investigate the benefits, effects, and challenges of using machine learning techniques for predicting job resources on HPC systems from different perspectives.

First, we have developed a machine learning model based on using several supervised ML discriminative models from the scikit-learn machine learning library applied on historical data from SunGrid Engine (SGE) provided by an HPC service provider at the Kansas State University called Beocat. Our methodology achieved high accuracy in predicting the amount of required time and the amount of required memory for Beocat HPC resources.

Second, we have designed a machine learning methodology called Mixed Account Regression Model (MARM) built based on several supervised machines learning discriminative models from the scikit-learn machine learning library and LightGBM. Our work has been implemented and tested using historical data (sacct data) provided from two HPC providers, an XSEDE service provider at the University of Colorado-Boulder (RMACC Summit) and the Kansas State University (Beocat). Our models help dramatically reduce computational average waiting time, reduce turnaround time. Moreover, our models help achieve higher utilization, throughput, and efficiency for HPC resources.

Third, we introduced our first-ever implemented, fully-offline, fully-automated, standalone, and open-source Machine Learning tool called AMPRO-HPCC. Our tool aims to help

HPC admins and HPC users predict memory and time requirements for their submitted jobs on HPC Clusters.

Finally, we study and investigate the impact of our machine learning models in running jobs on the cloud by comparing the cost of running the jobs with and without using our machine learning models on most popular cloud computing resources, including Amazon Web Services such as (AWS), Microsoft Azure, Google Cloud Platform, Digital Ocean, IBM Cloud, and using the local resources of Holland Computing Center at the University of Nebraska - Lincoln.

In summary, in this work, we present and develop novel methodologies for predicting job resources (memory and time) for submitted jobs on HPC systems based on historical jobs data provided by the HPC systems scheduler. The outcomes are expected to dramatically reduce computational average waiting time, reduce turnaround time for submitted jobs. Moreover, increased utilization, increased throughput, improved efficiency, and decreased power consumption for the HPC systems.

Table of Contents

List of Figures

List of Tables

Acknowledgments

Praise be to Allah for the strengths and blessings in completing this dissertation. The success and outcome of this dissertation required a lot of guidance and assistance, I am extremely fortunate to have gotten this all along with the completion of my thesis work. Whatever I have done is only due to such guidance and assistance.

I would like to express my deep gratitude and very great appreciation to Prof. Daniel Andresen, for providing me with his high support, unlimited guidance, and patience. I could not have imagined having a better advisor and I could not have done this dissertation without his excellent help, assistance, and advice. Thank you so much from the bottom of my heart.

I would like to express my special appreciation and thanks to Prof. William Hsu for his assistance, guidance, support, scientific advice, and valuable input.

Prof. Daniel Andresen and Prof. William Hsu are two of the smartest people I know, and they are the best advisors to work with.

I would like to thank and acknowledge the committee members for their helpful suggestions and advice.

A special thanks to the graduate advisor of the Computer Science at Kansas State University Prof. Mitchell Neilsen for his help, support, and guidance to all Computer Science graduate students.

I would like to thank the people in the Computer Science department office, especially Sheryl Cornell and Wynne Reichart for their help.

I greatly appreciate the HPC staff at Kansas State University and Holloand Computing Center at University of Nebraska-Lincoln including Adam Tygart, Kyle Hutson, and Caughlin Bohn for their help and technical support. Having a good support system is important to surviving in graduate school journey.

I also thank the authors of the Slurm simulator at SUNY U. of Buffalo for releasing their work.

Last but not the least, I place a deep sense of thankfulness and appreciation to my parents, my brothers, my sisters, and my friends for all of the sacrifices that you have made on my behalf. Your prayer for me was what sustained me thus far.

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 the 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.

Dedication

This thesis is wholeheartedly dedicated to my role models Mom Randa Tanash, and to my Dad Awad Tanash. This work would not be complete without your pray, love, support, and patience.

Chapter 1

Introduction

1.1 Introduction and Background

High Performance Computing (HPC) systems are referred to as supercomputers, and are comprised of the aggregation of many computer resources. HPC systems are also commonly referred to as computing clusters. A cluster consists of a group of compute nodes. Each node in a cluster has an operating system, processors with multiple cores, storage unit, and networking capabilities to communicate with other nodes in the cluster[1]. HPC has been used extensively in many fields of science including, astrophysics, bioinformatics, artificial intelligence, financial, weather and climate, big data analytics, molecular dynamics, cyber-security, and much more. Moreover, HPC drives research, development, and innovation in many industries, from rendering the visual effects of the latest blockbuster movie, to sequencing the human genome to cure diseases, to risk analysis in financial services, or even to design the next generation of cars[2][3]. Hence, HPC is typically used to solve complex problems with large datasets (gigabytes, terabytes, petabytes, and zettabytes) by leveraging distributed compute resources to produce results in a relatively short amount of time. Therefore, the primary purpose of HPC is to solve complex problems which are either too large or would take too long for a laptop or a desktop to handle.

One of the most important parts of HPC systems is a scheduler, which is a software package that decides when and where to run jobs in the cluster. When an HPC user submits a job to the scheduler, they specify the resource requirements they need, such as the number of nodes, the number of processors, the amount of time, and the amount of memory. When an HPC user submits a job, the job scheduler assigns which resources the job is allowed to run on, ensuring that jobs do not overlap. At some point, the cluster will reach a point where there are no more resources for more jobs to run. At this stage, the job scheduler will hold jobs in a wait queue. As the running jobs complete, making more resources available, the job scheduler then allows queued jobs to run to best ensure the compute resources are being fully utilized. Another task of the job scheduler is to assign a priority to each job waiting in the queue. The highest priority job sits at the top of the queue, waiting for computing resources to become available. Every short period of time, the job queue is re-evaluated, and adjustments are made to the queue based on waiting jobs and the respective priority levels. The job scheduler will frequently re-arrange the order of jobs in the queue based on the priority. For the job scheduler to calculate a job's priority, it uses a formula based on various factors, including jobs already running on the cluster, number of resources required, fairness, and usage history. Another task of the job scheduler is to look at resources limitations set by the system administrators. Common limits include the maximum number of jobs a single user can run on the cluster, the maximum number of processors a single user or a group can request, or the maximum number of processors a single job can consume on the HPC system. If limits are exceeded, the job scheduler will place waiting jobs in the blocked queue. Jobs in the blocked queue will not run regardless of whether system resources are available as long as they remain in the blocked queue. As jobs complete, a user with blocked jobs will drop below the threshold limits, and their jobs will move up in the eligible wait queue.

Usually, in the HPC system, there are more jobs that can run at any one time on the compute nodes. Therefore, the scheduler needs to delay some submitted jobs to allow

another set of jobs to run. Every HPC job has an environment that the scheduler will consider for scheduling and then distributes the jobs to the cluster. This environment includes the run time limit, core requirements, memory limits, and the running command. Another big responsibility of the scheduler is to allocate required resources on the cluster and other resources such as licenses or I/O bandwidth.

One of the common challenges with HPC is processing and producing more data and generating more information and results than the infrastructure of HPC resources can handle. Hence, the HPC jobs have to wait for weeks or months in the queue in order to get start computations and produce results, which typically delays innovation and slows down research. On the other hand, it is the HPC users' responsibility to keep their jobs as efficient as possible to ensure less waiting time on the queue and keep the efficiency of the HPC system at the highest possible level. Many HPC users do not know the amount of resources their jobs need for calculation, so they are often encouraged to overestimate the amount of resources required for their submitted jobs, so their jobs will not be killed during the running time due to the insufficient amount of the resources. This overestimation of resources will negatively affect the performance, efficiency, throughput, and power consumption of the HPC system by consuming more resources than needed for a job's computations.

1.2 Research questions and contributions

In this dissertation, we study the potential benefits of using Machine learning (ML) methods in order to help HPC users to predict the resources needed (memory and time) for their submitted jobs on the cluster and the cloud.

This dissertation explores and investigate the following research questions:

- How to improve the performance of HPC systems via applying machine learning algorithms on historical jobs log data provided from the scheduler? Improving performance means decreasing average job turnaround time, decreasing average job waiting time,

increasing HPC system utilization, increasing the performance of the HPC scheduling and back-filling, and decreasing the power consumption of the HPC systems.

- How accurate is our proposed model in terms of prediction of the resource needed for submitted jobs?
- Which is more significant to predict in terms of job resources? Required memory or required time for submitted jobs on the HPC systems?
- Can we enhance the performance of HPC systems by applying supervised machine learning algorithms using Slurm-based HPC historical data?
- 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 determining the amount of the required resources (memory and time) accurate and easy for the submitted jobs.
- How much resources (memory and time) in average do HPC users overestimate for their submitted jobs?
- How efficient and effective is using our machine learning model for predicting job resources (memory and time) in terms of average cost and resources reduction for running jobs on the cloud, especially the economic impact of predicting job resources?

The major contributions referenced in this dissertation are summarized as follows:

- M. Tanash, B. Dunn, D. Andresen, Y. Huichen, A. Okanlawon, W. Hsu, "Improving HPC System Performance by Predicting Job Resources via Supervised Machine Learning," in the Proceedings of the 2019 Conference on Practice and Experience in Advanced Research Computing (PEARC19), Chicago, IL, July 28-August 1, 2019.
- M. Tanash, H. Yang, D. Andresen, W. Hsu, "Ensemble Prediction of Job Resources to Improve System Performance for Slurm-Based HPC Systems," selected as *Best Full*

Paper for the Systems and Systems Software Track and awarded the *Phil Andrews Award* for paper likely to be most impactful on the practice of research computing, in the Proceedings of the 2021 ACM Conference on Practice and Experience in Advanced Research Computing (PEARC21), pp. 1-8, Portland, OR, July 19–22, 2021. https://doi.org/10.1145/3437359.3465574

- M. Tanash, D. Andresen, W. Hsu, "AMPRO-HPCC: A Machine-Learning-Tool for Predicting Resources On Slurm HPC Clusters," to appear in the Proceedings of The Fifteenth International Conference on Advanced Engineering Computing and Applications in Sciences (ADVCOMP 2021), Barcelona, Spain, October 3-7, 2021.

- M. Tanash, R. Knepper, D. Andresen, "Improving XSEDE Systems Performance by Predicting Job Resources via Supervised Machine Learning," in the poster session of the Rocky Mountain Advanced Computing Consortium HPC Symposium (RMACC 20), Boulder, CO, May 20-21, 2020.

- M. Tanash, B. Dunn, D. Andresen, Y. Huichen, A. Okanlawon, W. Hsu, "Improving HPC System Performance by Predicting Job Resources via Supervised Machine Learning," in poster session of the Rocky Mountain Advanced Computing Consortium HPC Symposium (RMACC 19), 2nd place winner in best poster competition, Boulder, CO, May 21-23, 2019.

1.3 Dissertation structure

The dissertation is organized as follows. Chapter 1 introduces the introduction, background, problem statement, research questions, and contributions of this dissertation. Chapter 2 demonstrates the impact and benefits of using machine learning from the scikit-learn machine learning library for predicting the job resources needed (memory and time) for submitted jobs on the HPC systems applied over historical data from SunGrid Engine (SGE).

Chapter 3 introduces our Mixed Account Regression Model (MARM) designed for better accuracy of predicting job resources (memory and time) for submitted jobs on the HPC systems using the historical data provided from the Slurm workload manager. Chapter 4 introduces and discusses the implementation of our first-ever implemented, fully-offline, fully-automated, stand-alone, and open-source Machine Learning tool for predicting resources on Slurm HPC clusters (AMPRO-HPCC). Our accurate predicting tool has been tested over millions of jobs provided from the slurm scheduler of the HPC cluster of the Kansas State University called Beocat. In Chapter 5 we study and investigate the impact and effectiveness of using our supervised machine learning methodology for resource provisioning of cloud computing, especially the cost and resource provisioning using most popular cloud computing resources such as Amazon Web Service (AWS), Microsoft Azure, Google Cloud Platform, Digital Ocean, IBMCloud, and using the local resources of Holland Computing Center at the University of Nebraska - Lincoln. Finally, Chapter 6 presents the key conclusions, reflects on the limitations, and intimates directions for future research work.

Chapter 2

Improving HPC System Performance by Predicting Job Resources via Supervised Machine Learning [1]

2.1 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 to insufficient resources. This process will waste and devour HPC resources; hence, this 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 involves using different machine learning algo-

[1] This chapter is a slightly modified version of our published article [4]

rithms. Our goal is to integrate and test the proposed supervised machine learning model on 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 required jobs memory resources and the time required for each particular job 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.

2.2 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 that decides and controls what calculations to run next and wherein the HPC systems[5]. 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, geo-information science, and evolutionary biology rely on and demand HPC resources through simulations, experiments, and dealing with a tremendous amount of data[6 7]. These users are usually not familiar and do not have the 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.

2.2.1 Slurm Workload Manager

There are different varieties of job schedulers such as Sun Grid Engine (SGE)[8], Maui Cluster Scheduler[9], Tera-scale Open-source Resource and Queue manager (TORQUE)[10], and Portable Batch System (PBS)[11]. Simple Linux Utility for Resource Management (Slurm) which is one of the most popular among all of them[5]. Slurm is open-source; fault-tolerant; secure; highly configurable; highly scalable, and supports most Unix variants. Slurm's role is both workload manager and a job scheduler, which makes Slurm more convenient to use. The Resource manager role is allocating resources such as nodes, sockets, cores, hyperthreads, 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 fairshare scheduling, preemption, gang scheduling, advanced reservation, etc.[12].

2.2.2 Slurm Simulator

In order to test our module, we implemented a machine learning module and testing it using the *Slurm simulator* developed by the Center for Computational Research, SUNY University at Buffalo. The Slurm simulator is located in Github[13]. 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[14]. 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 a 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 the needed amount of memory resources and required running time for each particular submitted job (regression). We hope to improve the efficiency of Slurm and the HPC systems themselves 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 3.4. In Section 3.5 we present the results of our experiments and conclude in Section 3.6.

2.3 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 are to predict and determine the resources needed to accomplish the jobs submitted on the cluster and determine which is more harmful to the HPC system, overestimate the memory or the time for the jobs running

on the cluster?

Matsunaga and Fortes[15] 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[16] 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[17] developed a prediction system called Predicting Quick Starters (PQStar) for identified and predicting quick starters jobs (jobs that have waiting time ¡ 1 hour). PQStar prediction based on jobs request size and estimated run-time time, queue, and processor occupancy states.

García[18] 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[19].

Other works focused on predict and maximize power consumption for scientific applications and maximize performance using machine learning techniques[20][21] .

2.4 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.

2.4.1 Workflow Model

The workflow model of our work is described in **Figure 4.2** 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.

2.4.2 Data Preparation and Feature Analysis

For training our machine learning model, we used **fourteen million** instances that 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 2.1** 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 significant cluster in the state of Kansas. It is located at Kansas State University and operated by the Computer Science department[22].

2.4.3 Machine Learning Algorithms

Several discriminative models from the **scikit-learn** machine learning library[23][24] were trained to implement predictive functionality in our experiments. Data preparation steps

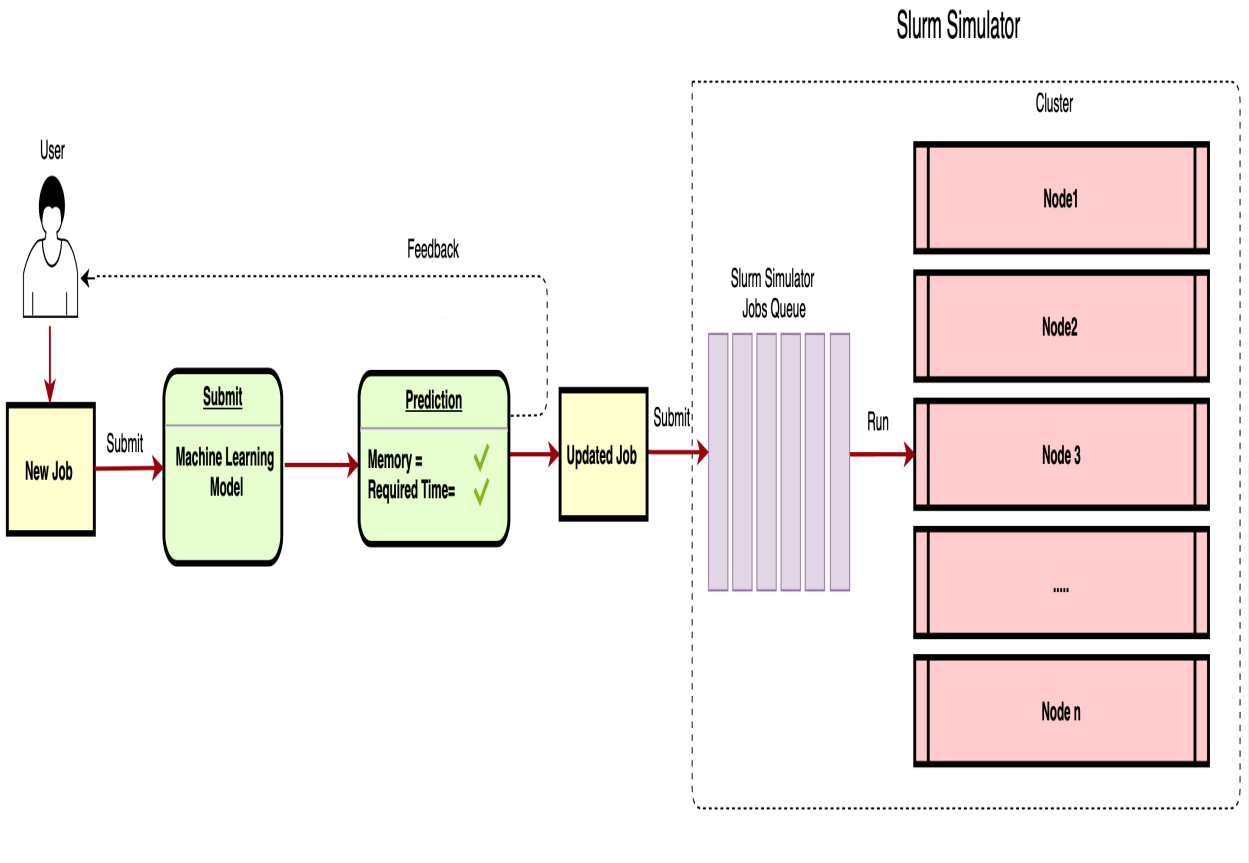

Figure 2.1: *Work Flow Diagram for our Model*

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[25] 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

Feature	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 2.1: *Feature Selected*

ratio), and a decision tree regressor. For the linear discriminants and their use on this task, we refer the interested reader to[26]. Using these flexible representations admits a balance of generalization quality (via overfitting control) and explainability.

2.5 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.

2.5.1 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 are shown in **Table 2.2** and **Table 2.3** respectively. Based on our results we chose **DecisionTreeRegressor** algorithm in our model since it has the most significant R-squared value which

Model	R^2 (%)	Time (Second)
LR	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 2.2: *Wall Clock Time Limit 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 2.3: *Memory Required Prediction Algorithms Results*

means the most fitted data to the regression line.

The legend for **Table 2.2** and **table 2.3** described as follows:

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

2.5.2 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). Job submission time is the time stamp that represents when the job was submitted, while the execution time is calculated as the difference between the start execution time and end execution time. **System Utilization** measures 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 run time.

Testbed-1

Testbed-1 contains larger jobs (jobs that are requesting at least 4GB of memory and four cores per task). *Testbed-1* includes a set of a thousand jobs. **Figure 2.2** shows the 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.

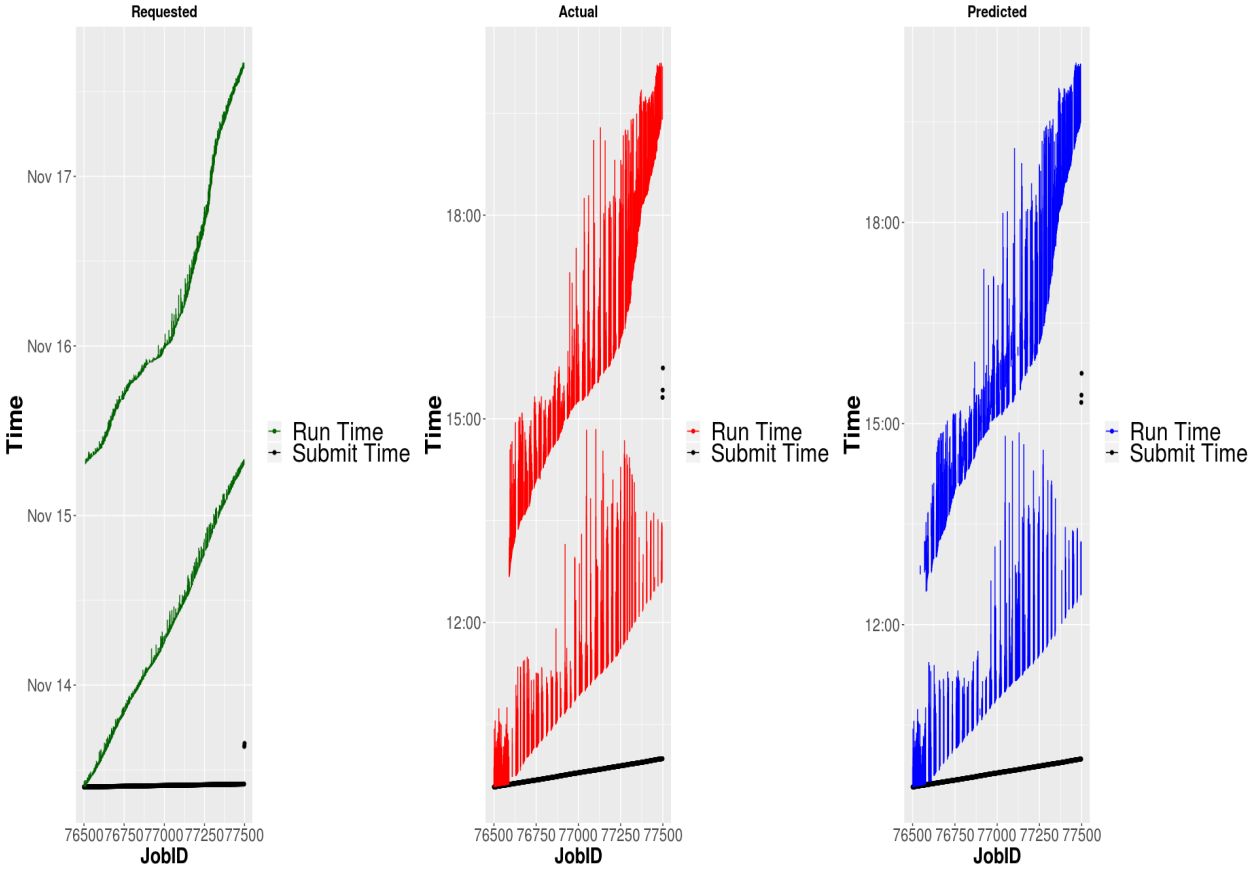

Figure 2.2: *Jobs Submission and Running time (Requested vs Actual vs Predicted) for Jobs in Testbed-1. Note dramatic improvement of Y axis range between graphs*

Figure 2.3 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 2.4** 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 2.4 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 the average turnaround time from **46.29** hours to **4.94**. Both predicted average waiting time and turn-around time are almost exactly the same as the actual average waiting time and turnaround time for jobs in *testbed-1*.

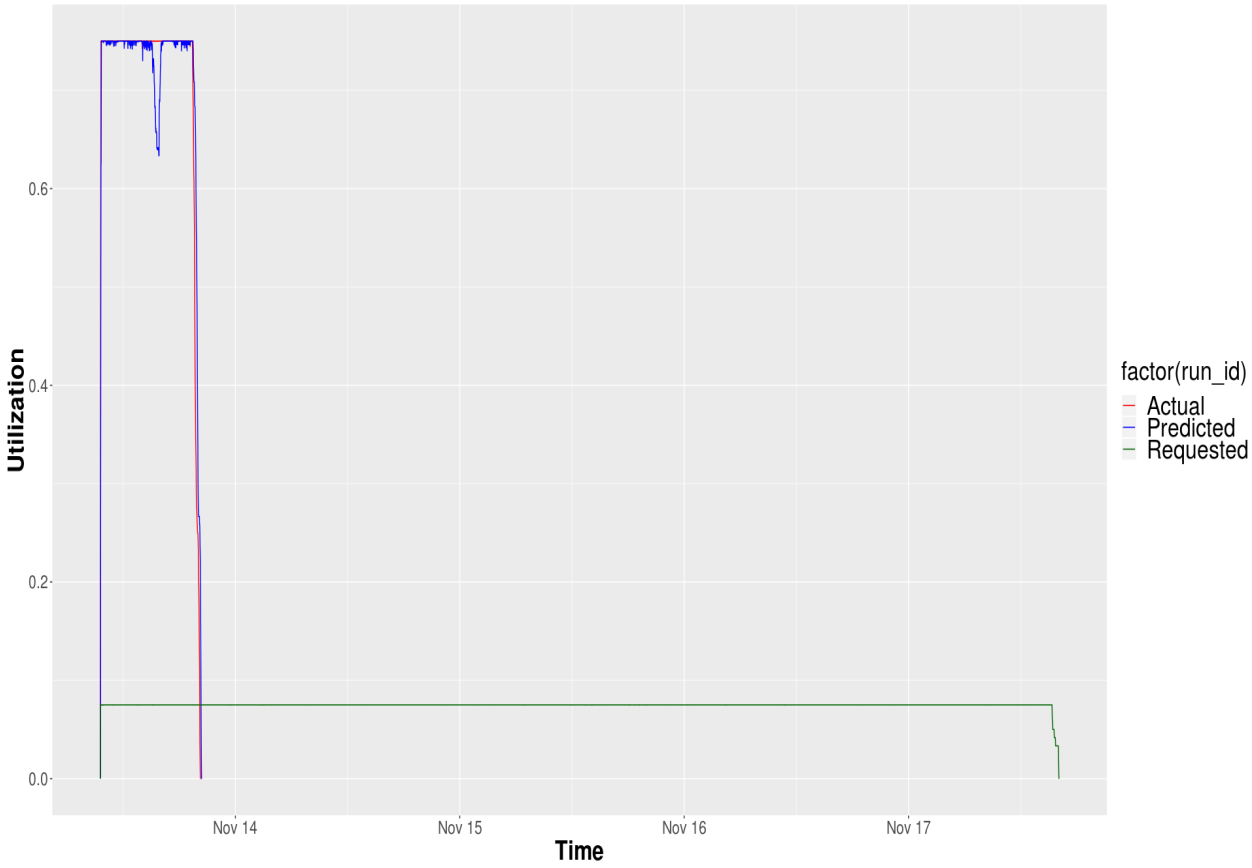

Figure 2.3: *Utilization (Requested vs Actual vs Predicted) for Jobs in Testbed-1*

Testbed-2

Testbed-2 contains smaller jobs (jobs that 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, **Figures 2.5 and 2.6** shows that our predicted model achieved better utilization and better backfilling performance. Moreover, **Table 2.5** shows that our predicted model incrementally reduced the average

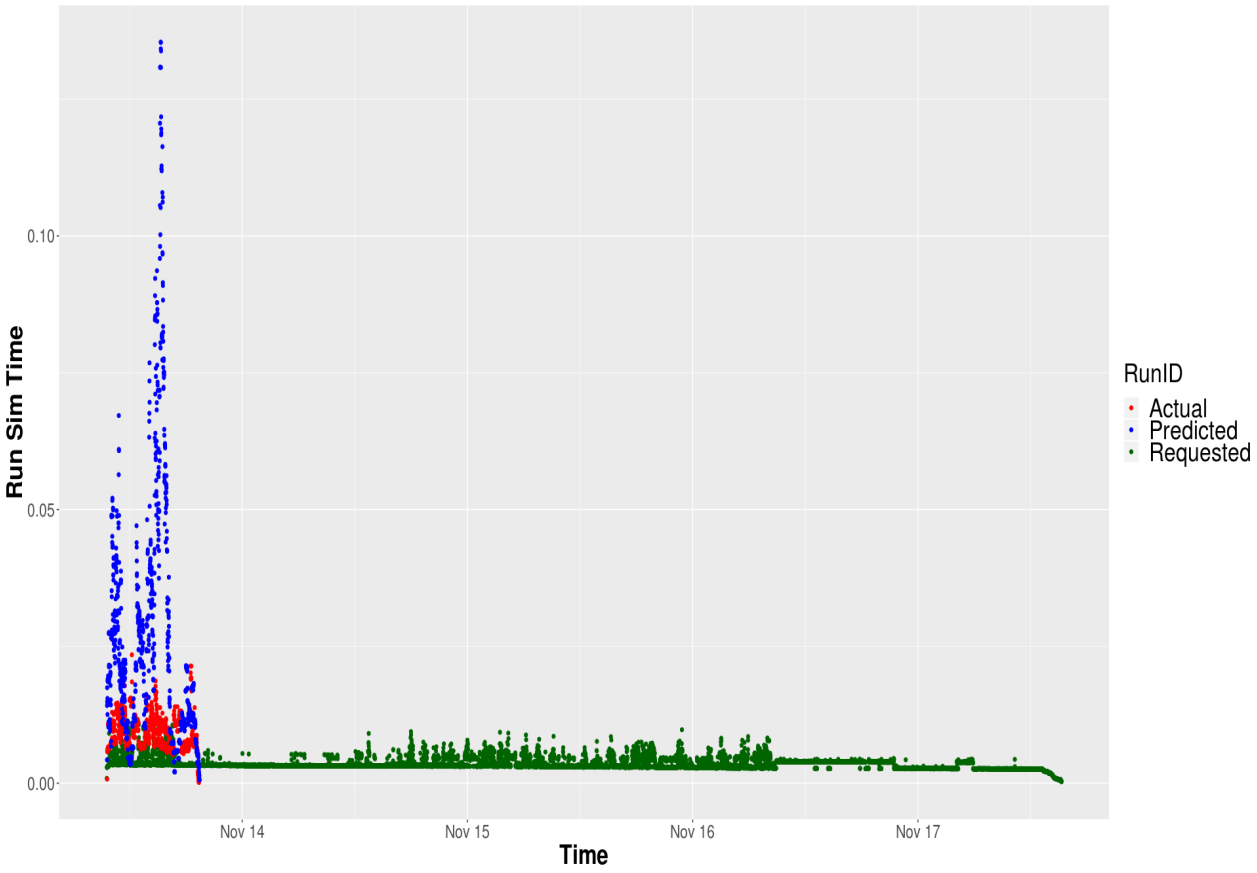

Figure 2.4: *Backfill-Sched Performance for Jobs in Testbed-1*

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

Table 2.4: *Average Waiting and Turnaround Time (Requested vs Actual vs Predicted) For Jobs in Testbed-1*

waiting and turnaround time from **(0.08 to 0.06 hours)** and from **(3.90 to 3.54 hours)** respectively.

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

Table 2.5: *Average Waiting and Turnaround Time (Requested vs Actual vs Predicted) For Jobs in Testbed-2*

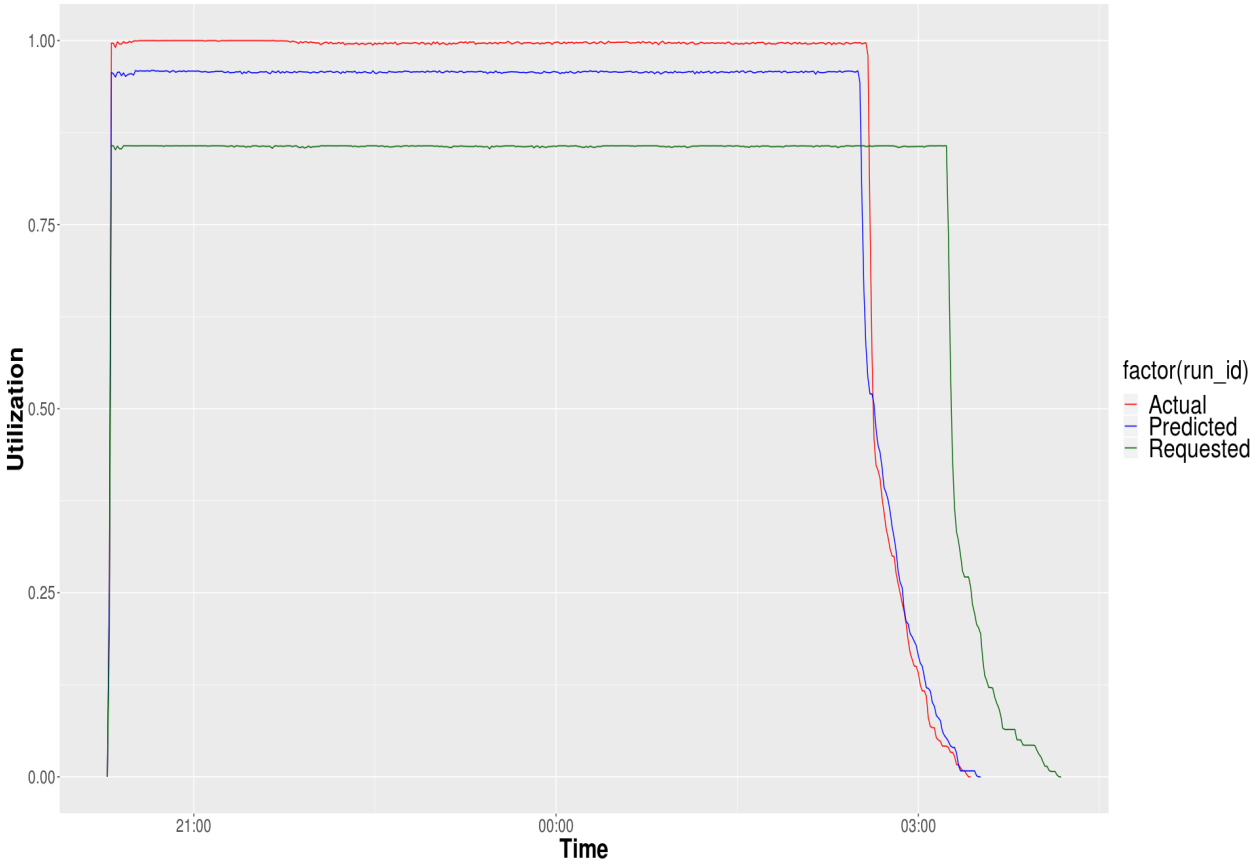

Figure 2.5: *WUtilization (Requested vs Actual vs Predicted) For Testbed-2*

2.5.3 Predicting Memory Required vs. Predicting Time Required

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

Figure 2.7 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

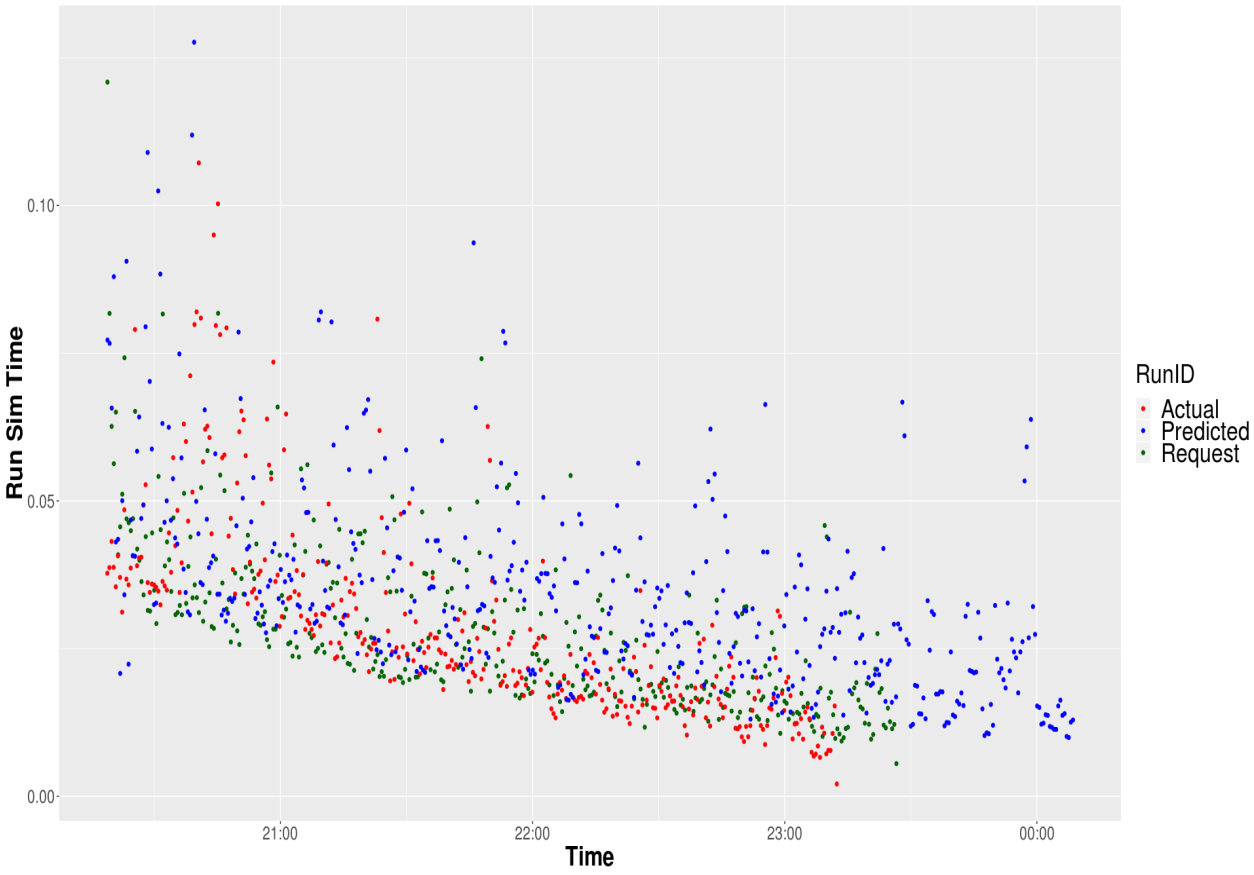

Figure 2.6: *Backfill-Sched Performance for Testbed-2*

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.

Figures 2.8 and 2.9 show the comparison of the utilization and the performance of the 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 indicate that both memory prediction and time requested prediction are highly valuable and are almost equally important because they achieved similar performance as shown in the graphs. We achieved peak performance and utilization by combining both of them in one model.

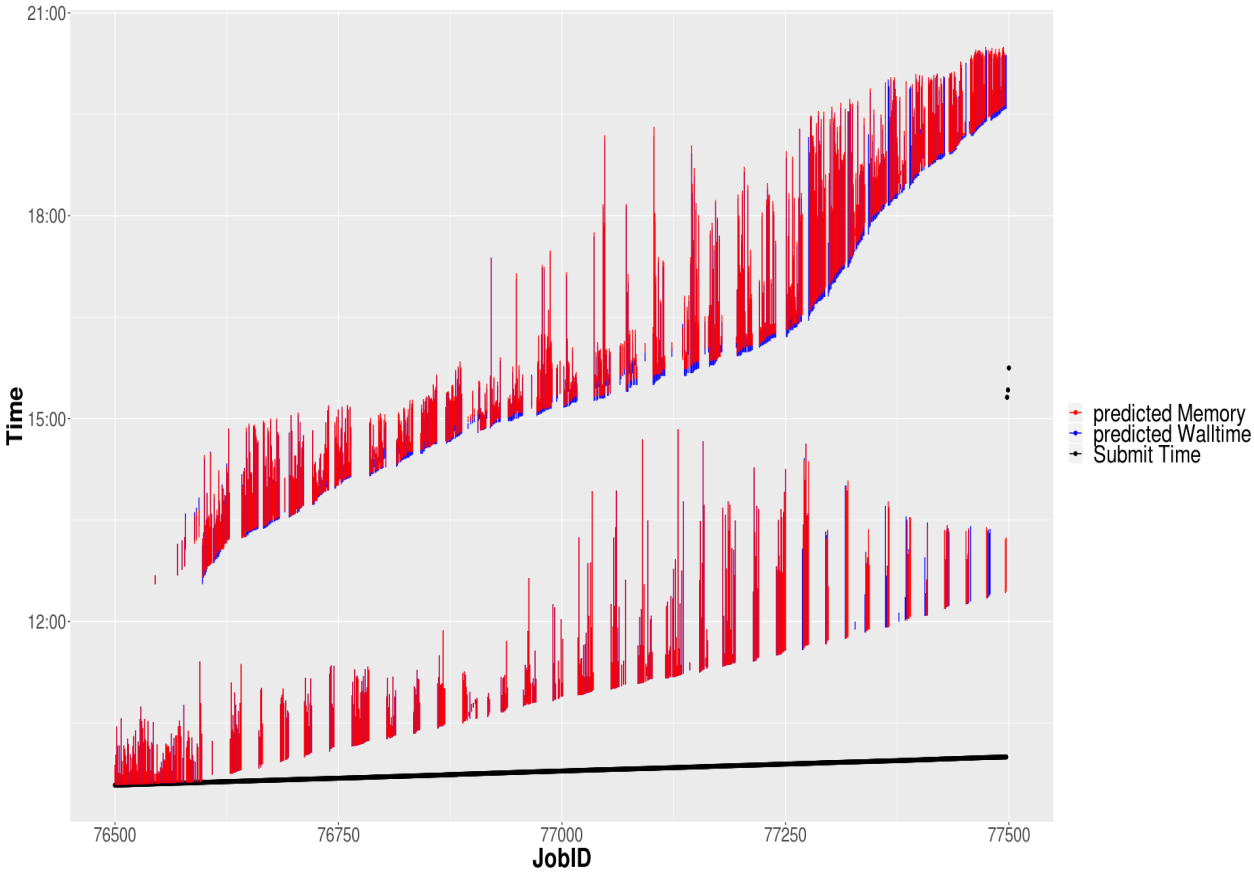

Figure 2.7: *Jobs Submission and Running time (Predicted Time Required vs Memory*

2.6 Summary

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 capability, and decrease the power consumption of the cluster.

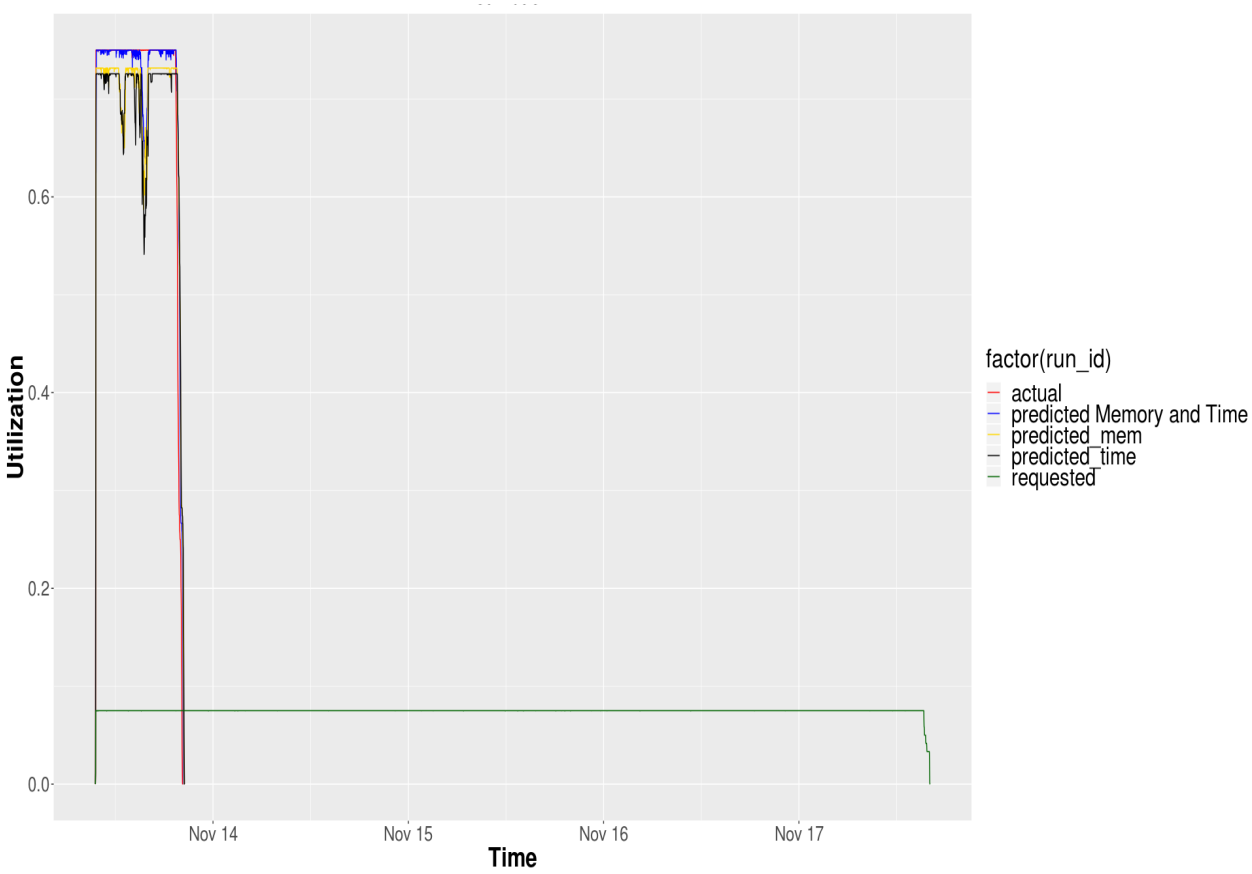

Figure 2.8: *Utilization (Requested vs Actual vs Required Time Predicted vs Memory Predicted vs Required Time and Memory Predicted)*

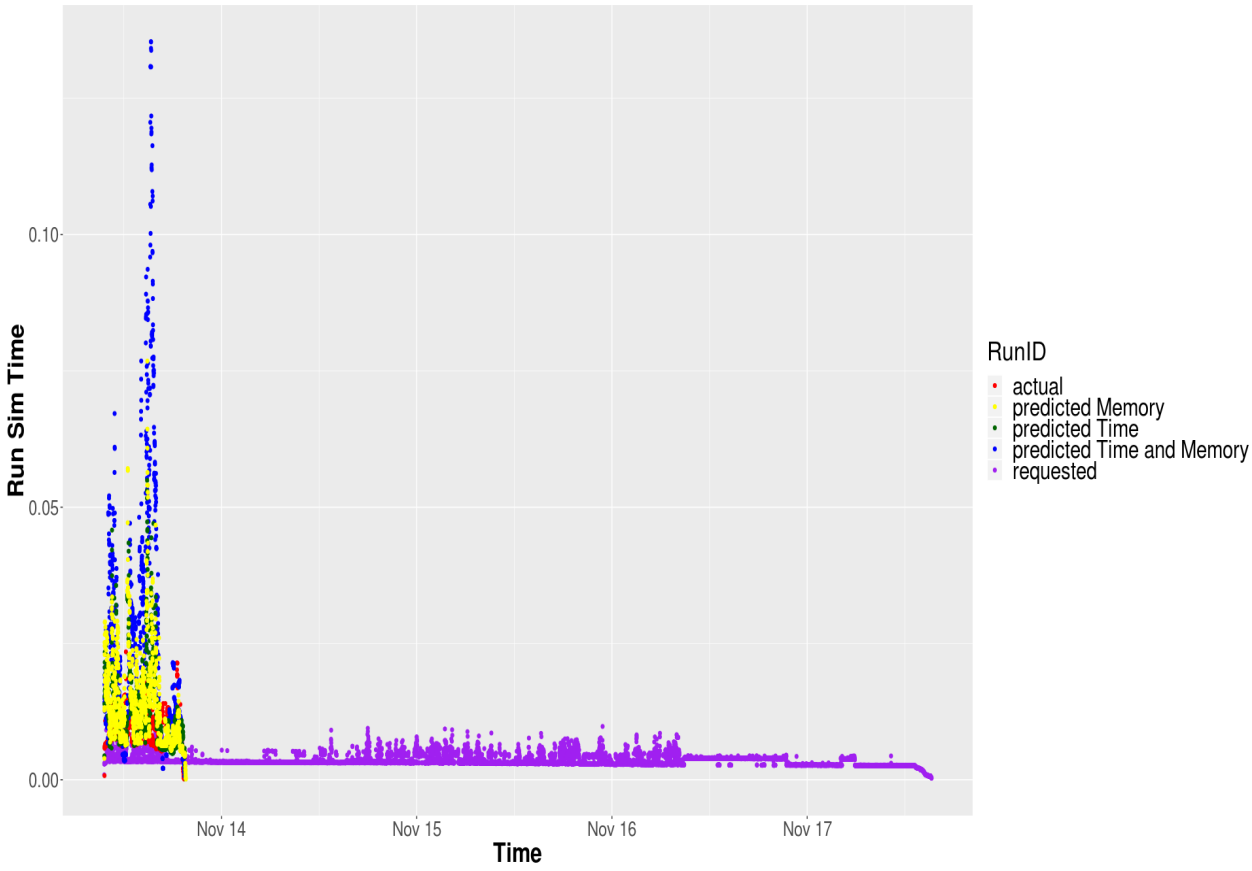

Figure 2.9: *ackfill-Sched Performance for (Requested vs Actual vs Required Time Predicted vs Memory Predicted vs Required Time and Memory Predicted)*

Chapter 3

Ensemble Prediction of Job Resources to Improve System Performance for Slurm-Based HPC Systems [1]

A paper accepted by Practice and Experience in Advanced Research Computing (PEARC '21), July 18–22, 2021, Boston, MA, USA

3.1 abstract

In this paper, we present a novel methodology for predicting job resources (memory and time) for submitted jobs on HPC systems. Our methodology is 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 of HPC systems, thus reducing power consumption as well. Our model involves using Several supervised machine learning discriminative models from the scikit-

[1] This chapter is a slightly modified version of our published article [27]

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 RMACC-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.

3.2 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 a lack of knowledge about the structure and implementation of HPC systems, running applications, and size of submitted jobs. Moreover, there is no existing 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[28], improving and expanding our previous work[4 29] 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.

3.2.1 Why the Slurm Workload Manager and Slurm Simulator?

Improving the performance of Simple Linux Utility for Resource Management (Slurm)[30] 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 wherein the cluster[5]. Slurm is the most popular job scheduler over all of the other schedulers such as SGE (Sun Grid Engine)[8], TORQUE (Tera-scale Open-source Resource and Queue manager)[10], PBS (Portable Batch System)[11], and the Maui Cluster Scheduler[31]. Testing our model on a real cluster is a hard process due to security and reliability issues, impacting the real cluster and the time needed to run all of the jobs needed for testing. Hence, The Slurm simulator developed by the 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[14]. The Slurm simulator is located in the Github[13]

3.3 Related Work

Job scheduling in HPC systems is 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.[32]. While there are some other main scheduling strategies that researchers focuse on the past by using different priority functions such as Shortest Job First (SJF), First Come First Serve (FCFS), Priority Scheduling, etc.[33] which don't achieve maximum performance and utilization in multiple nodes clusters. Previous studies show that batch

job scheduling is an NP-complete problem[34].

There have been many studies focused on predicting the run time for running applications on the HPC systems or the cloud[35 36 37 38 39 40], while there are very few studies focuses on 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 ®)[41].

Fan et al. proposed another HPC scheduling technique using deep reinforcement learning (DARS), which uses neural networks for resource reservation and backfilling.[42].

Similarly, Zhang et al. proposed a deep reinforcement learning-based job scheduler called RLScheduler which is capable of learning high-quality scheduling policy.[43]. Some other work focused on measuring power consumption based on submitted jobs on a cluster.[44].

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.[44]

Other different methods 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 specific applications during the running time. This method increases the application overhead by around 5%[45].

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 proposes a stand-alone ML model which dramatically improves efficiency, and utilization. In addition, our model will decrease turnaround time, waiting time for the submitted jobs.

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 3.1: *Feature Selected*

3.4 Methodology

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

3.4.1 Data Preparation and Feature Analysis

Two data sets (sacct data) were collected from the 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)[46]. The data set has **7.8 million** instances and covers the years from 2016 – 2019 of the usage. The second data set was collected from the HPC resources of Kansas State University (Beocat)[22]. The data side has **10.9 million** instances and covers 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 3.1**. 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[47] 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 the Scikit-learn Python package[48].

3.4.2 Regression Models

Our objective was to model time (CPUTimeRAW) and memory (MaxRSS) as a function of requested parameters Account, Timelimit, ReqNodes, ReqMem, ReqCPUS, and QsS. Thus, we considered the following seven popular regression models for the task: **i)** Lasso Least Angle Regression (LL)[49,50], **ii)** Linear Regression (LR)[50], **iii)** Ridge Regression (RG)[50], **iv)** Elastic Net Regression (EN)[50], **v)** Classification and Regression Trees (DTR)[51], **vi)** Random Forest Regression (RFR)[52], and **vii)** LightGBM (LGBM)[53]. We used the Coefficient of determination (R^2)[50] and root mean squared error (RMSE)[50] to evaluate the regression models. We used scikit-learn's[48] implementation for all models and performance metrics.

3.4.3 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 an 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)

```
1  unacc = unique(acc)
2  acc_pool = {}
3  Repeat num_acc times:
4      for i ∈ unacc
5          tac = append(acc_pool, i), if not i ∈ acc_pool
6          indices = which tac ∈ acc
7          X_a, Y_a = X[indices], Y[indices]
8          Repeat 20 times:
9              Split X_a, Y_a into 80% training and 20% testing.
10             RM = Build model using m.
11             Calculate training and testing R^2 of RM.
12         R2_tr[i] = mean R^2 of RM using training data.
13         R2_te[i] = mean R^2 of RM using testing data.
14     best_aid = Choose i with best mean R2_tr and R2_te ranks.
15     best_r2_tr = R2_tr[best_aid]
16     best_r2_te = R2_te[best_aid]
17     acc_pool = append(acc_pool, best_aid)
18 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.

3.5 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.

3.5.1 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. **Figurse 3.1, 3.2, 3.3, 3.4, 3.5, and 3.6** 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. **Figurse 3.7, 3.8, 3.9, 3.10, 3.11, and 3.12** 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 BEOCAT for each regression model. We found similar trends among 20 accounts in both RMACC-Summit and 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.

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. **Figures 3.13, 3.14, 3.15, and 3.16** shows the mean R^2 score distribution of DTR, RFR, and LGBM on testing data versus the number of best account combinations. It can be seen that the R^2 decreases as the number of accounts (and jobs) increases. 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 combina-

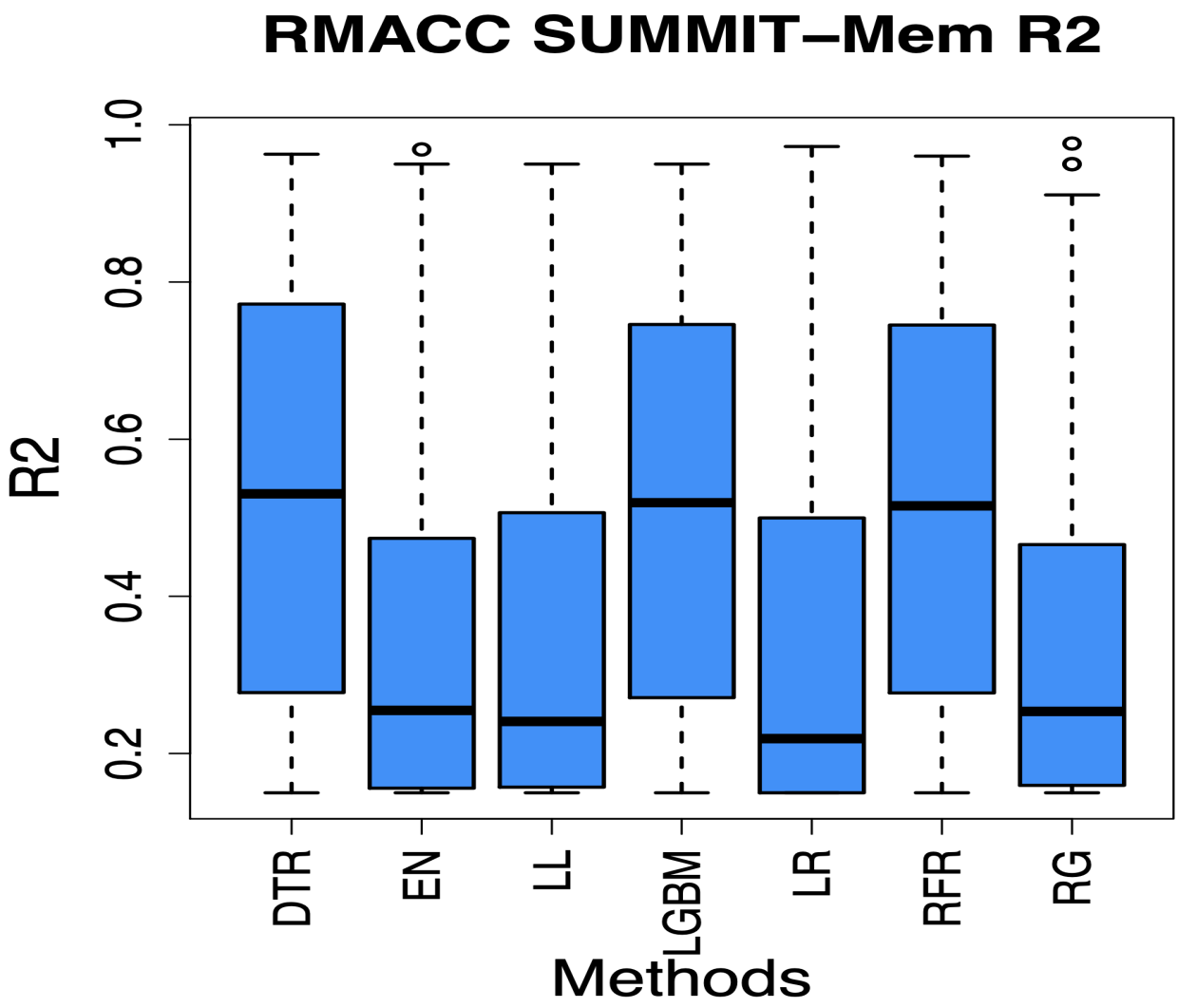

Figure 3.1: *R2 for predicting memory of seven methods across 50 accounts in RMACC-Summit*

tions 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

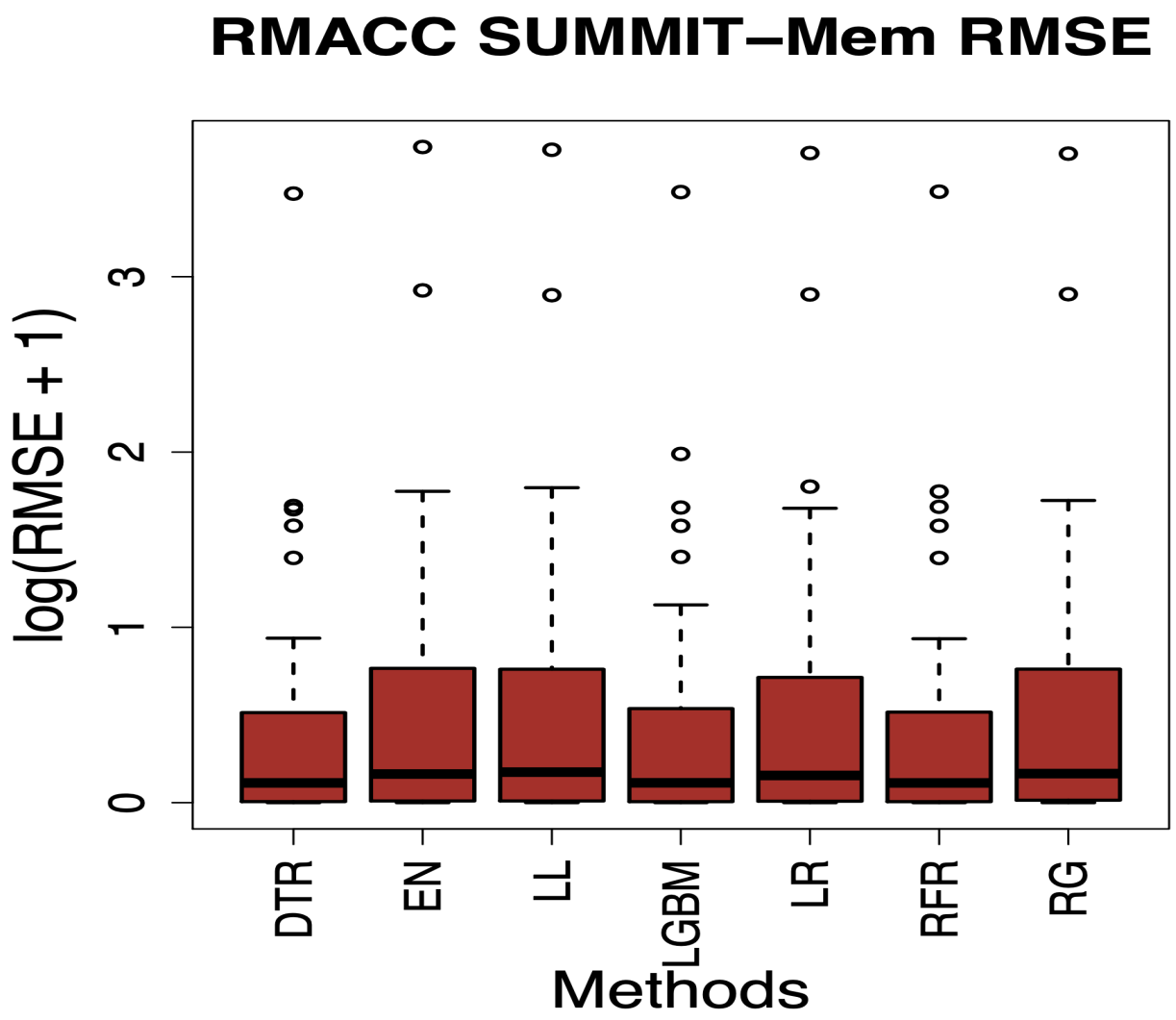

Figure 3.2: *RMSE for predicting memory of seven methods across 50 accounts in RMACC-Summit*

R^2 of **0.78**, for building a DTR based time model in RMACC-Summit.

3.5.2 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

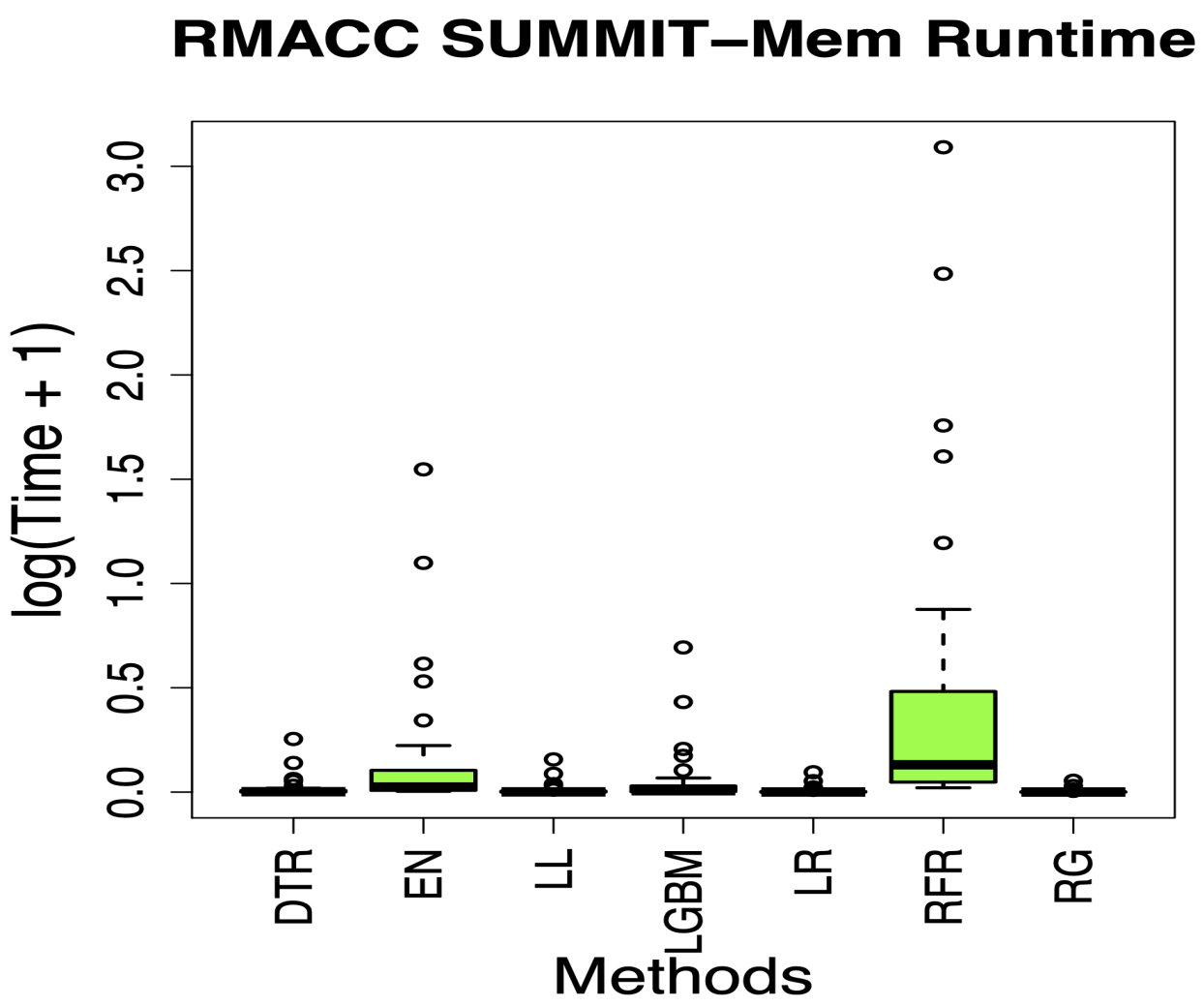

Figure 3.3: *Runtime for predicting memory of seven methods across 50 accounts in RMACC-Summit*

the job was submitted) and the execution time (the difference between the start and end execution time). **ii)** System Utilization which measures 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

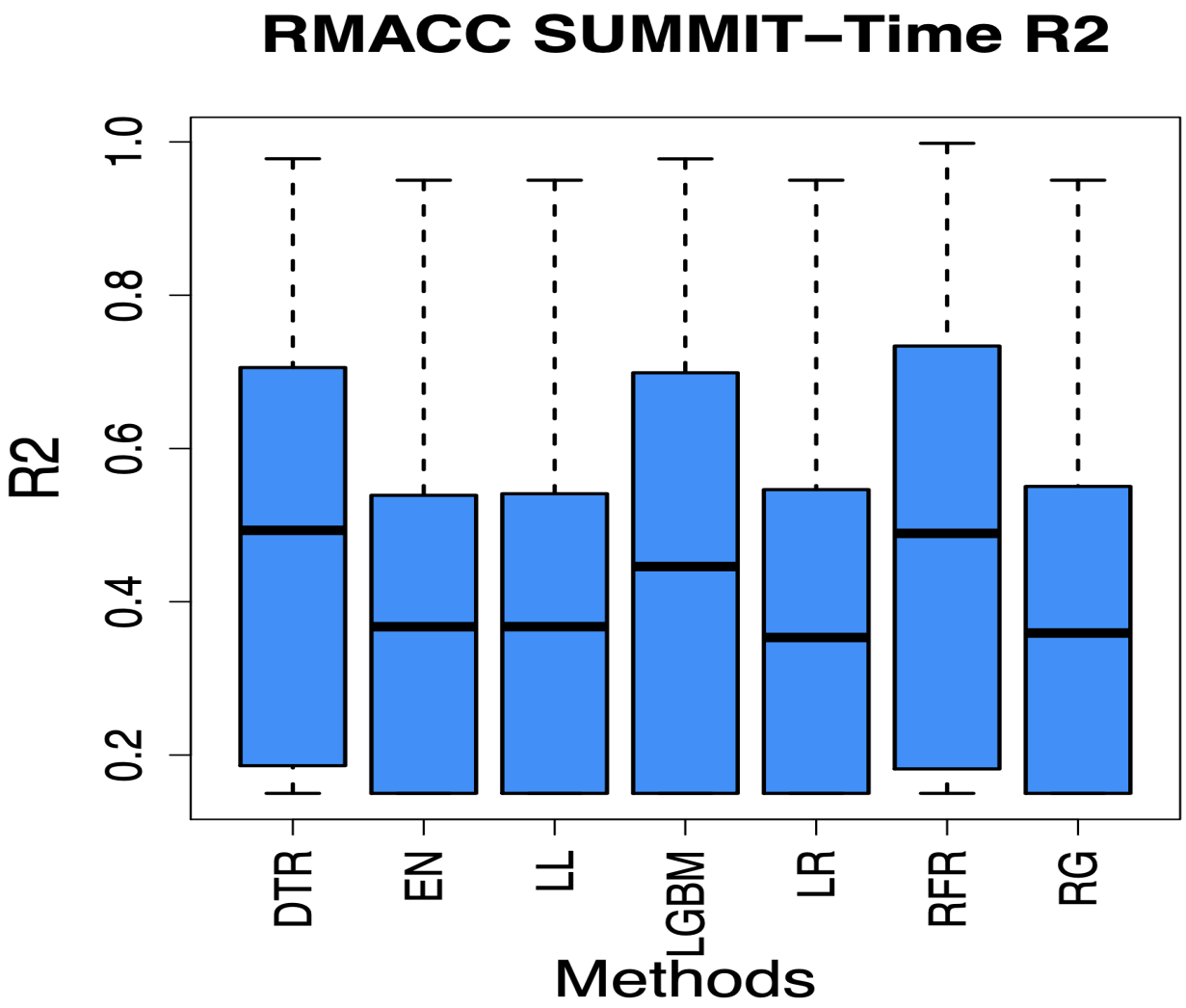

Figure 3.4: *R2 for predicting time of seven methods across 50 accounts in RMACC-Summit*

run time. Each RMACC-Summit and Beocat testbed contains one hundred thousand jobs.

Figure 3.17 and **Figure 3.18** show submission and execution time metrics 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

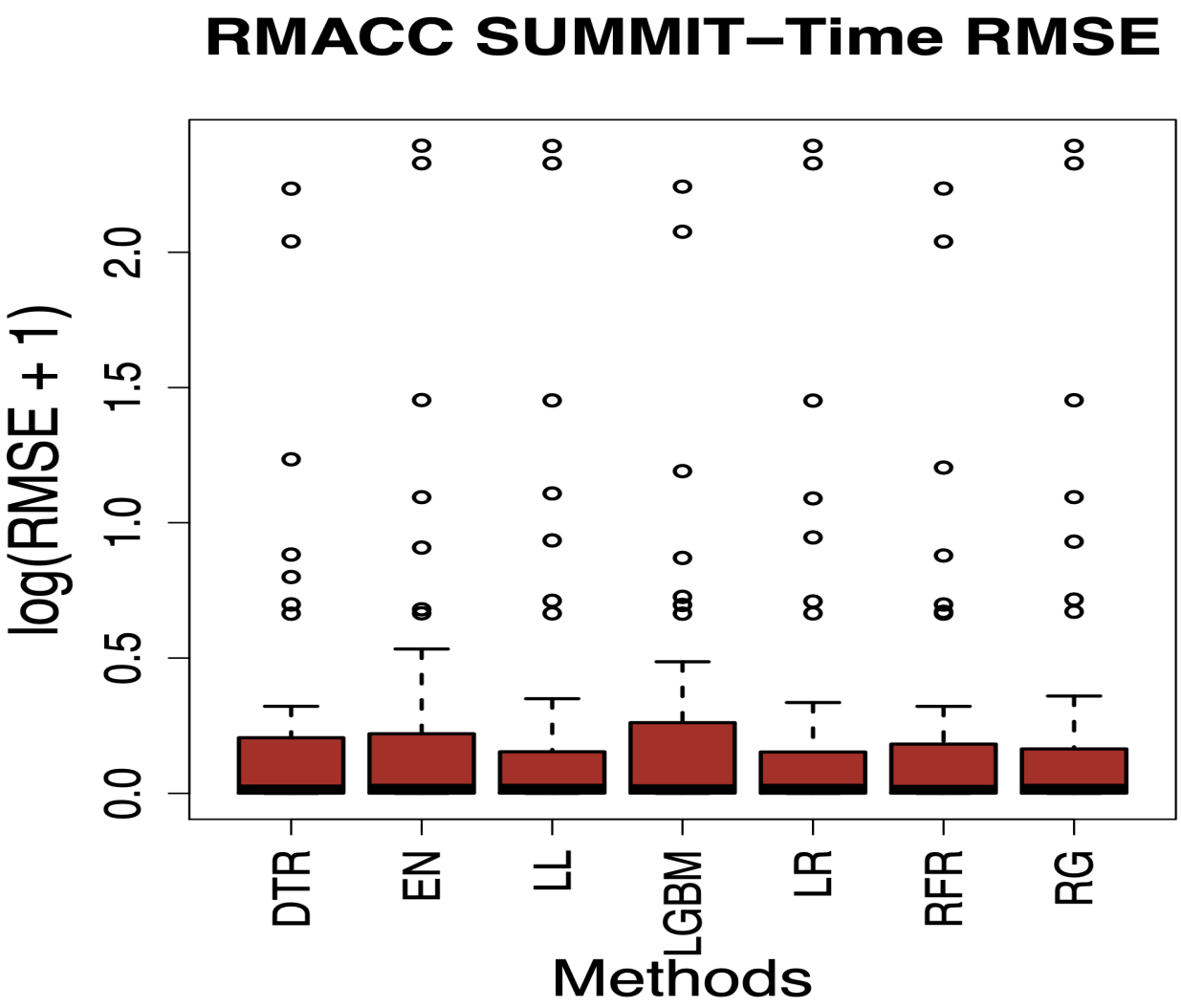

Figure 3.5: *RMSE for predicting time of seven methods across 50 accounts in RMACC-SUMMI*

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 3.19 and **Figure 3.20** 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 3.21 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 base on

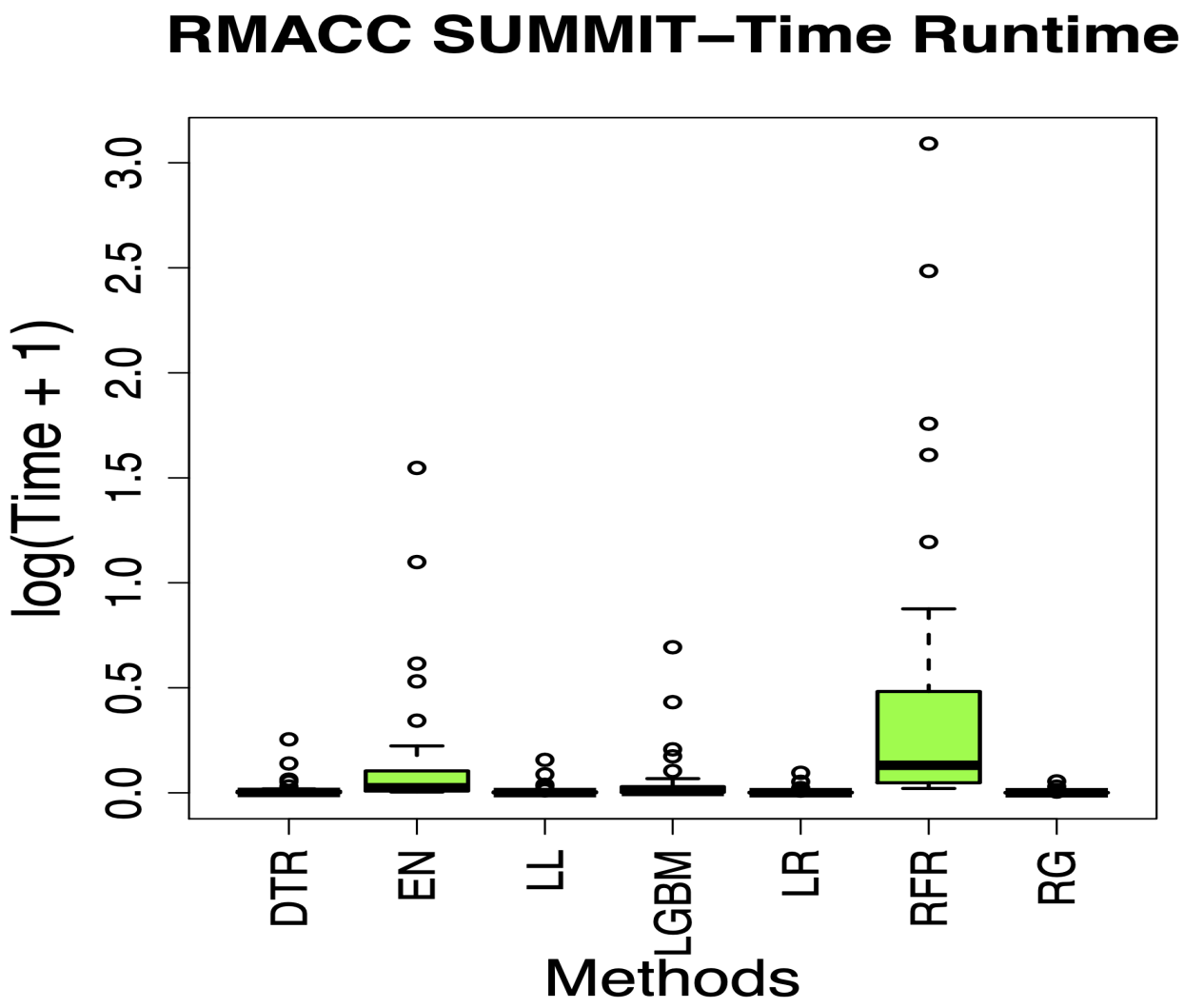

Figure 3.6: *Runtime for predicting time of seven methods across 50 accounts in RMACC-Summit*

measuring the density of jobs attempts to schedule over time. While our model achieved similar BEOCAT results for the 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.

Tables 3.2, 3.3, 3.4, and 3.5 provide the calculated average waiting time with the

	Average Wait Time (Hour)	Average TA Time (Hour)
Requested	380.6 ±241.2	403.14 ±243.3
Actual	1.3 ±0.7	2.9 ±3.2
Predicted	3.7 ±1.1	5.5 ±4.8

Table 3.2: *Average Waiting Time (Requested vs Actual vs Predicted) For RMACC-Summit*

	Median Wait Time (Hour)	Median TA Time (Hour)
Requested	401.2	425.7
Actual	0.5	1.1
Predicted	1.3	4.5

Table 3.3: *Median Waiting Time (Requested vs Actual vs Predicted) For RMACC-Summit*

	Average Wait Time (Hour)	Average TA Time (Hour)
Requested	662.9 ±193.6	673.5±196.6
Actual	1.5 ±1.1	4.1 ±2.2
Predicted	27.7±25.3	34.8 ±27.1

Table 3.4: *Average Waiting Time (Requested vs Actual vs Predicted) For RMACC-Summit*

	Median Wait Time (Hour)	Median TA Time (Hour)
Requested	681.6	652.2
Actual	0.9	3.2
Predicted	6.2	13.9

Table 3.5: *Median Waiting Time (Requested vs Actual vs Predicted) For RMACC-Summit*

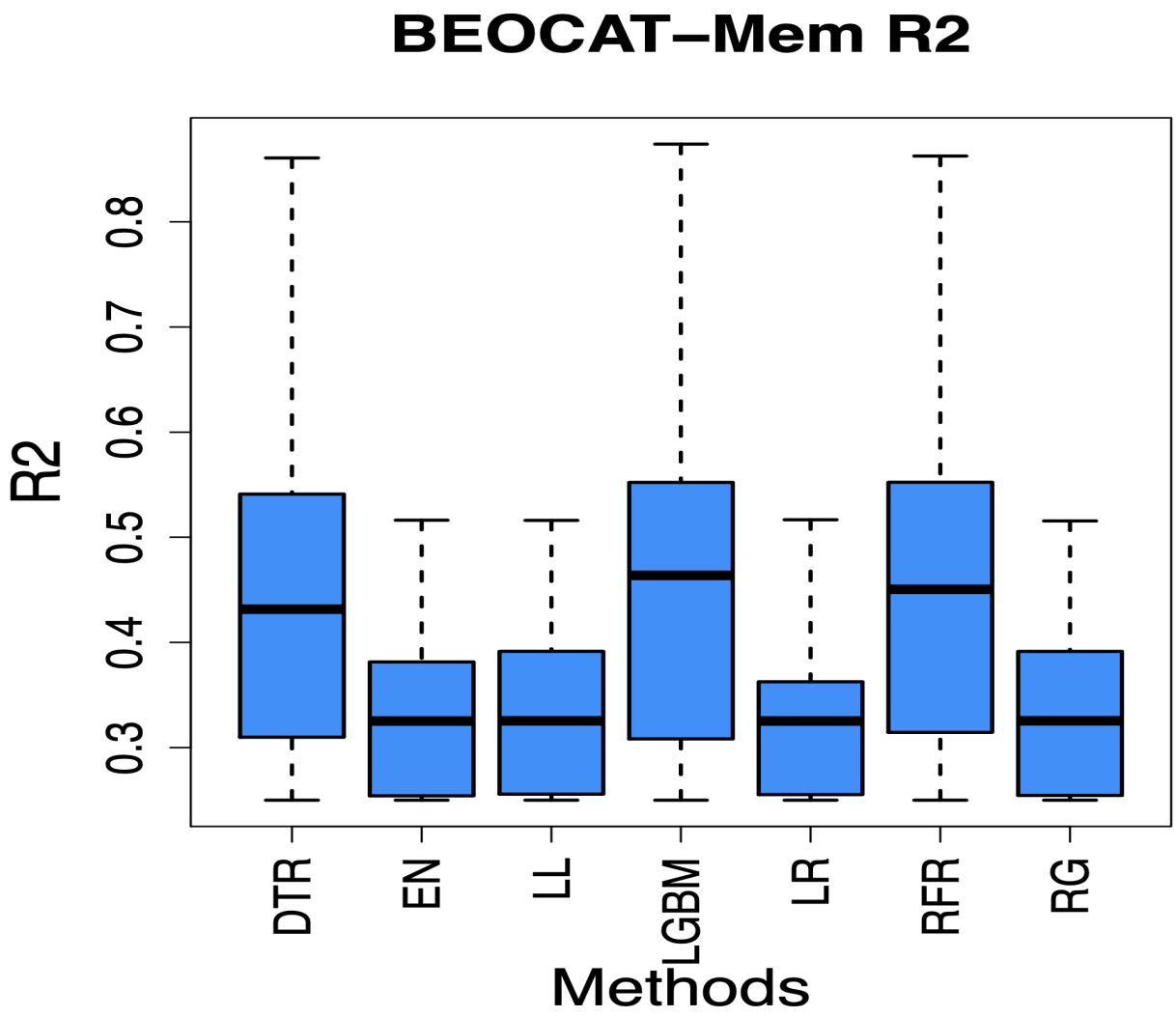

Figure 3.7: *R2 for predicting memory of seven methods across 50 accounts in BEOCAT*

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 the 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.

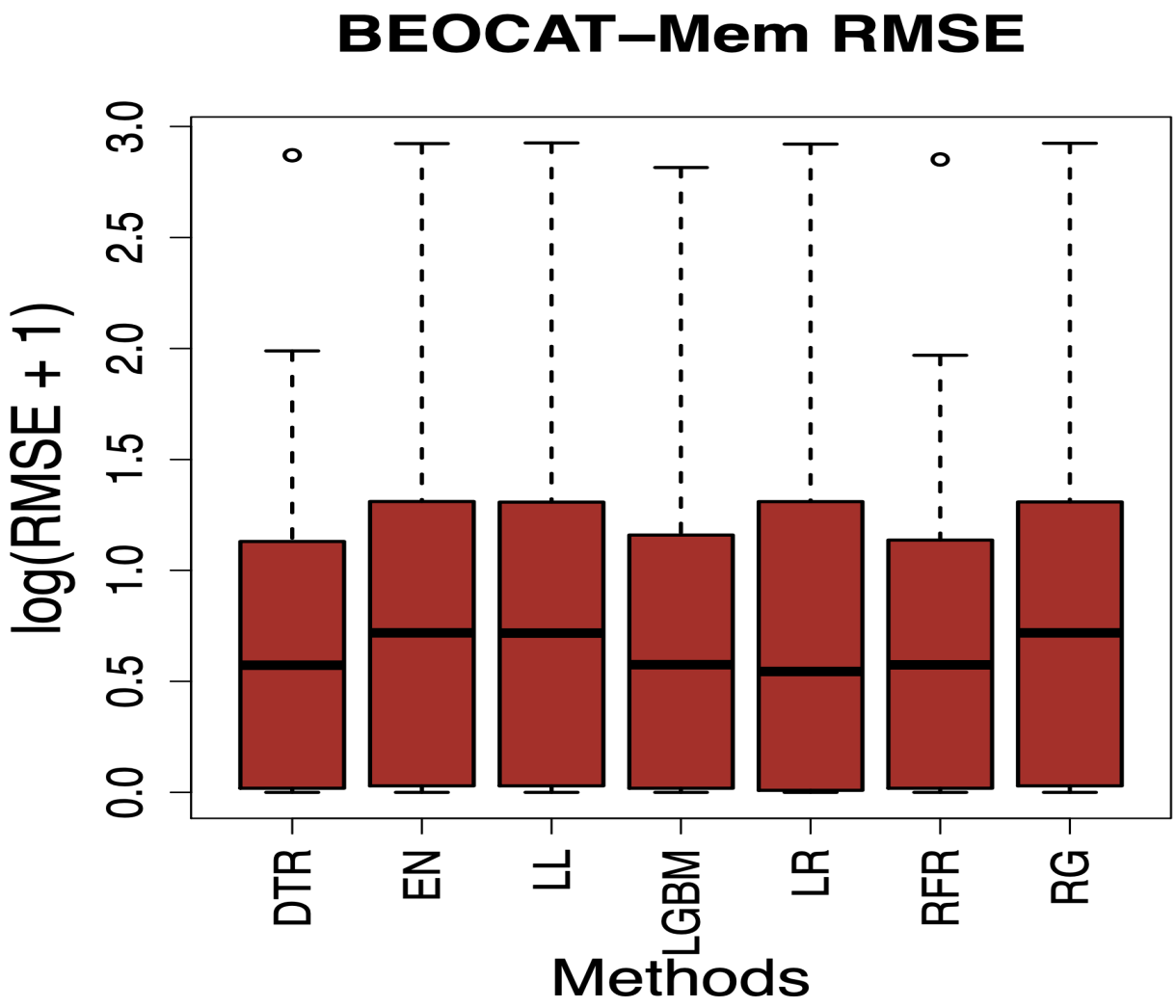

Figure 3.8: *RMSE for predicting memory of seven methods across 50 accounts in BEOCAT*

3.6 Summary

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) required for their submitted jobs on the HPC cluster. Our ML model is built based on the implementation of different machine learning algorithms (Six discriminative models 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 (RMACC-Summit)

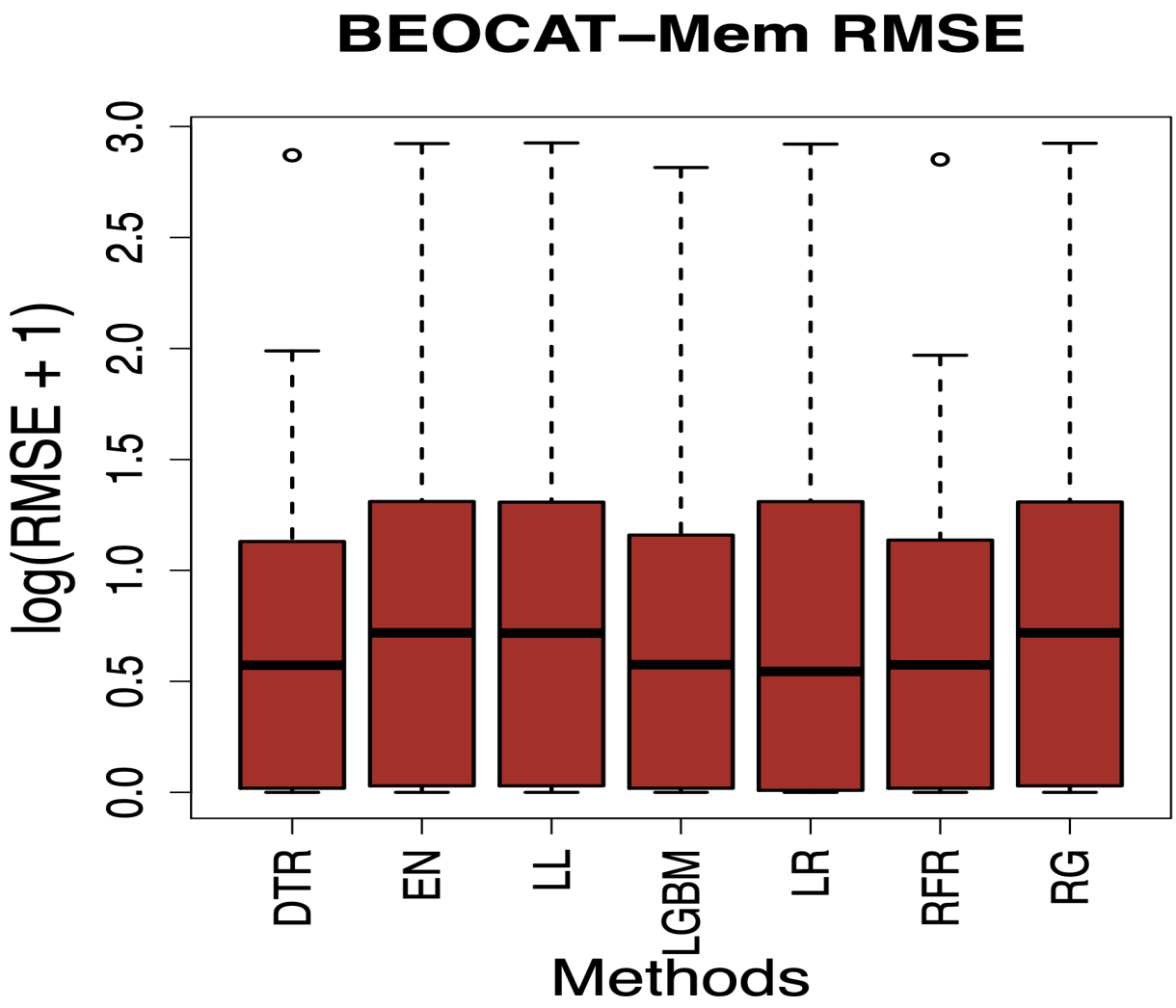

Figure 3.9: *Runtime for predicting memory of seven methods across 50 accounts in BEOCAT*

and Kansas State University (Beocat)) HPC resources. We tested our ML model using one hundred thousand jobs for each testbed.

Our results show 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

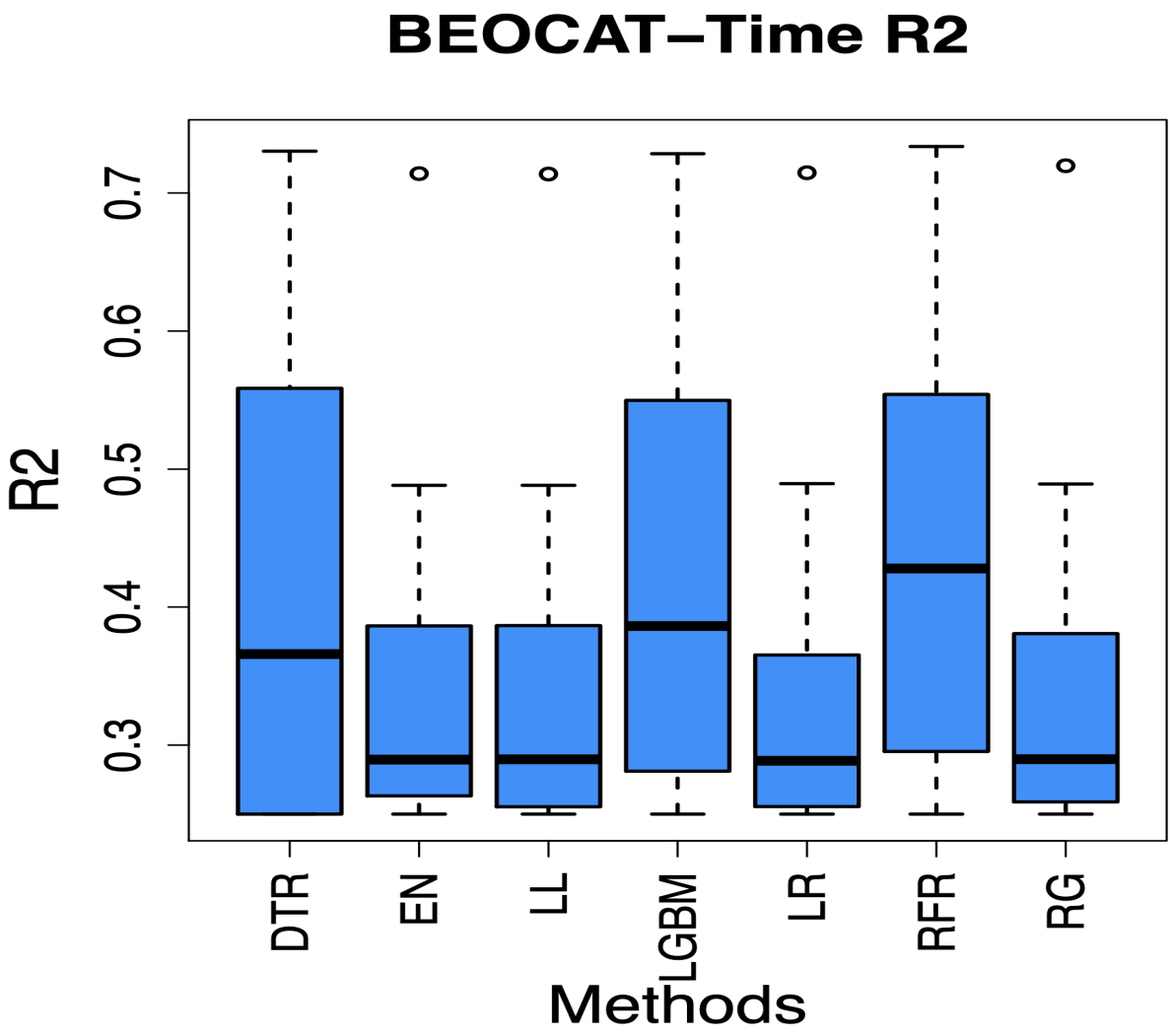

Figure 3.10: *R2 for predicting time of seven methods across 50 accounts in BEOCAT*

HPC resources.

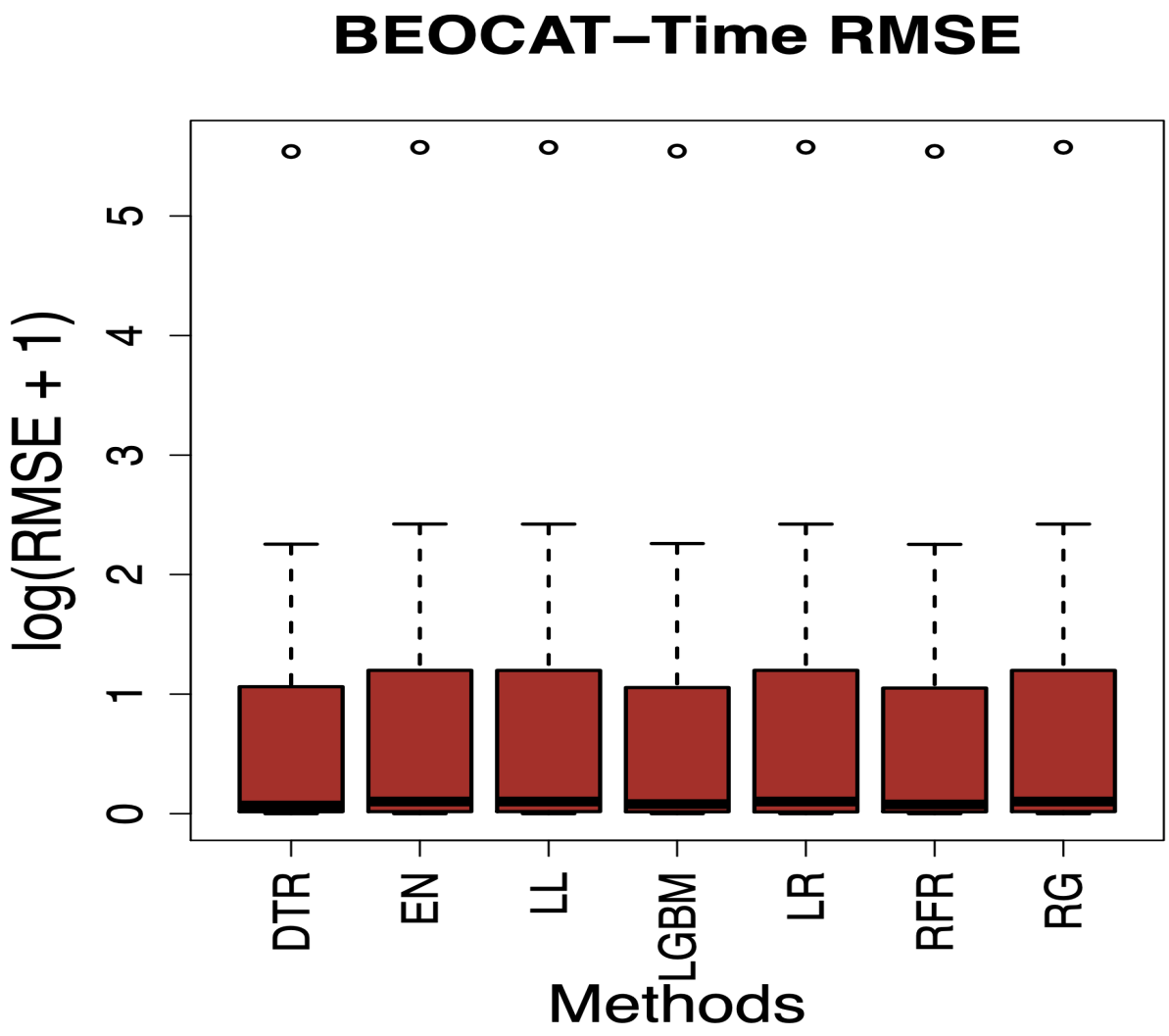

Figure 3.11: *RMSE for predicting time of seven methods across 50 accounts in BEOCAT*

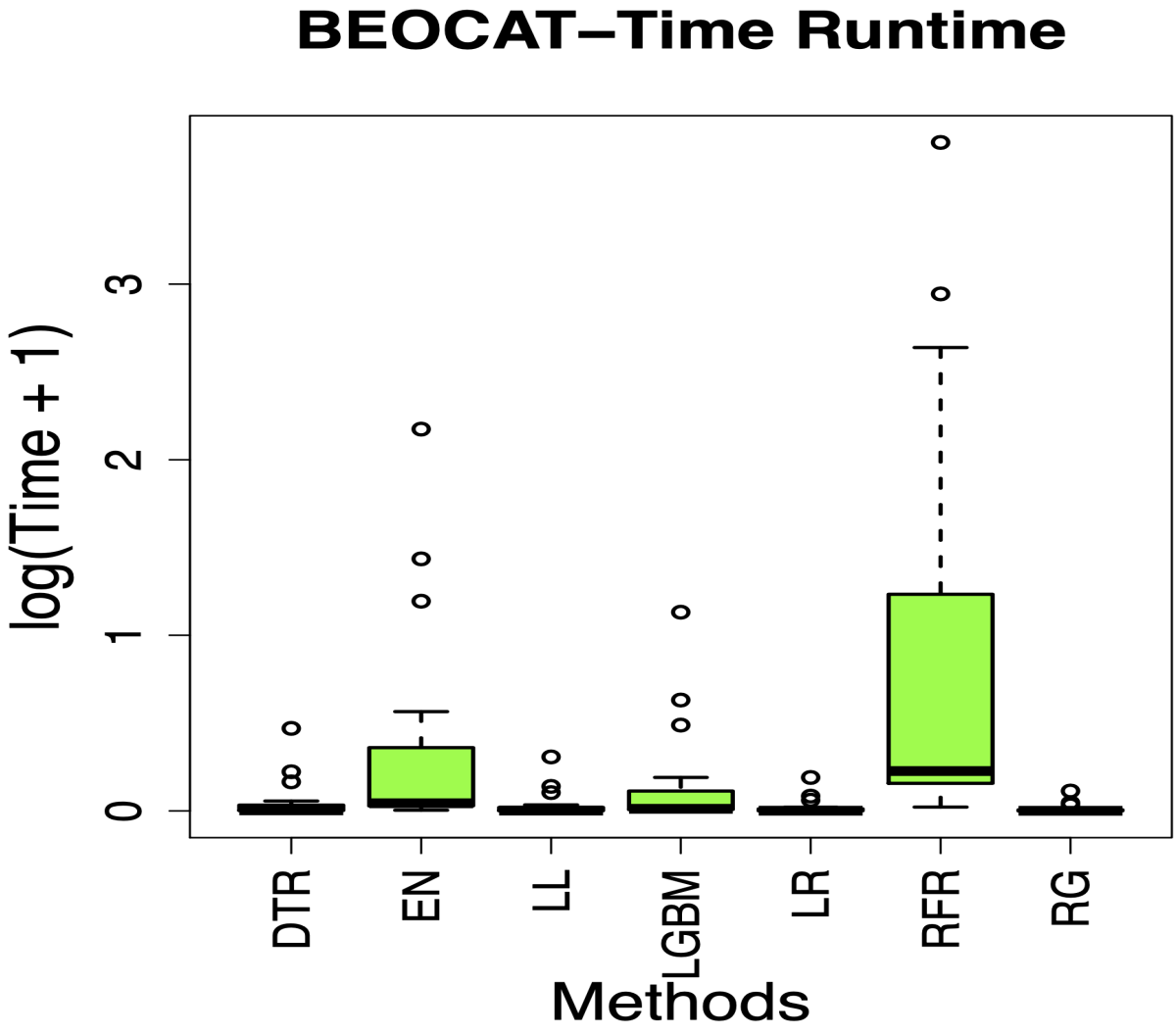

Figure 3.12: *Runtime for predicting time of seven methods across 50 accounts in BEOCAT*

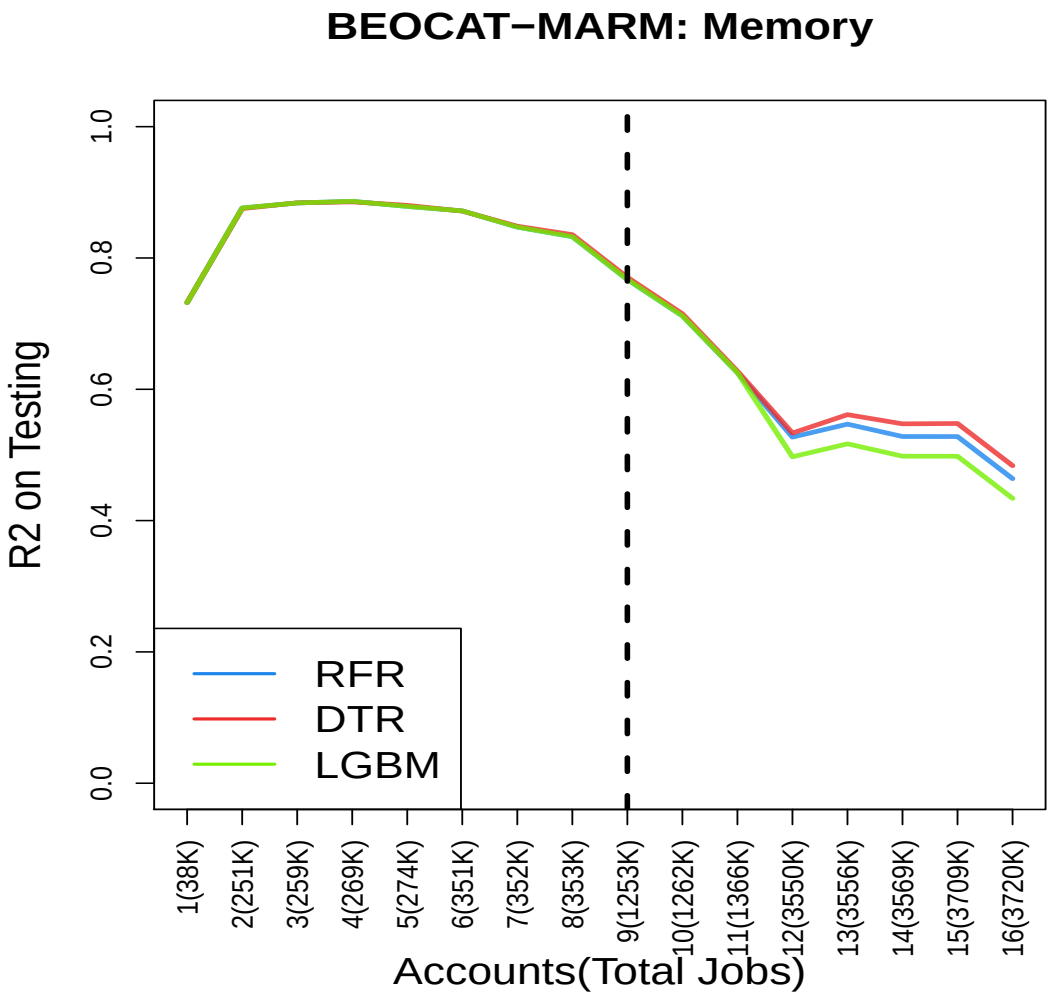

Figure 3.13: *R^2 versus Number of Accounts in predicting memory using MARM across BEOCAT*

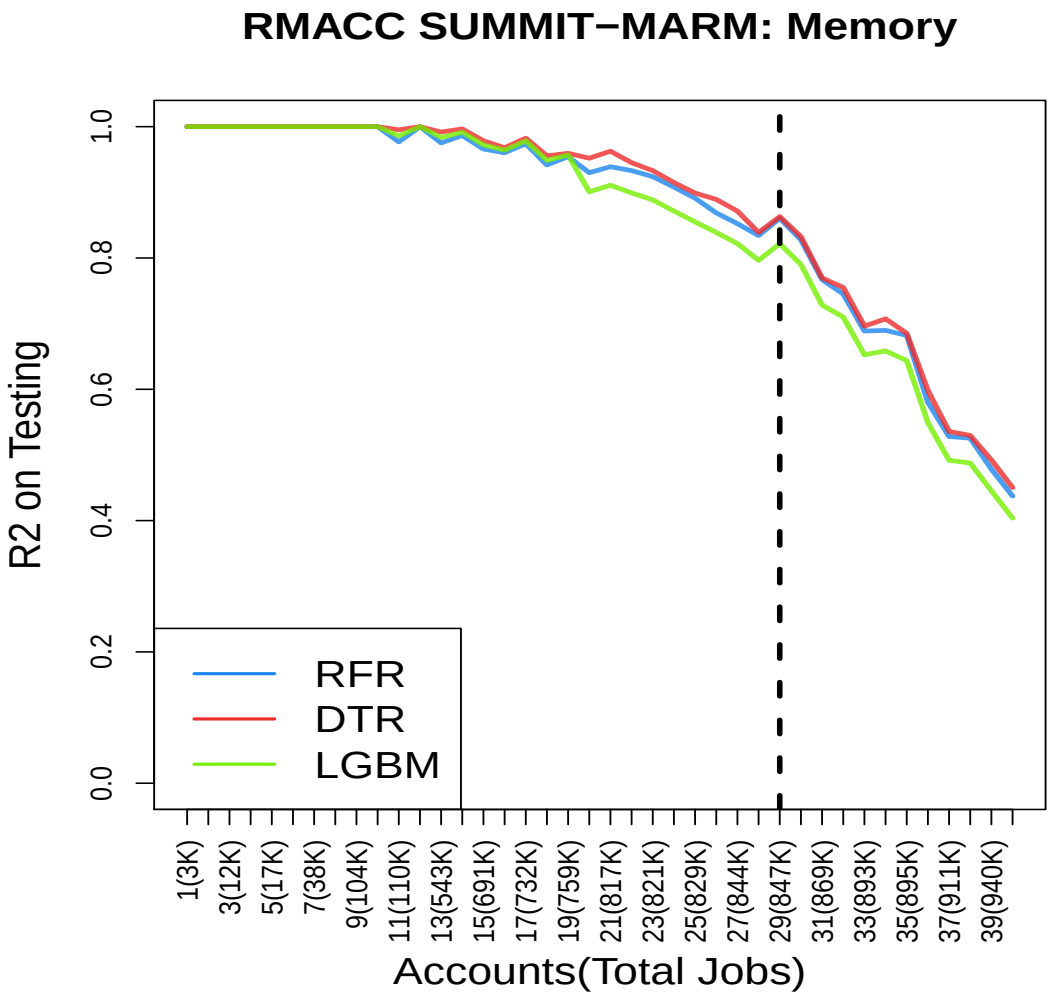

Figure 3.14: *R^2 versus Number of Accounts in predicting memory using MARM across RMACC-Summit*

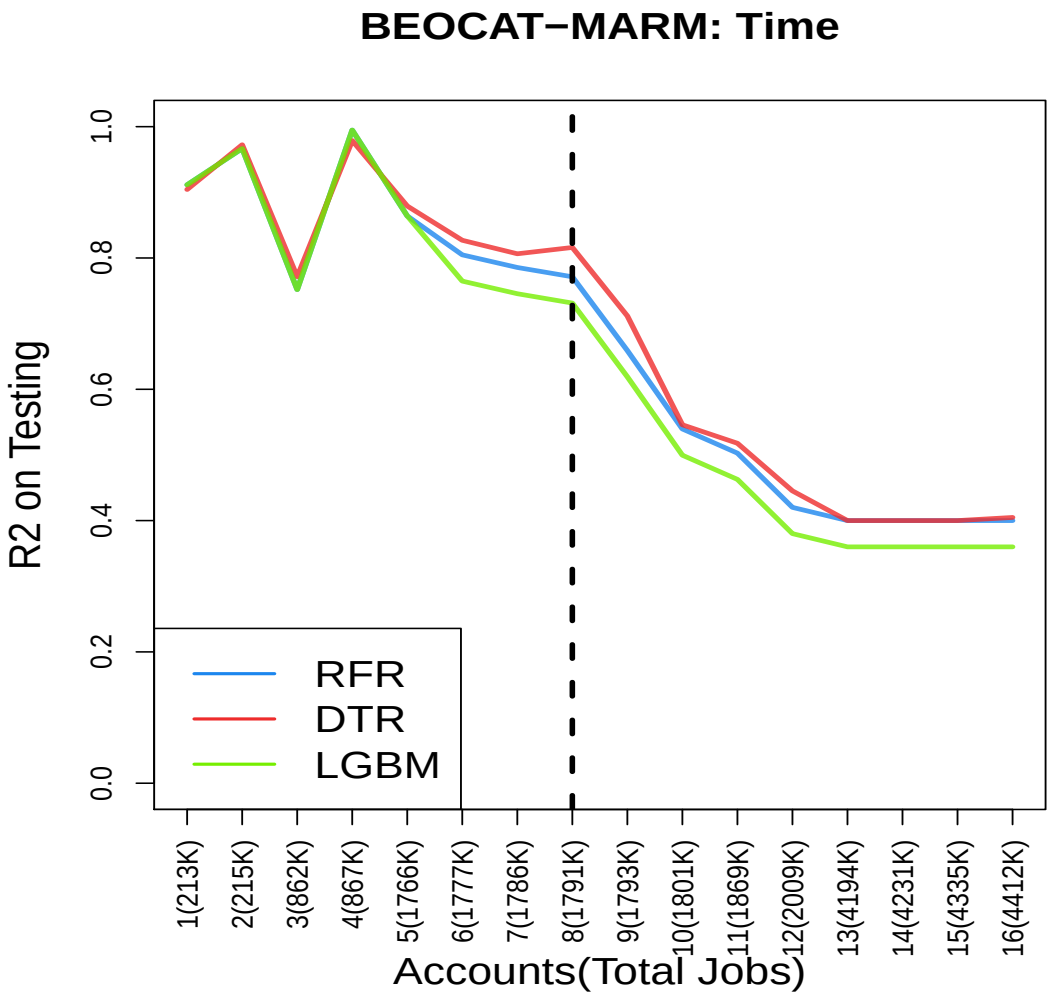

Figure 3.15: *R^2 versus Number of Accounts in predicting time using MARM across BEOCAT*

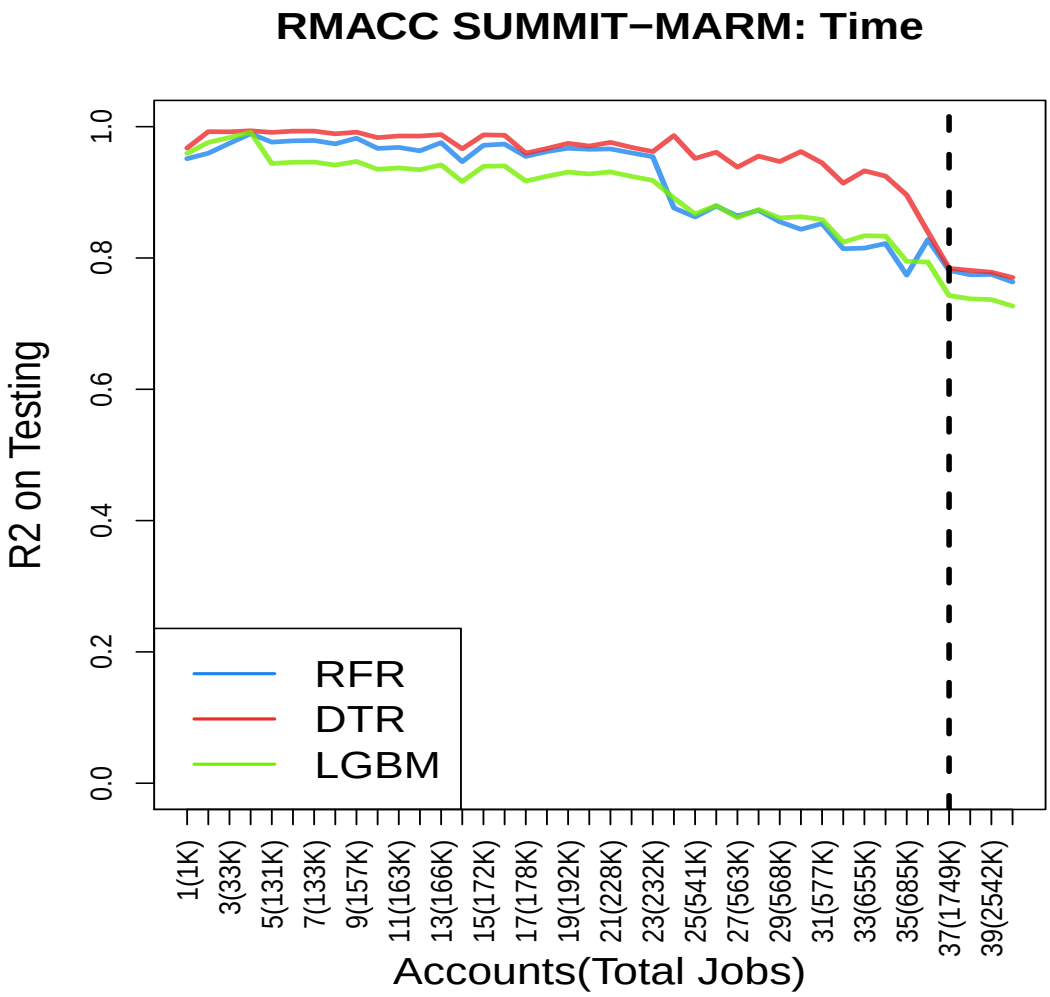

Figure 3.16: *R^2 versus Number of Accounts in predicting time using MARM across RMACC-Summit*

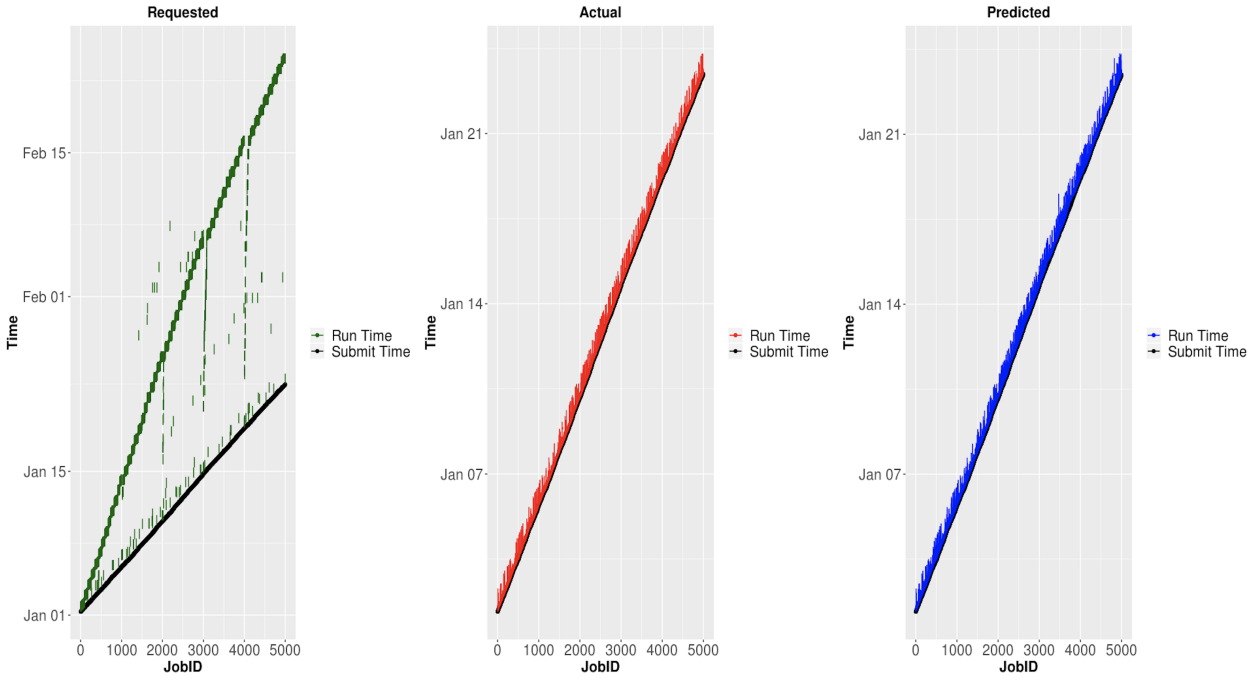

Figure 3.17: *Jobs Submission and Running time (Requested vs Actual vs Predicted) for RMACC-Summit Jobs. Note dramatic improvement of Y axis range between graphs.*

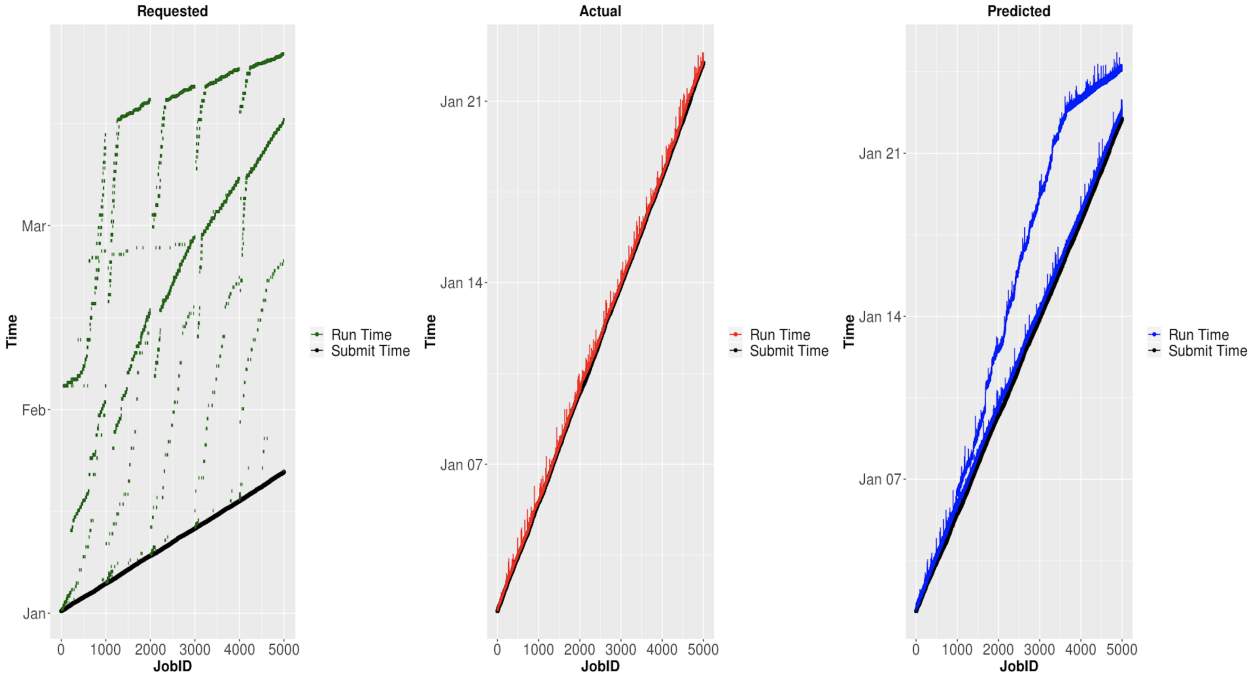

Figure 3.18: *Jobs Submission and Running time (Requested vs Actual vs Predicted) for Beocat Jobs. Note dramatic improvement of Y axis range between graphs*

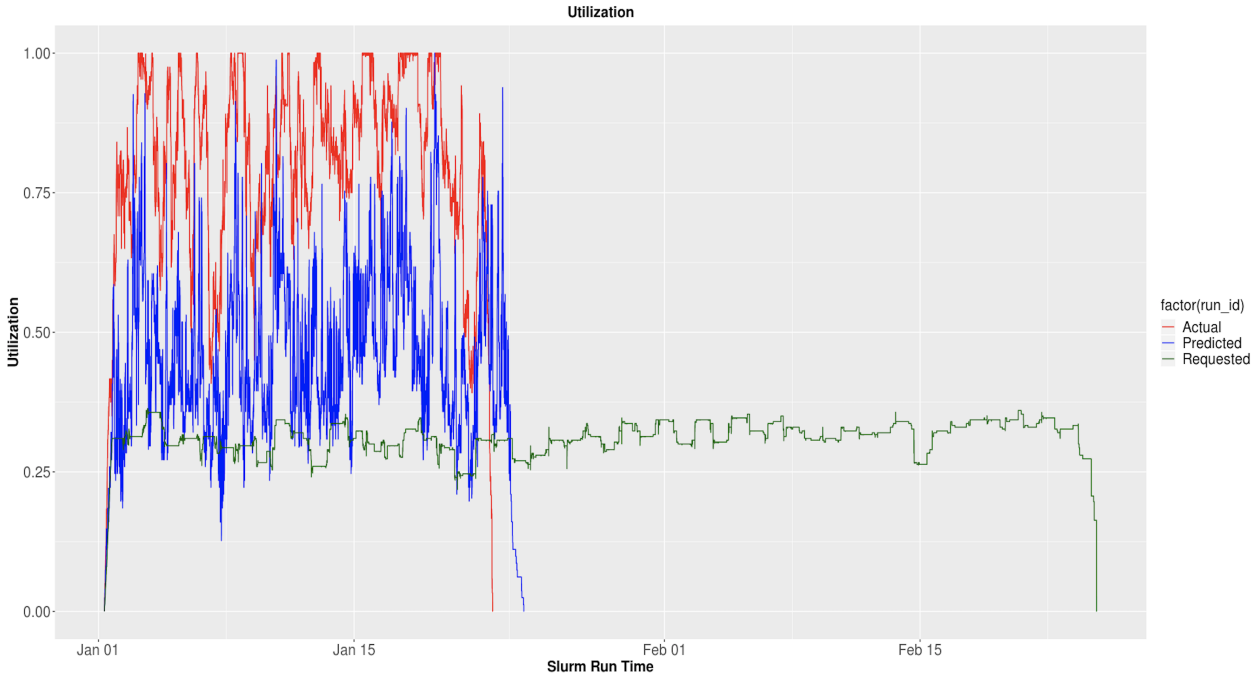

Figure 3.19: *Utilization (Requested vs Actual vs Predicted) for RMACC-Summit Jobs*

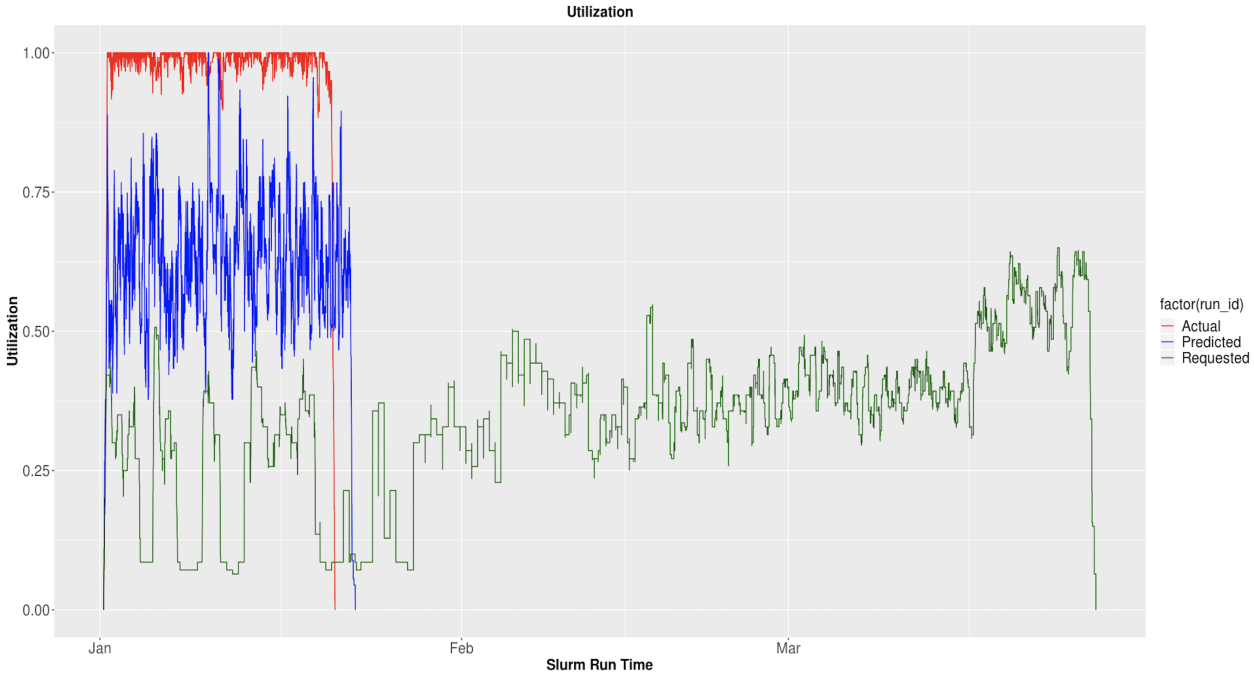

Figure 3.20: *Utilization (Requested vs Actual vs Predicted) for Beocat Jobs*

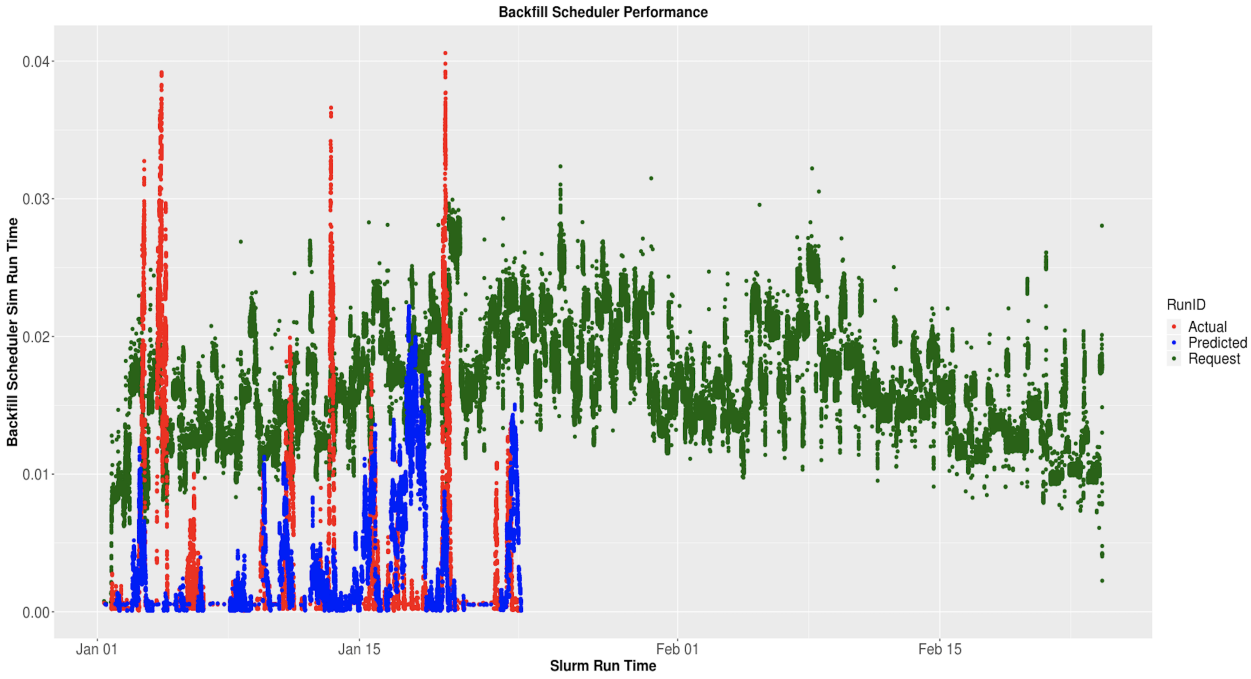

Figure 3.21: *Backfill-Sched Performance for RMACC-Summit Jobs*

Chapter 4

AMPRO-HPCC: A Machine-Learning Tool for Predicting Resources on Slurm HPC Clusters [1]

4.1 Abstract

Determining resource allocations (memory and time) for submitted jobs in High Performance Computing (HPC) systems is a challenging process even for computer scientists. HPC users are highly encouraged to overestimate resource allocation for their submitted jobs, so their jobs will not be killed due to insufficient resources. Overestimating resource allocations occurs because of the wide variety of HPC applications and environment configuration options, and the lack of knowledge of the complex structure of HPC systems. This causes a waste of HPC resources, a decreased utilization of HPC systems, and increased waiting and turnaround time for submitted jobs. In this paper, we introduce our first ever implemented fully-offline, fully-automated, stand-alone, and open-source Machine Learning (ML) tool to help users predict memory and time requirements for their submitted jobs on the cluster.

[1] This chapter is a slightly modified version of our published article [54]

Our tool involves implementing six ML discriminative models from the scikit-learn and Microsoft LightGBM applied on the historical data (sacct data) from Simple Linux Utility for Resource Management (Slurm). We have tested our tool using historical data (saact data) using HPC resources of Kansas State University (Beocat), which covers the years from January 2019 - March 2021, and contains around 17.6 million jobs. Our results show that our tool achieves high predictive accuracy R^2 (0.72 using LightGBM for predicting the memory and 0.74 using Random Forest for predicting the time), helps dramatically reduce computational average waiting-time and turnaround time for the submitted jobs, and increases utilization of the HPC resources. Hence, our tool decreases the power consumption of the HPC resources.

4.2 Introduction

High Performance Computing (HPC) resources have become more available to users to run their extensive computations and simulations. One of the most important parts of the HPC system is the batch scheduler. The batch scheduler manages resources and queues of all submitted jobs in the cluster. Hence, it is the part that decides where and when jobs will run in the cluster. On the other hand, batch scheduler performance depends on the resource requirements from the user such as the amount of memory, requested time, and the number of cores[32]. While these resource requirements are the responsibility of HPC users to determine, it is a fact that users may determine resource needs inaccurately[55]. Also, users are highly encouraged to overestimate these resources in order to satisfy job requirements, so their jobs will not be killed during the run time due to insufficient resources[56]. Overestimating job resource requirements negatively impacts the performance and the utilization of the HPC system. Moreover, over-estimating job resource process will increase average turn-around time and average waiting time for submitted jobs.

In this paper, we introduce the first-ever open-source, stand-alone, highly-accurate, fully-

offline, and fully-automated tool called AMPRO-HPCC, which stands for "A Machine-Learning-Tool for Predicting Resources On Slurm HPC Clusters". AMPRO-HPCC aims to help HPC users predict and estimate the required job resource allocations (memory and time) for their submitted jobs. Our tool uses Simple Linux Utility for Resource Management (Slurm) historical logs-data (sacct) and involves implementation of six Machine Learning (ML) discriminative models from the scikit-learn[48] and Microsoft LightGBM (LGBM)[53]. Our ML tool is invoked through Command Line Interface (CLI), and it consists of two parts: **i)** System administrator part, which is responsible for preparing data and all the required models for building the final models and tool; **ii)** HPC user side, which will automatically read the submission job script provided from the HPC user and recommend the required job allocation resources (memory and time) for the associated submitted job.

We have extended our previous work[28,27,4], and designed the AMPRO-HPCC tool to help HPC users determine the allocation of HPC resource needs (memory and time) using supervised ML over historical data (sacct). Our open-source tool can be found on GitHub[57].

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

4.3 Related Work

Simple Linux Utility for Resource Management (Slurm) is a resource manager, which enables HPC resources to execute parallel jobs efficiently[30]. 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, manages the nodes, and keeps track of architecture within a node such as sockets, NUMA boards, cores,

hyper threads, memory, interconnect, generic resources, and managing licenses. Slurm manages jobs through varieties of scheduling algorithms (fair share, gang, advanced reservation, etc.)[12].

While there are many kinds of resource management scheduler such as Sun Grid Engine (SGE)[8], Tera-scale Open-source Resource and Queue manager (TORQUE)[58,10], and Portable Batch System (PBS)[59,11], Slurm is the most popular and most used among them. Hence, we implemented our tool based on Slurm workload manager HPC systems.

There are many studies and research focusing on predicting the running time and the time required for running application on the HPC systems or the cloud[35,60,36,15,37,61,38,62,63,39,40,45,44], while there are quite a lot of research that focuses on predicting the amount of memory required for the submitted jobs[41,64].

Our work differs by the methodology used and the ability to predict both memory and time required for submitted jobs on the HPC systems. We conclude "there does not yet exist software that can help to fully automate the allocation of HPC resources or to anticipate resource needs reliably by generalizing over historical data, such as determining the number of processor cores and the amount of memory needed."[28]. Hence, we are introducing the first-ever open-source ML tool for predicting job resources (memory and time) for submitted jobs on the HPC systems.

4.4 Prediction Tool (AMPRO-HPCC)

Figure 4.1 illustrates the use-case diagram of our ML tool. We have two types of users: i) system administrators (referred to as admin henceforth) and ii) HPC users (referred to as users henceforth). Modules `PreProcess`, `BuildPerAccountModels`, `BuildMixedAccountModels` and `TrainSelectedMARM` are available to admins, while the `Ampro-hpcc` module is available to both admins and users. The main objective of our tool is to build Mixed Account Regression Models (MARM), which are regression models built

on a subset of slurm *Accounts* with the best overall predictive performance, containing a reasonable percentage of jobs. Here, we provide descriptions of each module along with its inputs and outputs.

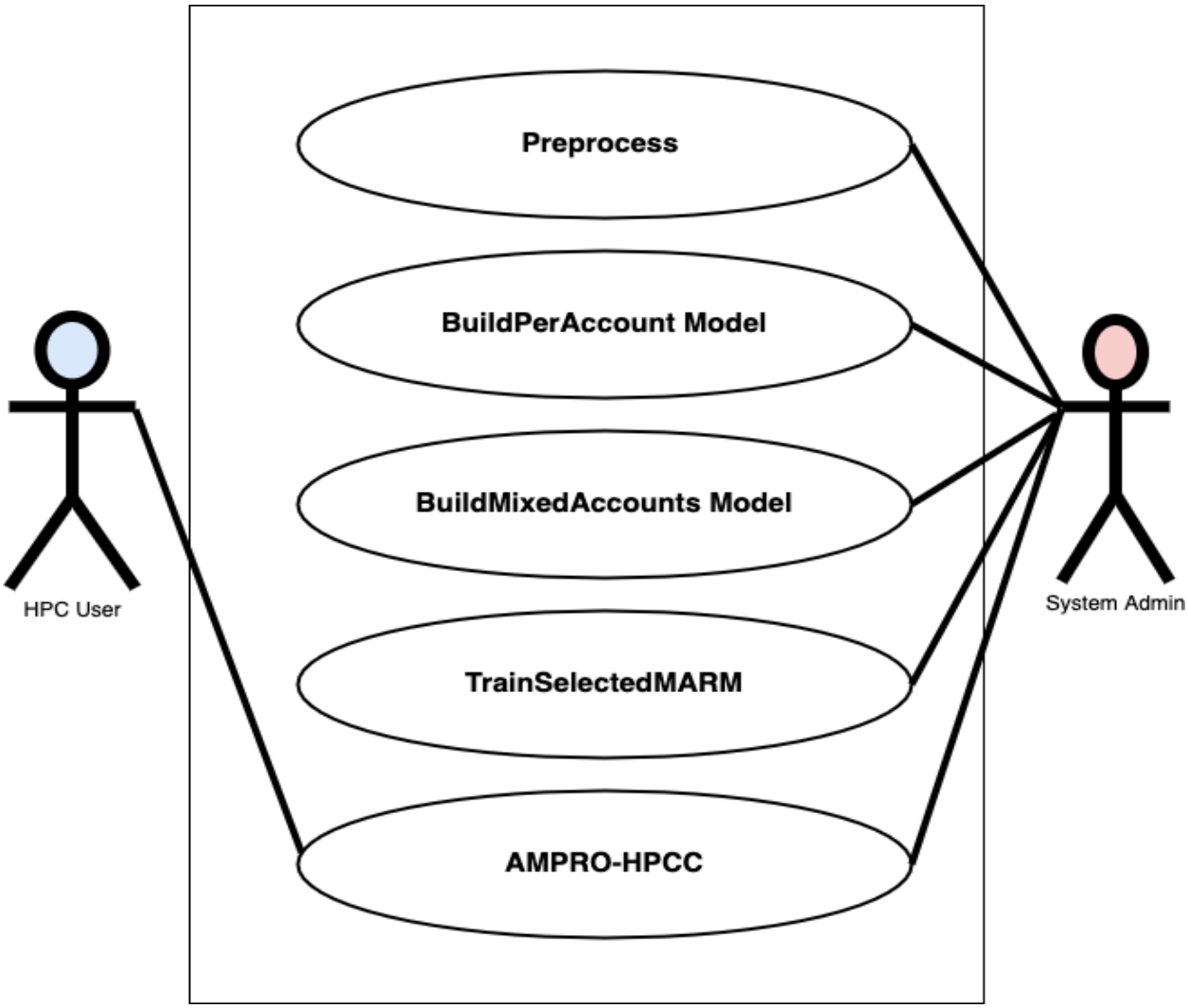

Figure 4.1: *Use-Case Diagram for AMPRO-HPC*

4.4.1 AMPRO-HPCC Workflow Model

Figure 4.2 describes the workflow model of our work as follows: i) The user prepares and creates a new job, which includes the requested amount of memory, time limit, quality of

service (QoS), and partition name for the proposed job. ii) The HPC user will submit their job and passes it through our ML model in order to predict the amount of the required memory and the amount of time needed for the job to run. iii) Our ML model will process the submitted job by parsing all of the parameters needed, then predicting required memory and time for the specific job. iv) The HPC user will get feedback from our model regarding the needed amount of memory and time for their submitted jobs. v) The user will have the option to confirm or deny to use the predicted values for the required memory and time. vi) If the user confirms the use of the predicted amounts for either the required memory or the required time or both, then our ML model will update the amounts of memory and time as needed for the submitted job. If not, then the submitted job will remain the same. vii) The user will be notified about the changes to their jobs. viii) Finally, either an updated job or the original job will be scheduled for running on the cluster.

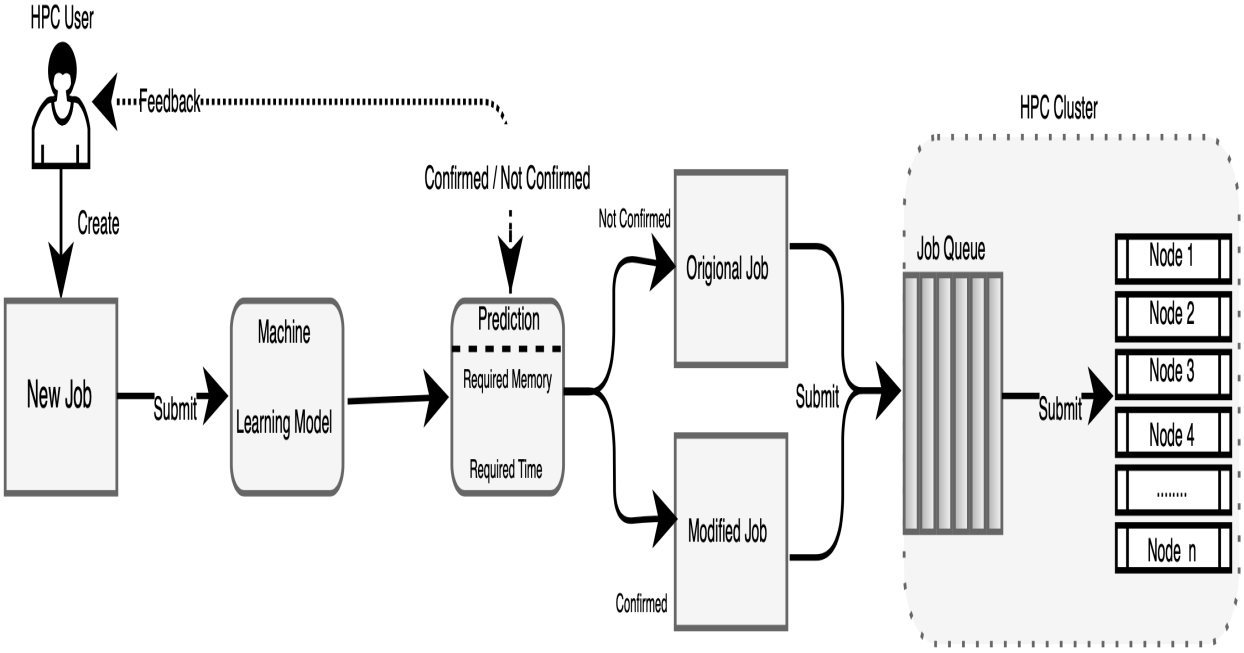

Figure 4.2: *AMPRO-HPCC Work-Flow Diagram*

4.4.2 Data Preparation

The data preparation or `Preprocess` module takes the path (`path_to_data`) to logs of slurm jobs accounting information (sacct) to extract *Account, ReqMem, Timelimt, ReqNodes, ReqCPUS, QoS, Partition, MaxRSS, CPUTimeRAW, and State* from the dataset. A description of these features can be found at[65]. The module also asks the admin to provide default time-limit (`def_time`), default quality of service (`def_qos`), and default partition assignment (`def_partition`) to deal with some of the missing values in the data. Finally, the admin also has the ability to specify a set of QoS (`sel_qos`) and partitions (`sel_partition`) that they want to select over the entire data. In addition, the `Pre-processing` module does its own filtration by only selecting jobs with *State* equals to *'COMPLETED'*, and having non-zero *MaxRSS* and *CPUTimeRAW*. Next, this module standardizes *Timelimit* to numeric hours, *MaxRSS* and *ReqMem* to gigabytes (GB), and *Account* and *QoS* to numeric factors. Finally, *Account*, *ReqMem*, *ReqNodes*, *Timelimit*, *QoS*, *MaxRSS*, and *CPUTimeRAW* are normalized using the `StandardScaler` transform in Scikit-learn Python package[48].

4.4.3 Evaluating individual regression models

Before building the *Mixed Account Regression Models* (MARM), the admin can evaluate individual regression models to note what may be most suited to their dataset. Although optional, the `BuildPerAccModels` module can provide initial insights on the quality of data and can significantly speed up MARM building time by nominating promising regression models for MARM overall possibilities. The `BuildPerAccModels` module requires the admin to provide the path to processed data (`path_to_data`), independent variables or features (`indep_vars`), and a dependent variable (`dep_var`) to train and evaluate seven popular regression models on all data-subsets containing individual *Account*. At this point, the admin can specify the minimum number of jobs an individual *Account* should have in order to be considered (`min_num_jobs`). The seven regression models include: i) Lasso Least Angle Regression (LL)[49,50], ii) Linear Regression (LR)[50], iii) Ridge Regression (RG)[50], iv) Elastic

Net Regression (EN)[50], v) Classification and Regression Trees (DTR)[51], vi) Random Forest Regression, (RFR)[52], and vii) LightGBM (LGBM)[53]. The regression models are evaluated by means of the Coefficient of determination (R^2), and root mean squared error (RMSE)[50]. We used scikit-learn's[48] implementation for all models and performance metrics.

4.4.4 Evaluating mixed account regression models

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

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

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

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

where $X_{A[n]}$ and $Y_{A[n]}$ correspond to independent and dependent variables respectively

for an unique *Account* $A[n]$. Thus, the MARM of N accounts depends upon the MARM of $N-1$ accounts appended with the best overall *Account* N' that results in the best overall performance. R^2 scores $R2_{tr}$ and $R2_{te}$ are calculated by randomly splitting the data into 80% (training) / 20% (testing), five times (5-fold) modeling using the regression model M, and averaging the R^2 scores on training and testing data subsets over the five runs. A comprehensive explanation of the Mixed Account Regression Model (MARM) can be found in our publication[27].

4.4.5 Building MARM for prediction

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

4.4.6 Job resource prediction

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

4.5 Results and Discussion

4.5.1 Preprocessing and PerAccount Models

We applied our ML tool using the HPC resources at Kansas State University, called Beocat. The data side has 17.6 million instances and covers the years 2018 - 2021 of the usage. After using `PreProcessing` module only selecting *'normal'* QoS, the dataset contained 7.8 million jobs spread across 21 unique accounts. Employing `BuildPerAccountModels`, we evaluated all seven regression models across 21 accounts, resulting in **Figures 4.3, 4.4, 4.5 , and 4.6** for predicting time (CPUTimeRAW) and memory (MaxRSS) that shows boxplots of R^2 and negative RMSE score distributions. We found LGBM, DTR, and RFR to be clear winners. Thus, we decided to only utilize LGBM, DTR, and RFR to build MARM.

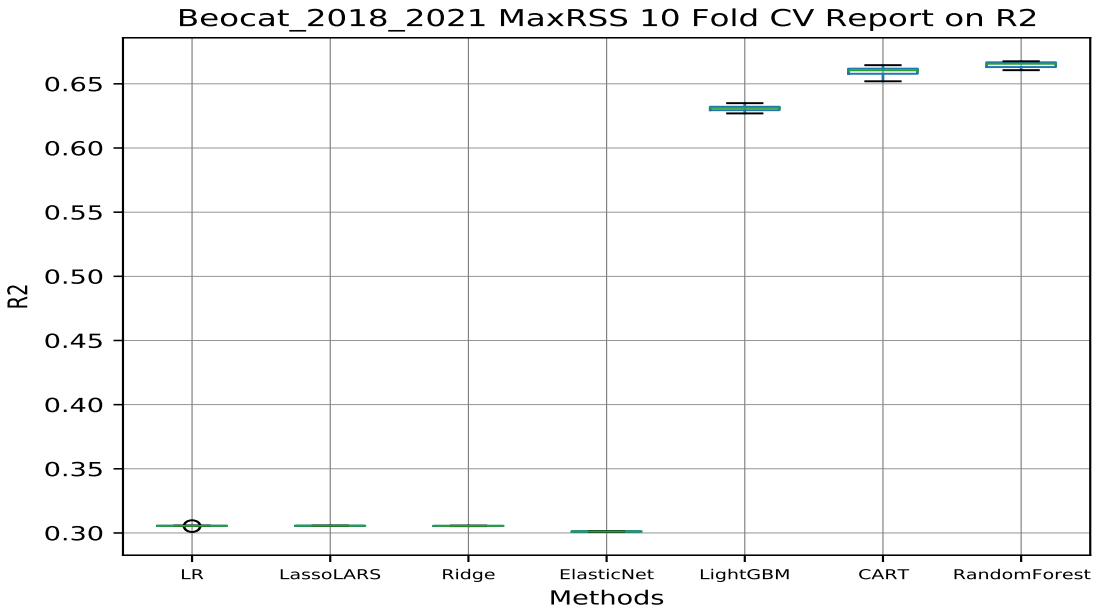

Figure 4.3: *Beocat 2018-2021 MaxRSS 10 Fold CV Report on* R^2

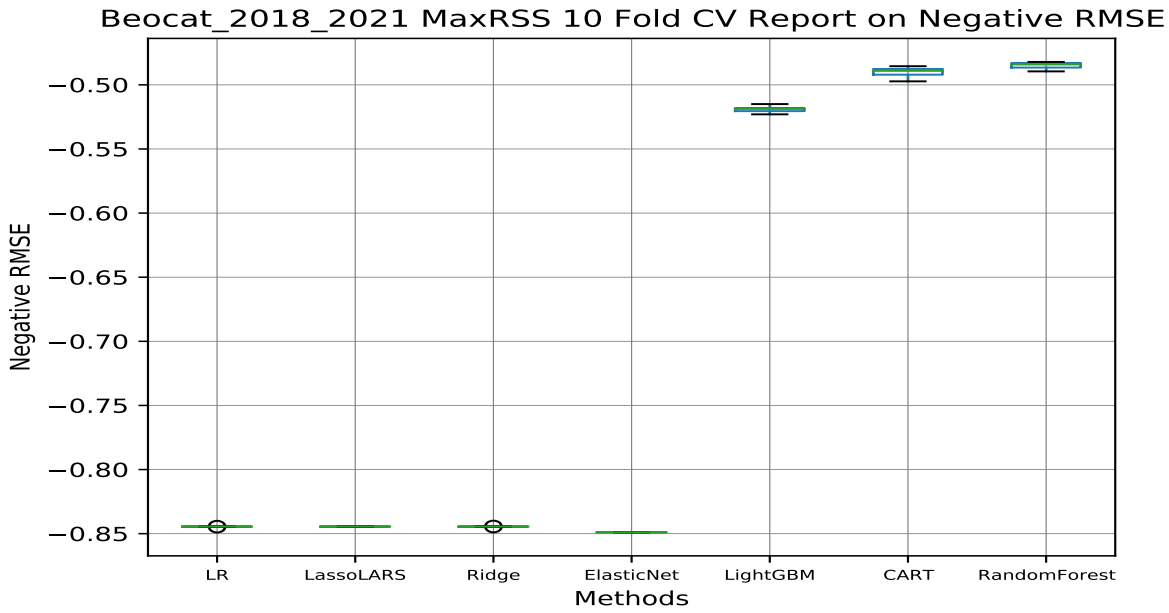

Figure 4.4: *Beocat 2018-2021 MaxRSS 10 Fold CV Report on Negative RMSE*

4.5.2 MARM models in BEOCAT

Utilizing `BuildMixedAccountModels`, we constructed MARMs to predict memory and time in Beocat using 17 out of 21 accounts (80% of the total accounts) in Beocat. **Figures 4.7, 4.8, 4.9, 4.10, 4.11, and 4.12** shows the mean R^2 score distribution of DTR, RFR, and LGBM on training and testing datasets versus the number of best account combinations in predicting time. It can be seen that the R^2 decreases as the number of accounts (and jobs) increases. We found RFR was the best performer in predicting time, while LGBM was the best performer in predicting memory. Thus, we finalized the memory and time MARM using `TrainSelectedMARM` to be i) best five account combination (spanning across 1.8 million jobs) with an average R^2 of 0.74, for building an RFR based time model as shown in **Figures 4.12** and ii) best thirteen accounts combination (spanning across 1.4 million jobs) with average R^2 of 0.72, for building an LGBM based memory model as shown in **Figures 4.8**. Using the finalized MARMs, we randomly sampled 5000 jobs from Beocat and ran them on a Slurm simulator with requested, actual, and predicted time and memory

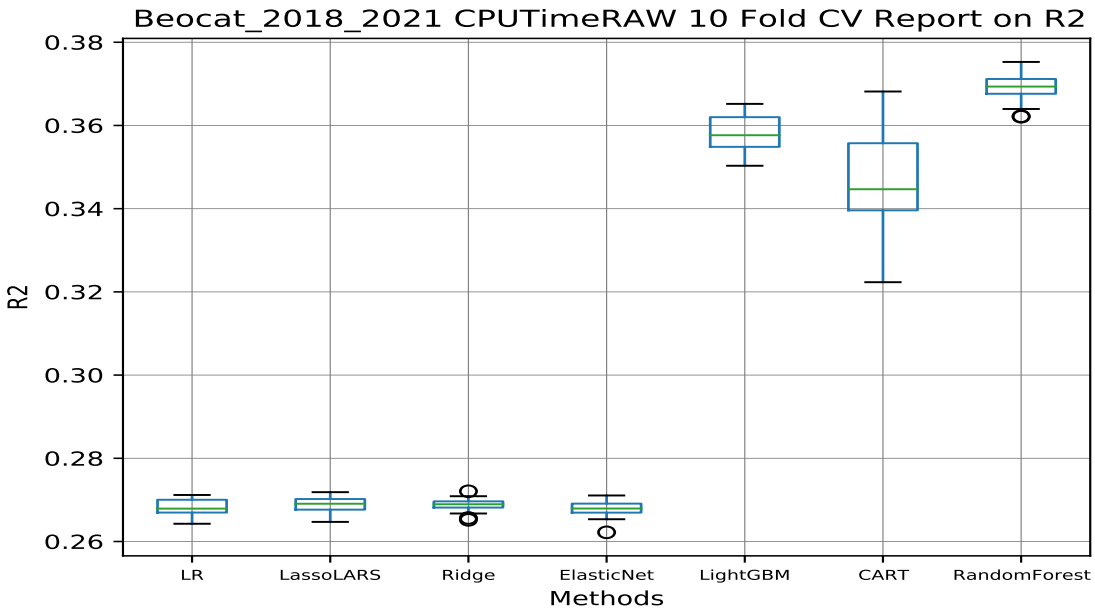

Figure 4.5: *Beocat 2018-2021 CPUTimeRAW 10 Fold CV Report on* R^2

values.

4.5.3 Evaluating Our Model

We assessed our model using the Slurm simulator[66,13], which was developed by the Center for Computational Research, SUNY Buffalo. The Slurm simulator was chosen because it is implemented from a modification of the actual Slurm code while disabling some unnecessary functions, which do not affect the functionality of the real Slurm[66].

Figure 4.13 shows submission and execution time, which indicates the difference between the job submission time (timestamp that represents when the job was submitted) and the execution time (difference between the start and end execution time) for five thousand jobs. Our results indicate that we have achieved almost identical running time compared to the actual running time.

Figure 4.14measures and compares system utilization using requested jobs resources versus actual job resources versus predicted job resources using the AMPRO-HPCC tool.

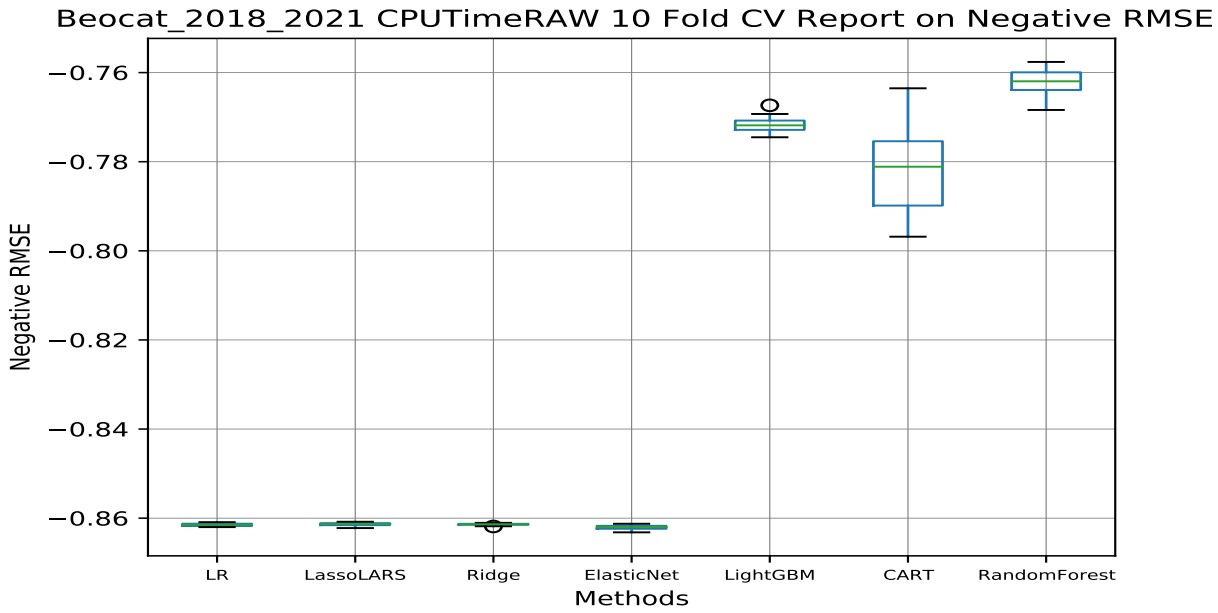

Figure 4.6: *Beocat 2018-2021 CPUTimeRAW 10 Fold CV Report on Negative RMSE*

Our results show that our tool reached almost similar utilization compared to the utilization of the HPC system that used actual job resources because of the high prediction accuracy of our ML tool.

Figure 4.15 compares and assesses the backfill-sched algorithm's performance. The graph shows more efficient performance on the backfill-sched algorithm on the Beocat testbeds that used our ML module than the ones that did not. The graph shows fewer density results when using predicted values since using our AMPRO-HPCC model decreases the number of resources required by the user for the submitted jobs in most cases. This situation results in helping Slurm fit more jobs on the cluster. It also reduces the need to use the backfill-sched algorithm and resulting in more overall system efficiency by using these available resources.

Tables 4.1 and 4.2 provides the calculated average waiting time, and average turn-around time for Beocat jobs for requested, actual, and predicted job resources allocation. Our results show that our tool was able to reduce the average waiting time for submitted

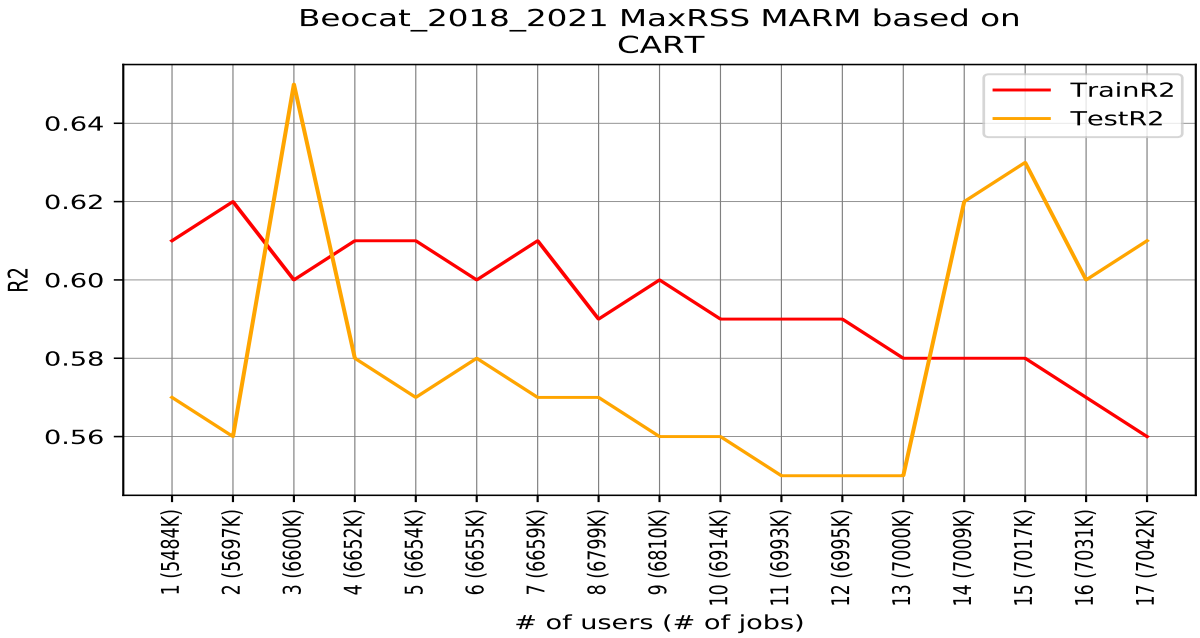

Figure 4.7: *Beocat 2018-2021 MaxRSS MARM based on CART*

	Avg Wait Time (Hour)	Avg TA Time (Hour)
Requested	680 ±128	692.8 ±130
Actual	0.4 ±0.08	3.62 ±1.8
Predicted	8.0±1.1	6.36 ±1.9

Table 4.1: *Average Waiting and Turnaround Time (Requested vs Actual vs Predicted) For Beocat*

jobs from 680 hours to 8.0 hours and the average turnaround time from 692 hours to 16.4 hours.

4.6 Summary

Determining the allocation of HPC resources for submitted jobs is a difficult process for HPC users. It is still an open question how many resources the user should specify (memory and time) for their submitted jobs on the cluster. HPC users are encouraged to overestimate job resources for their submitted jobs. In this paper, we have developed a novel and the

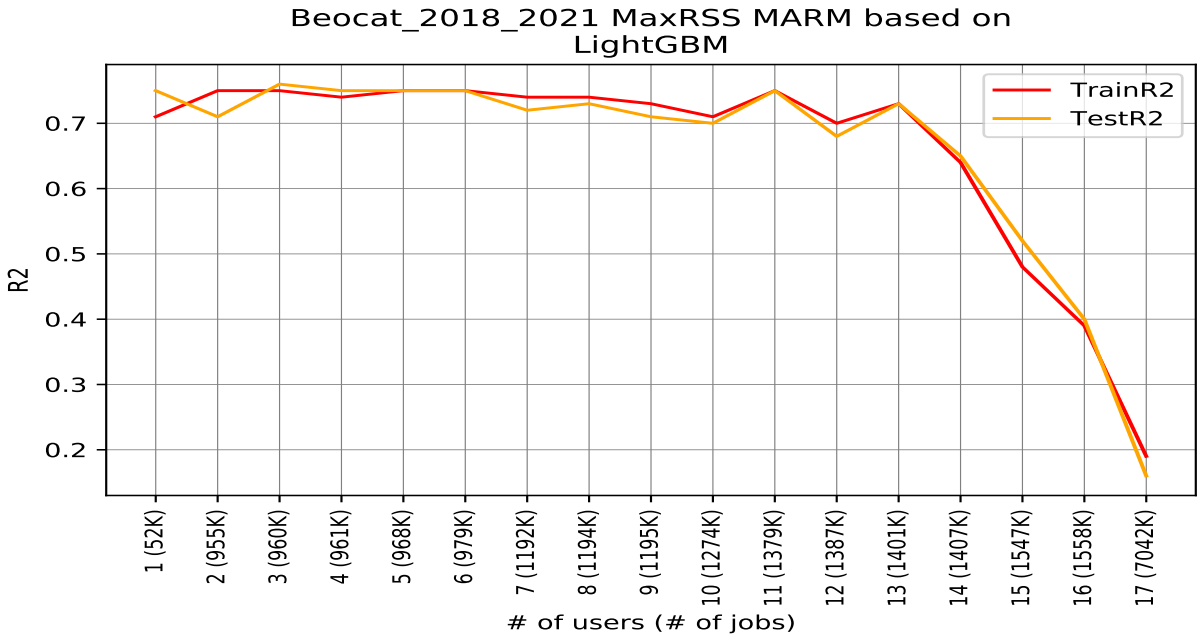

Figure 4.8: *Beocat 2018-2021 MaxRSS MARM based on LightGBM*

	Median Wait Time (Hour)	Median TA Time (Hour)
Requested	713.6	715.6
Actual	0	3.09
Predicted	1.4	5.9

Table 4.2: *Median Waiting and Turnaround Time (Requested vs Actual vs Predicted) For Beocat*

first-ever open-source, stand-alone, fully-automated, highly-accurate, and fully-offline ML tool to help HPC users to determine the amount of required resources (memory and time) for their submitted jobs on the HPC clusters. Our tool was built using supervised ML algorithms. Our tool consists of two parts: i) the system admin part, which is responsible for preparing and building the ML model based on Slurm historical data and providing it to the users; ii) the user part, which uses the ML model provided from the system admin part, reads the submitted job script, and predicts the required amount of the resources (memory and time). Our tool achieves high accuracy and can significantly increase the performance and utilization of the HPC systems. Moreover, our ML tool can dramatically decrease the

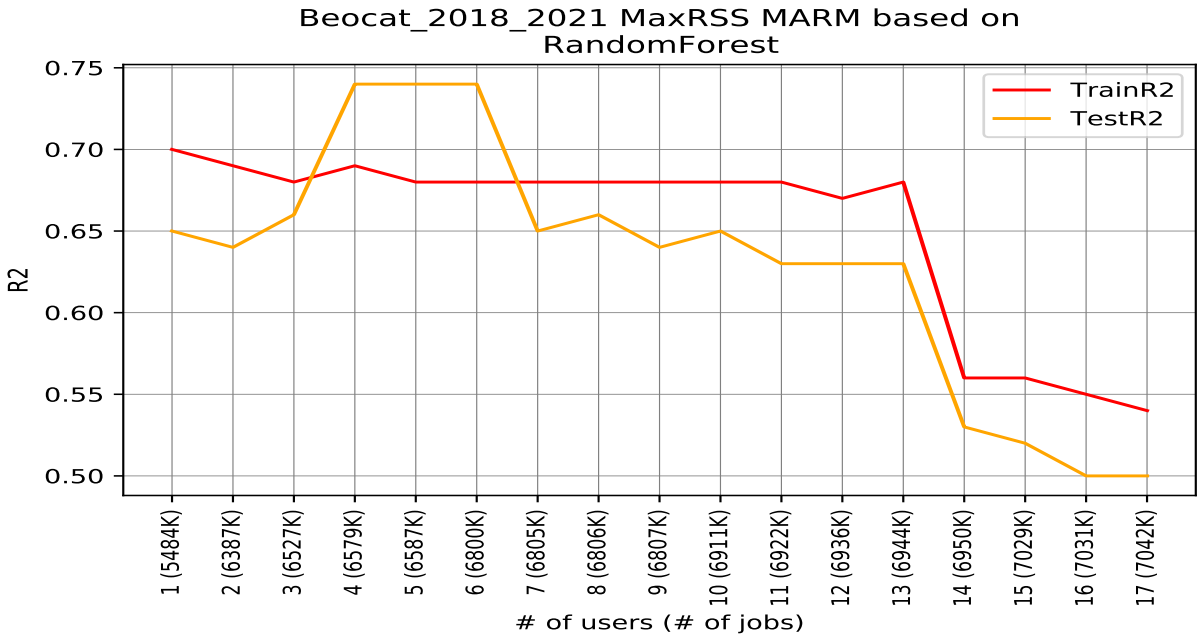

Figure 4.9: *Beocat 2018-2021 MaxRSS MARM based on RandomForest*

average turnaround and waiting time for the submitted jobs. Hence, our tool increases the efficiency and decreases the power consumption of the Slurm-based HPC resources.

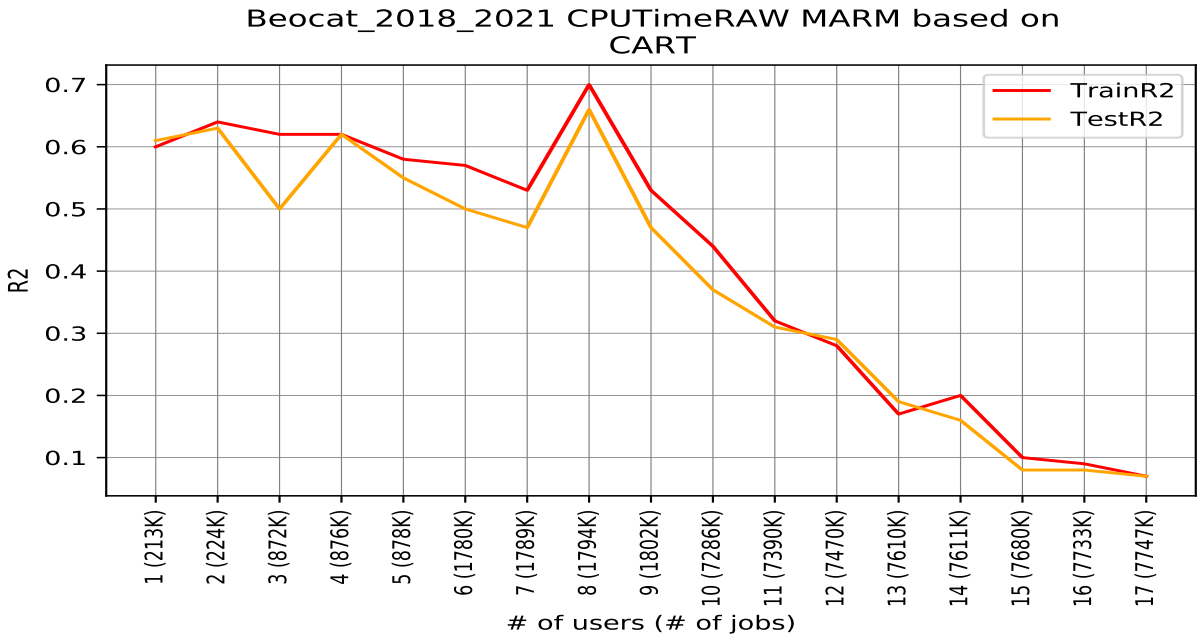

Figure 4.10: *Beocat 2018-2021 CPUTimeRAW MARM based on CART*

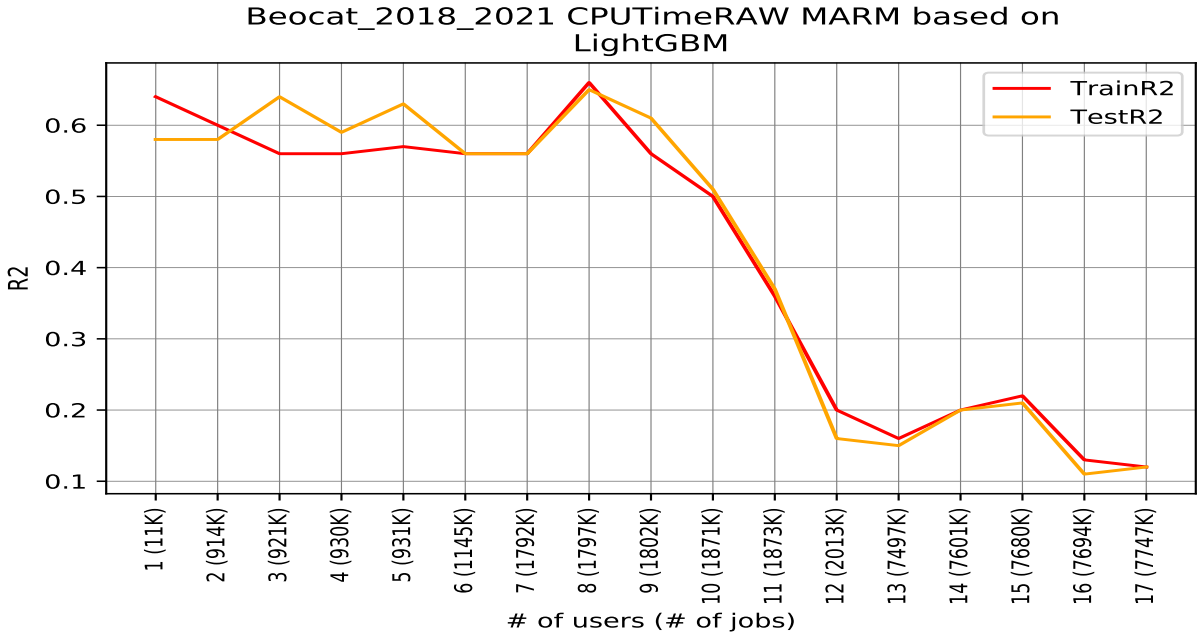

Figure 4.11: *Beocat 2018-2021 CPUTimeRAW MARM based on LightGBM*

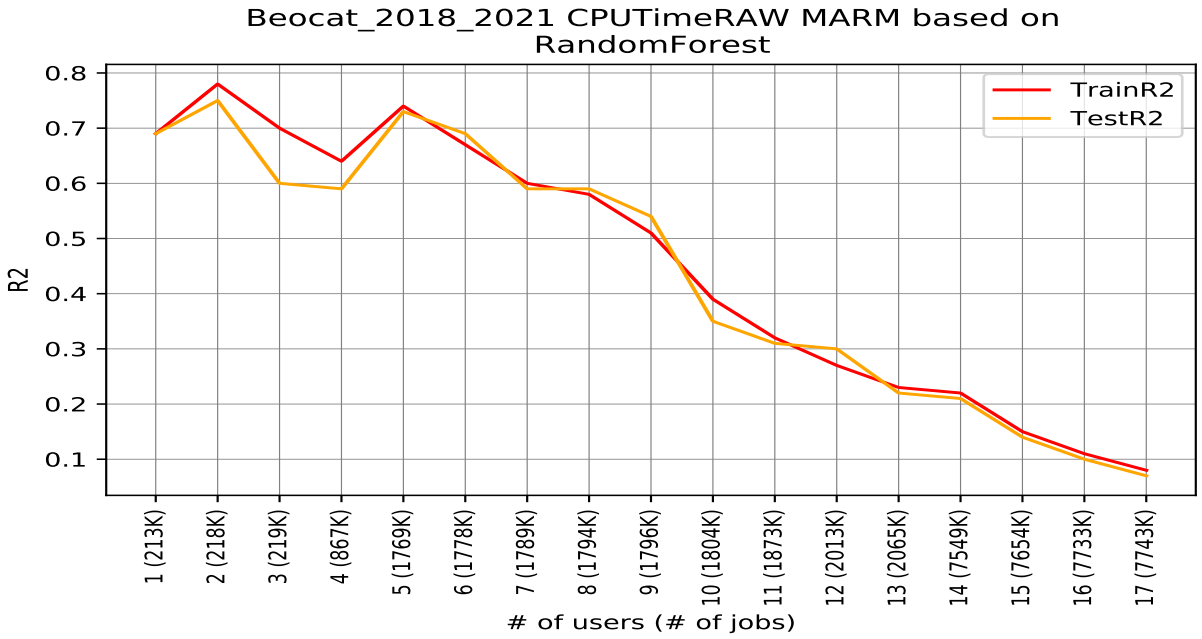

Figure 4.12: *Beocat 2018-2021 CPUTimeRAW MARM based on RandomForest*

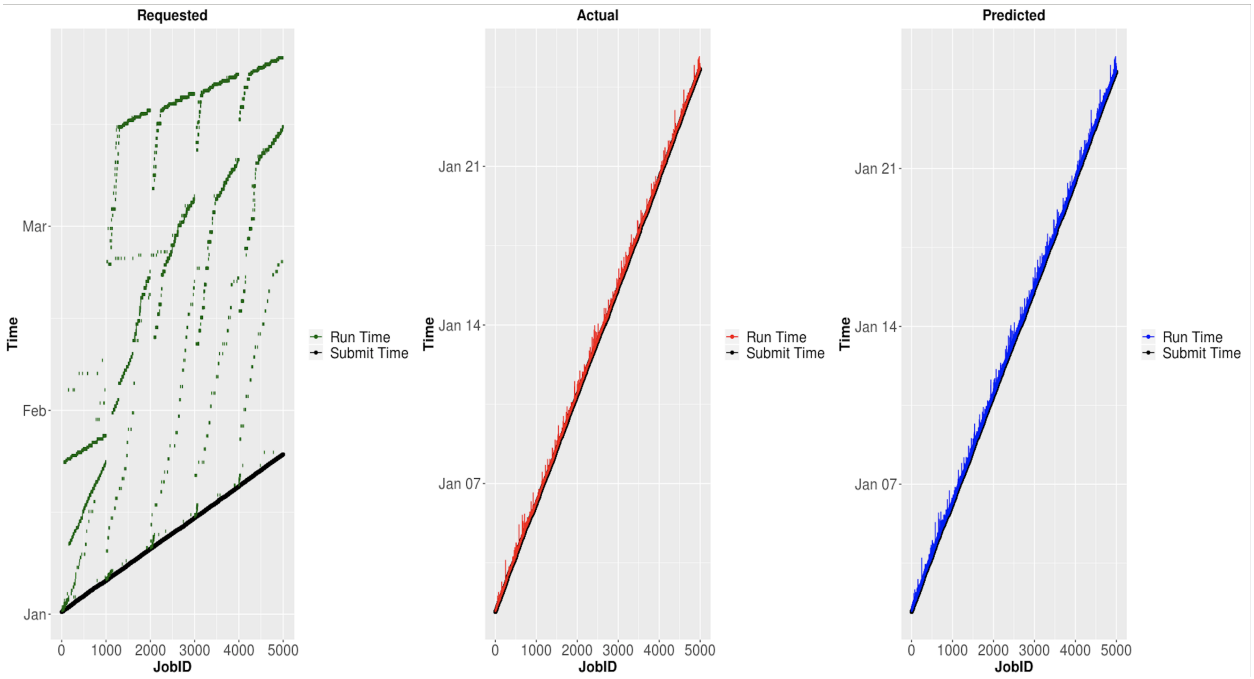

Figure 4.13: *Jobs Submission and Running Time. (Note Dramatic Improvement of Y Axis Range*

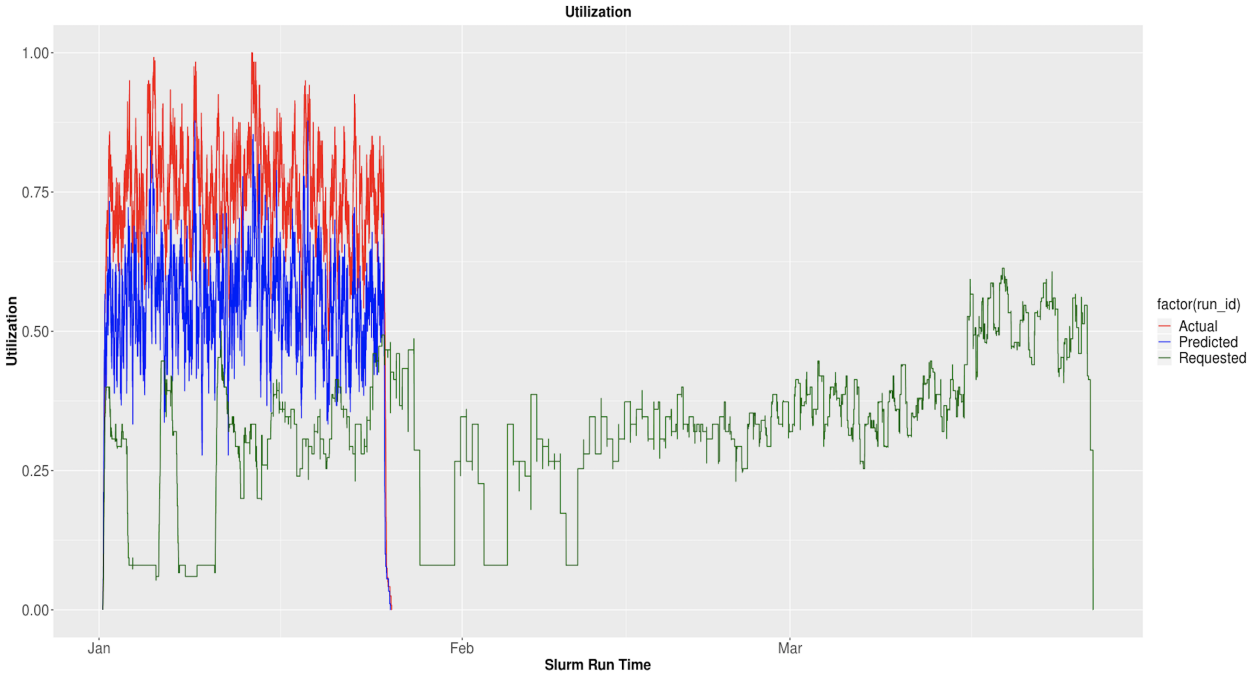

Figure 4.14: *Utilization (Requested vs Actual vs Predicted) for Beocat Jobs*

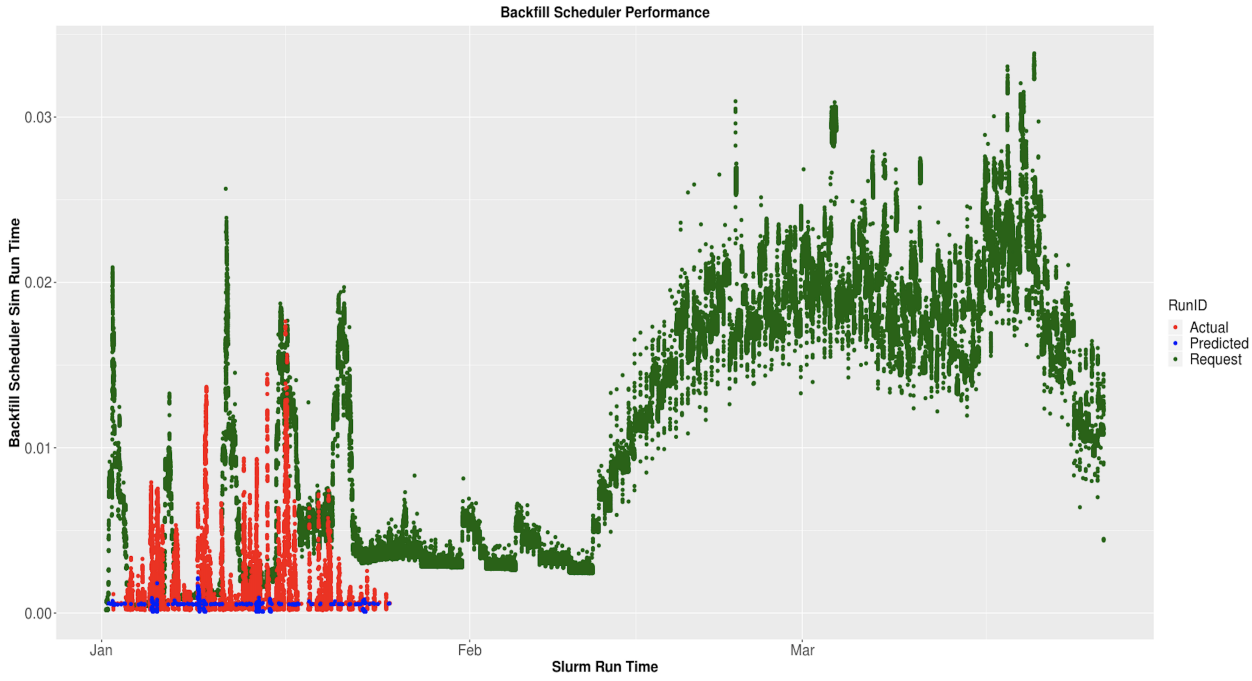

Figure 4.15: *Backfill-Sched Algorithm Performance (Requested vs Actual vs Predicted) for Beocat Jobs*

Chapter 5

Cost-Effective Resource Provisioning of Cloud Computing via Supervised Machine Learning [1]

5.1 Abstract

Cloud computing has become more readily available and users are getting the advantage of the powerful resources and receiving results of running their extensive computations and simulations that require lots of resources in a short period of time. Cloud computing is highly capable because of the powerful hardware provided by cloud service providers such as AWS, Microsoft Azure, Google cloud, and IBM Spectrum Computing. Two of the critical challenges of running jobs on the cloud are: cost-effectiveness and meeting critical deadlines.

In this work, we investigate the impact of using our Machine Learning techniques for predicting job resources (memory and time) in terms of resources and cost provisioning for running HPC jobs on the cloud.

[1] This work will be submitted to Practice and Experience in Advanced Research Computing (PEARC '22), July 10-14, 2022, Boston, MA, USA

We found that HPC users are overestimating job resources needed for their submitted jobs by upwards of 10X, which will highly impact the cost of running their jobs on the cloud. We can assist HPC users by recommending the amount of resources (memory and time) needed for their submitted jobs by predicting the amount of memory and time required for their submitted jobs on the cloud via our proposed first-ever implemented, fully-offline, fully-automated, stand-alone, and open-source machine learning tool called AMPRO-HPCC. We have evaluated our tool by comparing the run time and cost of 4.46 million jobs from 2018 to 2021 from the Kansas State University (BEOCAT), and 2.81 million jobs covered the years 2018-2019 from the University of Colorado-Boulder (RMACC-Summit) HPC resources. We found that our cost-effective Machine Learning tool can reduce the average cost of running jobs on the cloud by up to 39% for the BEOCAT jobs and up to 47% for the RMACC-Summit resources. Moreover, decrease the average running time to meet the deadlines by 39% for the BEOCAT jobs and up to 52% for the RMACC-Summit resources.

5.2 Introduction and Background

Cloud computing service providers such as Amazon Web Services (AWS)[67], Microsoft Azure[68], Google Cloud[69], and IBM Spectrum Computing[70] have been caught the attention in the field of High Performance Computing (HPC) and scientific computing community in the last decade due to availability and competition[71]. At the same time, cloud computing infrastructures are becoming more well known and popular because of the various number of services and different quality of service (QoS) they offer[72][73]. On the other hand, running many of jobs on the cloud can become quite costly, especially when users require significant resources for their submitted jobs. Moreover, cloud users frequently request many more resources than their submitted jobs actually need to avoid potential jobs being terminated due to an insufficient amount of resources requested. Therefore, "one of the most challenging problems with real-time workflows in cloud computing is to get a cost-effective way to

complete the workflow within the deadline" [74]. Thus, HPC users would benefit from getting feedback about determining resource allocations (memory and time) for their submitted jobs on the cloud, such as getting information about whether the amount of resources (memory and time) required of a particular submitted job were not enough, or too much of a particular run. Such feedback requires more information history about actual usage of the resources of previous runs to assess and give feedback regarding required resources for any new runs. This data can be found in the sacct data provided by the Slurm resource manager.

Executing each particular job using cloud provider services is similar to submitting jobs on a local cluster. After an HPC user submits a job with a specific resource requirement, the cloud provider will then assign and reserve the needed computing resources from the resources pool. The job will then be assigned to the best-suited resources in order to be executed. The cost of running a particular job on the cloud depends on the amount of resources the user asked for that job. This means the more resources requested, the more cost for executing the job. On the other hand, the more resources the job requires, the more probability the job will be waiting on the queue for allocating the required resources and vice versa [75].

One of the most important factors of using HPC in the cloud is the cost. The cost to use the HPC cloud depends on many factors such as types of hardware offered, the amount of resources needed (cores, memory, time, etc.), and the capacity of storage needed. While most of the HPC intensive usage users are looking for cost-effective HPC cloud resources, it is still hard and challenging to decide how many resources are needed for a particular submitted job on the cloud. Hence, HPC cloud users usually consume much more resources than needed for their submitted jobs. This process is not cost efficient and can consume significant funding from their budgets and grants.

The basic ideas of our work are: **i)** Helping HPC users estimate and reduce the amount of money and resources needed on the cloud while maintaining the minimum resources needed for their submitted jobs on the cloud. Hence, reduce the budget. **ii)** measure the average

saving budget using our machine learning model published in[27], and[4], versus not using our model for most popular HPC cloud services providers such as (AWS, Azure, Google Cloud, and IBM Spectrum Computing).

To achieve our goals, we calculate the amount of average resources and costs of usage of running millions of jobs from both HPC resources of the University of Colorado Boulder RMACC-Summit and from the Kansas State University Beocat, using actual, requested, and predicted usage generated from our ML tool AMPRO-HPCC[54]. We finally provide the comparison of resource usage and costs for both HPC resources using four well known cloud services providers Amazon Web Services (AWS), Microsoft Azure, Google Cloud, IBM Spectrum Computing, and on on-premises machine.

In this work, we investigate the impact of using our ML tool AMPRO-HPCC[54] in running jobs in the cloud by comparing the cost of running all jobs with and without using our machine learning tool on the most popular cloud computing resources.

Our research focuses on the following research question: **i)** How much resources (memory and time) on average do HPC users overestimate for their submitted jobs? **ii)** How efficient and effective is using our machine learning model for predicting job resources (memory and time) in terms of average cost and resources reduction for running jobs on the cloud, especially the economic impact of predicting job resources?.

5.3 Related Work

Using local HPC resources could be inadequate for application executions in many cases, such as big jobs that request more resources than the available ones on the local cluster. Moreover, big jobs need to wait a long time in a queue[76]. Thus, the ideal solution would be running these resource intensive jobs to the HPC cloud, which is known as HPCaaS (HPC as a Service)[77,78]. While cost-effective resource provisioning in clouds is still a critical challenge[74].

There are studies that focus on when and how to move HPC jobs to the cloud, where it helps HPC users to determine whether jobs need to run on the cloud-based cluster or to run on local clusters[79,80,81]. Another study focuses on evaluating the performance on running applications on the cloud[82,76,83,84].

Eduardo Roloff et al. introduced a comparison of three important factors: deployment, performance, and cost of the cloud compared to the performance of an on-premises machine by running NAS Parallel Benchmarks (NPB)[85] using three different cloud providers: Amazon EC2, Microsoft Azure and Rackspace[86].

Renato Cunha et al. proposed an advisor tool to choose where HPC users should submit their jobs, either cloud or on-premises, based on computing a turnaround time estimate for a certain job[81].

Other areas of research focus on predicting the amount of waiting time in an HPC queue using several techniques[87,88,89,90]. In addition, more effort was made in the area of predicting the amount of execution time on the HPC resources[91,92]. Warren Smith introduced a technique based on instance-based learning[93] and historical information to predict HPC scheduler queue waiting times and execution times for submitted jobs[94].

Researchers also focused on studying and improving job scheduling techniques in the HPC environments[95,96,97,98,99]. While task scheduling is an NP-hard problem, which means that up to now, there is no scheduling algorithm that can achieve an optimal solution within polynomial time[100].

Jiyuan Shi et al. introduced an elastic resource provisioning and task scheduling mechanism to perform scientific workflows in the cloud. Their goal is to complete as many high-priority workflows as possible under budget and deadline constraints. Their techniques consist of three phases: workflow pre-processing, elastic resource provisioning, and task scheduling[101].

Lei Wu et al. proposed a cost optimization algorithm that emphasizes on resource provisioning in order to meet the deadlines of real-time workflow[74].

In the area of cost provisioning for real-time workflow in cloud computing environment, recent studies show that resource provisioning is much capable and successful than task scheduling[100].

Verma and Kaushal introduced a heuristic that benefits trade-off between deadline and budget under given constraints. Their proposed constrained heuristic is based on Heterogeneous Earliest Finish Time (HEFT) to schedule workflow tasks over the available cloud resources[102].

Wei Zheng proposed a variety of algorithms to help minimize the monetary cost of running big jobs on the cloud with deadline constraints to a satisfactory level. Their proposed work uses separate CPU frequency for each task to reduce the overall user cost[103].

Our work focuses on studying the effects of using machine learning techniques provided in our work[27,57], and conducting a detailed comparison of resource usage and costs of running jobs on the cloud. We mainly study the benefits of using our ML techniques in terms of saving resources usage and costs on running jobs on the cloud providers such as AWS, Azure, Google Cloud, and IBM Spectrum Computing.

5.4 Implementation

In this work, we focus on resource provisioning in cloud computing using our proposed ML tool AMPRO-HPCC provided in GitHub[57], which uses our ML Mixed Account Regression Model (MARM), explained in detail in[4], that helps and recommends the HPC users for predicting the amount of resources needed (memory and time) for their submitted jobs.

To be able to perform and implement our work, the workflow process includes the following four stages:

Stage 1: Collecting the HPC log data (sacct data)

Two data sets (sacct data) were collected from the Slurm workload manager database as the following:

- HPC resources of the XSEDE service provider at the University of Colorado Boulder (RMACC-Summit)[46]: The data set has **7.8 million** instances and covers the years from 2018 – 2021 of the usage.
- HPC resources of the Kansas State University (Beocat)[22]. The data side has **10.9 million** instances and covers the years 2018-2019.

The collected data include the required features for the time requested set for each job (Timelimit), actual time usage for running each job (CPUTimeRAW), Minimum required memory (ReqMem), Maximum resident set the size of all tasks in each job (MaxRSS), State of the job (State), accounts information, name of Quality of Service (QoS), etc. We need that information in order to calculate the cost and build our MARM ML model to predict the amount of resources for each newly submitted job.

Stage 2: Data Cleaning and Filtration

At this stage, we prepare the sacct data by removing all certain jobs associated with missing values (NaN) associated with features MaxRSS or CPUTimeRAW. We replaced missing values of Timelimit, Partition, and QoS with default values. We consider all completed jobs only, therefore, jobs with incomplete State ('Cancelled', 'Failed', 'Deadline', etc.) were removed. At the end of this stage, we ended up having **4.46 million** jobs left in Beocat, while RMACC-Summit had **2.81 million** jobs left.

Stage 3: Resources Prediction

At this stage, we calculate the predicted amount of resources (memory and time) using our ML tool AMPRO-HPCC that uses our MARM methodology. So, we can use the predicted

values to calculate the accuracy of our ML model, the amount of resources provisioning, and the cost-effective resource provisioning in the cloud.

Stage 4: Calculate HPC resources needed

At this stage, we extract the amount of resources required (memory and time) for each job in both Beocat and RMACC-Summit resources from the cleaned data and calculate the average usage of memory and time for all jobs for each HPC resource as the following:

- Calculate the amount of resources (memory and time) required for each job using the amount of requested resources provided by the user (ReqMem, and Timelimit). Hence, we can calculate the total and average amount of resources using the requested usage of all completed jobs for each HPC resource.
- Calculate the amount of resources (memory and time) required for each job using the amount of actual usage of resources provided from the Slurm workload manager (MaxRSS, and CPUTimeRAW). Hence, we can calculate the total and average amount of resources used by all of the completed jobs for each HPC resource.
- Calculate the amount of resources (memory and time) required for each job using the amount of predicted usage of resources provided from our ML tool AMPRO-HPCC that uses our MARM methodology. Hence, we can calculate the total and average amount of resources using the predicted usage of all completed jobs for each HPC resource.

5.4.1 Calculate the Cost of Running Jobs on the cloud

Our study will provide analytical comparison and calculation for the cost of running all successfully completed jobs for both Beocat and RMACC-Summit resources using multiple cloud service providers (Amazon Web Services, Google Cloud, Microsoft Azure, Digital Ocean, IBM Cloud, and Holland Computing Center)

Cores	RAM (GB)	Instance name	Hourly Cost	Seats	Cost Per Seat
2	8	e2-standard-2	0.07	2	0.0350
4	16	e2-standard-4	0.13	4	0.0325
8	32	e2-standard-8	0.27	8	0.0337
16	64	e2-standard-16	0.54	16	0.0337
32	128	e2-standard-16	1.07	32	0.0334

Table 5.1: *Google Cloud Platform Cost for 1 Core / 4 GB Seat*

Cores	RAM (GB)	Instance name	Hourly Cost	Seats	Cost Per Seat
2	8	e2-standard-2	0.09	2	0.045
4	16	e2-standard-4	0.18	4	0.045
8	32	e2-standard-8	0.36	8	0.045
16	64	e2-standard-16	0.72	16	0.045

Table 5.2: *Google Cloud Platform Cost for 1 Core / 8 GB Seat*

We have used the Seat Pricing model to estimate the cost of a computing job. A seat will be used as a portion of a compute node, either with 1 CPU Core and 4 GB of RAM or 1 CPU Core and 8 GB of RAM as the unit. For example, an HPC job requesting 4 CPU cores and 20GB of RAM would require a minimum of 4 "seats". The RAM requirement could fit in three 8 GB seats or five 4 GB seats. Whichever of the two combinations is the lowest will be the price used as the estimated cost of the job being analyzed.

Tables 5.1 and 5.2 describes the cost for the calculated pricing using Google Cloud Platform[104]. **Tables 5.3 and 5.4** describes the cost for the calculated pricing using Microsoft Azure[105]. **Tables 5.5 and 5.6** describes the cost for the calculated pricing using Digital Ocean[106]. **Tables 5.7 and 5.8** describes the cost for the calculated pricing using IBM Cloud[107]. **Tables 5.9 and 5.10** describes the cost for the calculated pricing using Amazon Web Services (AWS)[108].

While we chose Holland Computing Center at the University of Nebraska-Lincoln as an example to calculate the cost for using the local resource. **Tables 5.11 and 5.12** describes the cost for the calculated pricing using Holland Computing Center at the University of Nebraska - Lincoln[109].

Cores	RAM (GB)	Instance name	Hourly Cost	Seats	Cost Per Seat
2	8	D2 v4	0.096	2	0.048
4	16	D4 v4	0.192	4	0.048
8	32	D8 v4	0.384	8	0.048
16	64	D16 v4	0.768	16	0.048
32	128	D32 v4	1.536	32	0.048
48	192	D48 v4	2.304	48	0.048
64	256	D64 v4	3.072	64	0.048

Table 5.3: *Microsoft Azure Cost for 1 Core / 4 GB Seat*

Cores	RAM (GB)	Instance name	Hourly Cost	Seats	Cost Per Seat
2	16.	E2 v5	0.063	2	0.0315
4	32	E4 v5	0.126	4	0.0315
8	64	E8 v5	0.252	8	0.0315
16	128	E16 v5	0.504	16	0.0315
20	160	E20 v5	0.630	20	0.0315
32	256	E32 v5	1.008	32	0.0315
48	384	E48 v5	1.512	48	0.0315
64	512	E64 v5	2.016	64	0.0315
96	672	E96 v5	3.024	96	0.0315

Table 5.4: *Microsoft Azure Cost for 1 Core / 8 GB Seat*

Cores	RAM (GB)	Instance name	Hourly Cost	Seats	Cost Per Seat
2	8	General Purpose	0.08929	2	0.0446
4	16	General Purpose	0.17857	4	0.0446
8	32	General Purpose	0.35714	8	0.0446
16	64	General Purpose	0.71429	16	0.0446
32	128	General Purpose	1.42857	32	0.0446
40	192	General Purpose	1.78571	40	0.0446

Table 5.5: *Digital Ocean Cost for 1 Core / 4 GB Seat*

Cores	RAM (GB)	Instance name	Hourly Cost	Seats	Cost Per Seat
2	16.	Memory Optimized	0.11905	2	0.0595
4	32	Memory Optimized	0.23810	4	0.0595
8	64	Memory Optimized	0.47619	8	0.0595
16	128	Memory Optimized	0.95238	16	0.0595
24	160	Memory Optimized	1.42857	24	0.0595
32	256	Memory Optimized	1.90476	32	0.0595

Table 5.6: *Digital Ocean Cost for 1 Core / 8 GB Seat*

Cores	RAM (GB)	Instance name	Hourly Cost	Seats	Cost Per Seat
2	8	bx2-2x8	0.096	2	0.048
4	16	bx2-4x16	0.192	4	0.048
8	32	bx2-8x32	0.384	8	0.048
16	64	bx2-16x64	0.768	16	0.048
32	128	bx2-32x128	1.536	32	0.048
48	192	bx2-48x192	2.305	48	0.048
64	256	bx2-64x256	3.073	64	0.048
96	384	bx2-96x384	4.609	96	0.048
128	512	bx2-128x512	6.146	128	0.048

Table 5.7: *IBM Cloud Cost for 1 Core / 4 GB Seat*

Cores	RAM (GB)	Instance name	Hourly Cost	Seats	Cost Per Seat
2	16.	mx2-2x16	0.124	2	0.0620
4	32	mx2-4x32	0.248	4	0.0620
8	64	mx2-8x64	0.497	8	0.0621
16	128	mx2-16x128	0.994	16	0.0621
32	160	mx2-32x256	1.987	32	0.0621
48	256	mx2-48x384	2.981	48	0.0621
64	384	mx2-64x512	3.974	64	0.0621
96	512	mx2-96x768	5.961	96	0.0621
128	1024	mx2-128x1024	7.949	128	0.0621

Table 5.8: *IBM Cloud Cost for 1 Core / 8 GB Seat*

Cores	RAM (GB)	Instance name	Hourly Cost	Seats	Cost Per Seat
2	8	t4g.large	0.0672	2	0.0336
4	16	t4g.xlarge	0.1344	4	0.0336
8	32	t4g.2xlarge	0.2688	8	0.0336
16	64	m6g.4xlarge	0.616	16	0.0385
32	128	m6g.8xlarge	1.232	32	0.0385
48	192	m6g.12xlarge	1.848	48	0.0385
64	256	m6g.16xlarge	2.464	64	0.0385
96	384	m5a.24xlarge	4.128	96	0.0430
128	512	m6i.32xlarge	6.144	128	0.0480

Table 5.9: *Amazon Web Services Cost for 1 Core / 4 GB Seat*

Cores	RAM (GB)	Instance name	Hourly Cost	Seats	Cost Per Seat
2	16.	r6g.large	0.1008	2	0.0504
4	32	r6g.xlarge	0.2016	4	0.0504
8	64	r6g.2xlarge	0.4032	8	0.0504
16	128	r6g.4xlarge	0.8064	16	0.0504
32	256	r6g.8xlarge	1.6128	32	0.0504
48	384	r6g.12xlarge	2.4192	48	0.0504
64	512	r6g.metal	3.2256	64	0.0504
96	768	r5a.24xlarge	5.424	96	0.0565

Table 5.10: *Amazon Web Services Cost for 1 Core / 8 GB Seat*

Cores	RAM (GB)	Instance name	Yearly Cost	Hourly Cost	Seats	Cost Per Seat
16	64	Crane	633	0.0723	16	0.0045

Table 5.11: *Holland Computing Center Cost for 1 Core / 4 GB Seat*

Cores	RAM (GB)	Instance name	Yearly Cost	Hourly Cost	Seats	Cost Per Seat
36	256	CraneOPA	1967	0.2245	32	0.0070

Table 5.12: *Holland Computing Center Cost for 1 Core / 8 GB Seat*

5.5 Results

5.5.1 RMACC-Summit

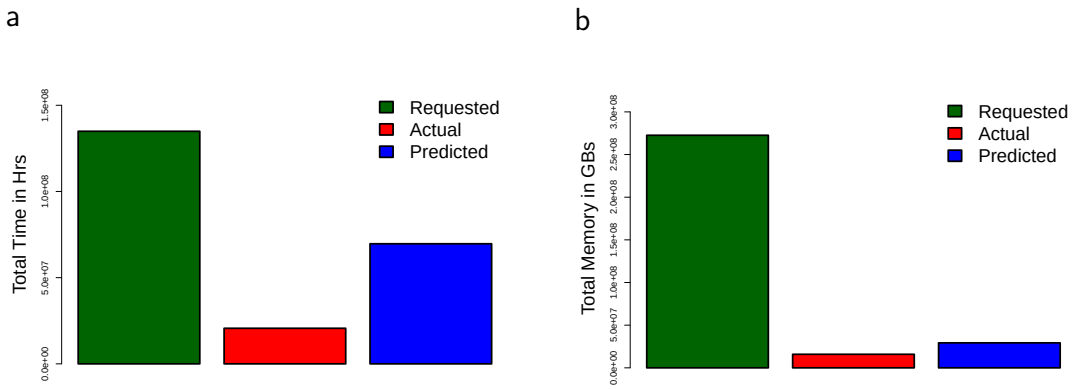

Figure 5.1: *(a) Total aggregate execution time requested (green), used (red), and predicted (blue) in hours and (b) Total aggregate memory requested (green), used (red), and predicted (blue) in gigabytes on RMACC-Summit HPC system across 2.8 million jobs.*

Figures 5.1a and b show the total aggregate execution time and memory requested, used, and predicted by our MARM algorithm on 2.8 million job logs obtained from RMACC-Summit. It can be seen that the requested time and memory are significantly higher than the actual and time and memory used. Our predicted time and memory are both closer to the actual than requested configurations.

Figures 5.2 (a) to (f) show the logarithm of cost distribution across six cloud platforms using requested, actual, and predicted configurations of time and memory. These statistics were obtained on the 2.8 million job logs from RMACC-Summit. In all cases, the cost of running services with the requested configuration is significantly higher than the actual cost. Our MARM predicted configurations incur smaller costs that are close to the actual costs.

Figures 5.3 shows the mean cost distribution across various cloud platforms when using requested, actual, and predicted time and memory configurations. Holland Computing Center costs the least amount of money. The mean cost of other services is seen in the following increasing order: i) Google Cloud Platform, ii) Amazon Web Services, iii) Digital

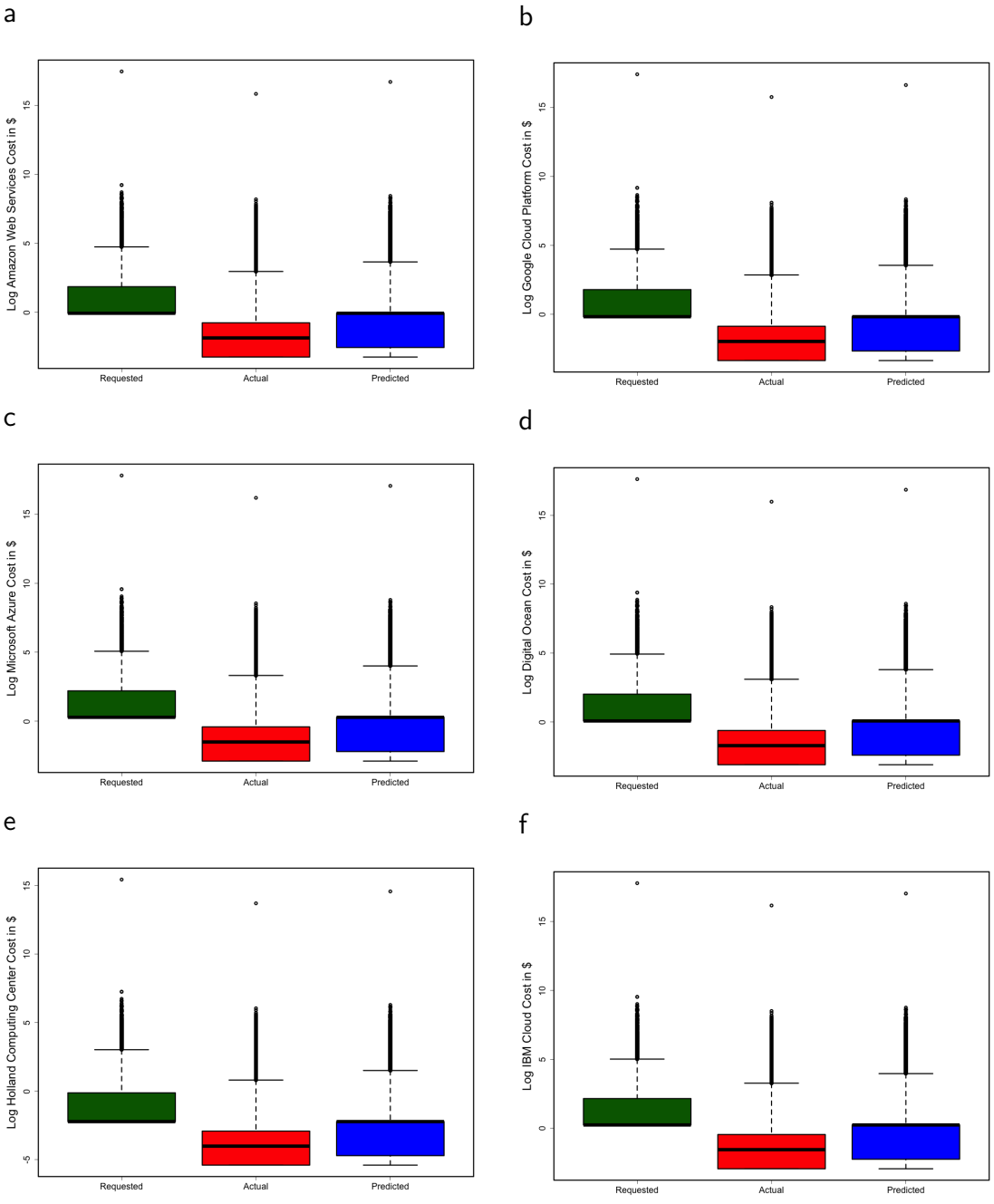

Figure 5.2: *Logarithm of cost distribution incurred in dollars for running 2.8 million jobs with requested (green), actual (red), and predicted (blue) configurations for execution time and memory on (a) Amazon Web Services, (b) Google Cloud Platform, (c) Microsoft Azure, (d) Digital Ocean, (e) Holland Computing Center, and (f) IBM Cloud.*

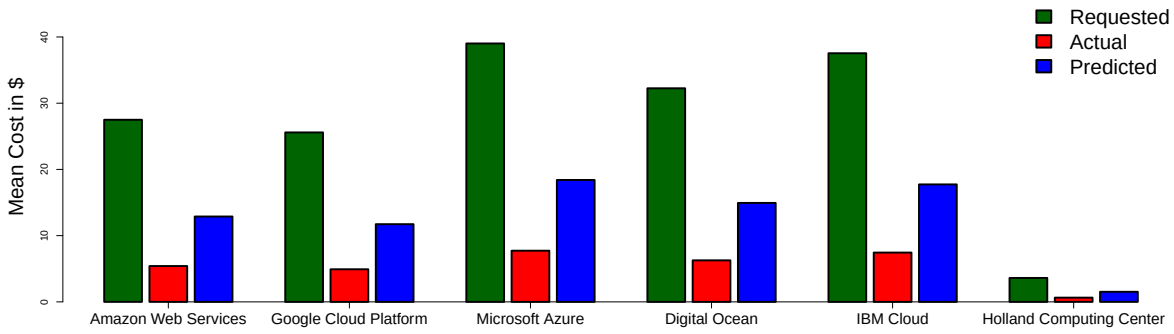

Figure 5.3: *Mean cost incurred in dollars for running 2.8 million jobs across various cloud platform with requested (green), actual (red), and predicted (blue) configurations.*

Ocean, iv) IBM Cloud, and v) Microsoft Azure. The cost distribution changes in magnitude, being the highest for requested configuration followed by predicted configuration, the lowest being the actual configuration. However, the cost distribution itself does not change among cloud platforms.

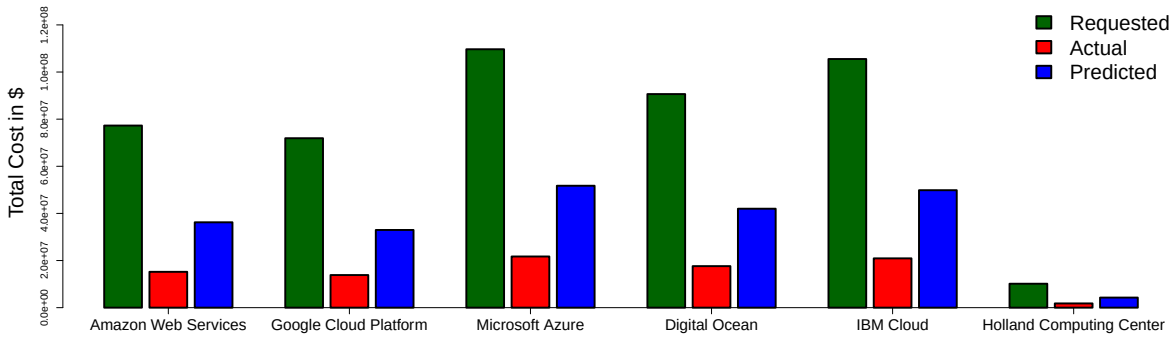

Figure 5.4: *Total aggregate cost incurred in dollars for running 2.8 million jobs across various cloud platform with requested (green), actual (red), and predicted (blue) configurations.*

Figure 5.4 and table 5.13 show some more cost related statistics across cloud platforms, where, consistently, the cost of MARM predicted configuration is smaller than the requested configuration.

			Requested	
	Mean	Median	75th Quantile	95th Quantile
Amazon Web Services	27.50$	0.92$	6.35$	29.03$
Google Cloud Platform	25.58$	0.84$	5.99$	27.36$
Microsoft Azure	39.02$	1.32$	8.98$	41.04$
Digital Ocean	32.25$	1.07$	7.50$	34.28$
IBM Cloud	37.54$	1.27$	8.63$	39.46$
Holland Computing Center	3.61$	0.11$	0.88$	4.04$
			Actual	
	Mean	Median	75th Quantile	95th Quantile
Amazon Web Services	5.41$	0.15$	0.46$	4.62$
Google Cloud Platform	4.92$	0.14$	0.42$	4.20$
Microsoft Azure	7.73$	0.22$	0.66$	6.60$
Digital Ocean	6.27$	0.18$	0.54$	5.35$
IBM Cloud	7.44$	0.21$	0.64$	6.36$
Holland Computing Center	0.64$	0.02$	0.05$	0.54$
			Predicted	
	Mean	Median	75th Quantile	95th Quantile
Amazon Web Services	12.89$	0.92$	3.22$	22.18$
Google Cloud Platform	11.73$	0.84$	2.92$	20.16$
Microsoft Azure	18.41$	1.32$	1.32$	31.68$
Digital Ocean	14.94$	1.07$	2.17$	25.69$
IBM Cloud	17.74$	1.2$	3.12$	30.53$
Holland Computing Center	1.52$	0.11$	1.04$	2.60$

Table 5.13: *Mean, median, 75th-quantile, and 95th-quantile cost incurred across various cloud platforms when using requested, actual, and predicted configurations of time and memory.*

5.5.2 Beocat

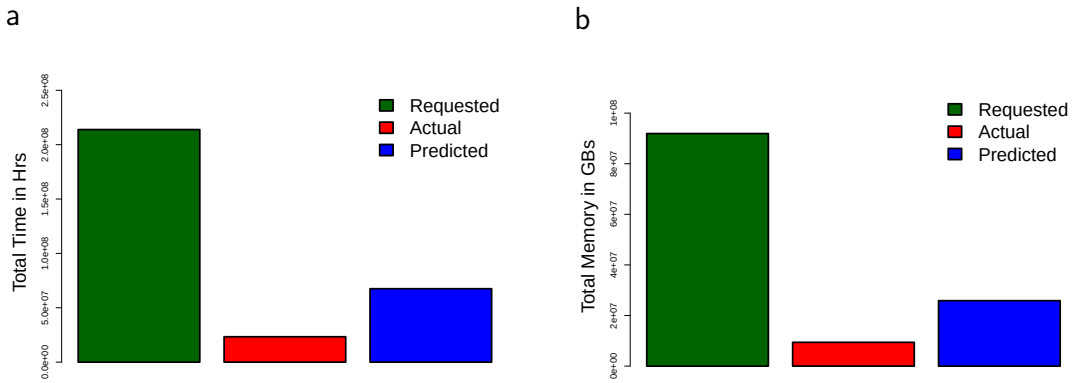

Figure 5.5: *(a) Total aggregate execution time requested (green), used (red), and predicted (blue) in hours and (b) Total aggregate memory requested (green), used (red), and predicted (blue) in gigabytes on Beocat HPC system across 4.5 million jobs.*

Figures 5.5 (a) and (b) show the total aggregate execution time and memory requested, used and predicted by our MARM algorithm on 4.5 million job logs obtained from Beocat. Similar to RMACC-Summit, the requested time and memory is significantly higher than the actual and time and memory used. Our predicted time and memory are both closer to the actual configurations.

Figures 5.6 (a) to (f) show the logarithm of cost distribution across six cloud platforms using requested, actual and predicted configurations of time and memory. These statistics were obtained on the 4.5 million job logs from Beocat. Similar RMACC-Summit, the cost of running services with requested configuration is significantly higher than actual cost in all cases. Our MARM predicted configurations incur smaller costs that are close to the actual costs.

Figures 5.7 shows the mean cost distribution across various cloud platforms when using requested, actual, and predicted time and memory configurations. The cost distributions are comparable for RMACC-Summit with the Holland Computing Center costing the least amount of money and other services costing in the order as aforementioned for RMACC-Summit. Similarly, the cost distribution changes in magnitude, being the highest

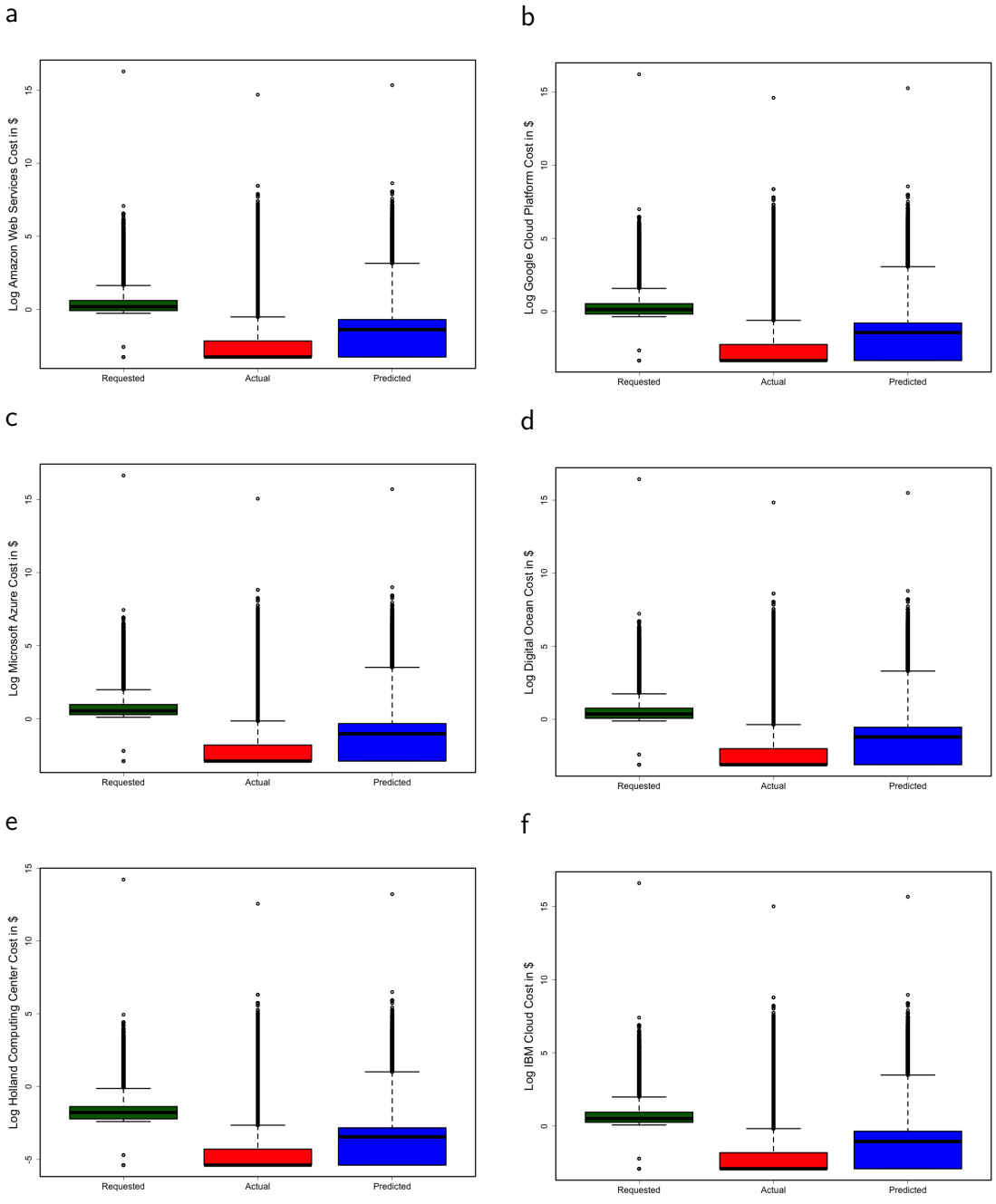

Figure 5.6: *Logarithm of cost distribution incurred in dollars for running 4.5 million jobs with requested (green), used (red), and predicted (blue) configurations for execution time and memory on (a) Amazon Web Services, (b) Google Cloud Platform, (c) Microsoft Azure, (d) Digital Ocean, (e) Holland Computing Center, and (f) IBM Cloud.*

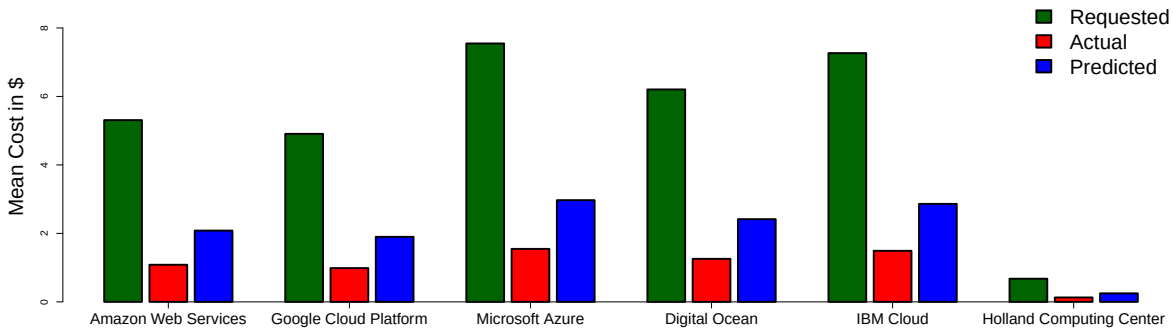

Figure 5.7: *Mean cost incurred in dollars for running 4.5 million jobs across various cloud platform with requested (green), used (red), and predicted (blue) configurations.*

for requested configuration followed by predicted configuration, the lowest being the actual configuration.

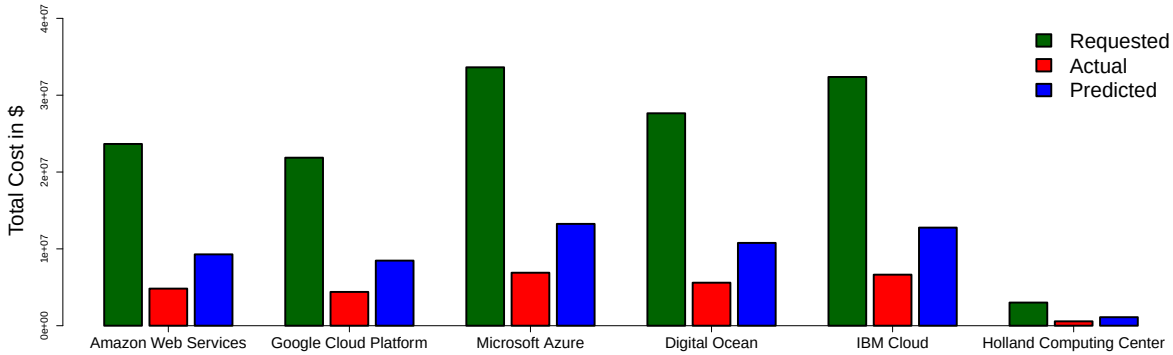

Figure 5.8: *Total aggregate cost incurred in dollars for running 4.5 million jobs across various cloud platform with requested (green), used (red), and predicted (blue) configurations.*

Figure 5.8 and table 5.14 show some more cost related statistics across cloud platforms, where, similar to RMACC-Summit, consistently, the cost of MARM predicted configuration is smaller than the requested configuration.

	Mean	Median	75th Quantile	95th Quantile
			Requested	
Amazon Web Services	5.31$	1.21$	1.85$	10.58$
Google Cloud Platform	4.91$	1.14$	1.71$	9.97$
Microsoft Azure	7.55$	1.71$	2.64$	14.96$
Digital Ocean	6.20$	1.43$	2.14$	12.50$
IBM Cloud	7.27$	1.64$	2.54$	14.39$
Holland Computing Center	0.68$	0.17$	0.25$	1.47$
			Actual	
	Mean	Median	75th Quantile	95th Quantile
Amazon Web Services	1.08$	0.04$	0.12$	0.77$
Google Cloud Platform	0.99$	0.04$	0.10$	0.70$
Microsoft Azure	1.55$	0.06$	0.17$	1.10$
Digital Ocean	1.26$	0.04$	0.13$	0.89$
IBM Cloud	1.49$	0.05$	0.16$	1.06$
Holland Computing Center	0.13$	0.00$	0.01$	0.09$
			Predicted	
	Mean	Median	75th Quantile	95th Quantile
Amazon Web Services	2.08$	0.25$	0.50$	2.04$
Google Cloud Platform	1.90$	0.24$	0.46$	1.89$
Microsoft Azure	2.97$	0.36$	0.71$	2.92$
Digital Ocean	2.42$	0.30$	0.58$	2.38$
IBM Cloud	2.86$	0.34$	0.69$	2.81$
Holland Computing Center	0.25$	0.03$	0.06$	0.25$

Table 5.14: *Mean, median, 75th-quantile, and 95th-quantile cost incurred across various cloud platforms when using requested, actual, and predicted configurations of time and memory.*

5.6 Summary

High Performance Computing and cloud computing has become more and more available over recent history. The performance and availability of these resources have allowed researchers to accelerate their research, using a large quantity of resources in a short period of time to meet their deadlines. Cloud computing services such as AWS, Microsoft Azure, Google cloud, and IBM Spectrum Computing have helped further increase availability by sacrificing a more considerable financial cost to projects and researchers. This study further verified the machine learning models, AMPRO-HPCC and MARM, to help use both financial and computational resources more efficiently by better predicting the resources required for HPC jobs, reducing the overall cost of running HPC jobs on the cloud anywhere from 39% to 47%. This provides a great benefit to researchers by saving their budget on resources and allowing researchers to more efficiently reach their deadlines quicker. This study also demonstrated the financial and computational cost of the overestimation of time and memory resources by HPC users on a project or researcher's budget and the impact it can have on HPC systems.

Chapter 6

Conclusion and future work

6.1 Summary

This thesis summarizes the author's contributions in improving the performance of HPC systems by presenting novel methodologies for predicting job resources (memory and time) for submitted jobs on HPC systems based on historical jobs data provided from the HPC systems scheduler.

Our work involves using several supervised machine learning discriminative models from the scikit-learn machine learning library and LightGBM applied on historical data from Simple Linux Utility for Resource Management (Slurm Workload Manager) and Sun Grid Engine (SGE).

Our work has been implemented and tested using two HPC providers, an XSEDE service provider at the University of Colorado-Boulder (RMACC-Summit) and the Kansas State University (Beocat).

Our methodologies achieved high accuracy (up to 86 %) in predicting the amount of time and the amount of memory for both RMACC-Summit and Beocat HPC resources. Moreover, 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),

reduces turnaround time (from 403 to 6 hours in RMACC-Summit and from 673 hours to 35 hours in Beocat), and achieves high utilization (up to 100 %), higher throughput and efficiency for HPC resources.

Moreover, we introduced our first-ever implemented fully-offline, fully automated, standalone, and open-source Machine Learning tool to help HPC users predict job resources requirements for their submitted jobs on HPC Clusters. We have been tested our AMPRO-HPCC tool using historical data (saact data) of the HPC resources of Kansas State University (Beocat), which covers years from January 2019 - March 2021, and contains around 17.6 million jobs. Our results show that our tool achieves high predictive accuracy R2 (72 % using LightGBM for predicting the memory and 74 % using Random Forest for predicting the time).

Finally, we demonstrate the financial and computational cost impact of the overestimation of job resources on the cloud. We compare the cost of running the jobs with and without using our machine learning tool on the most popular cloud computing resources such as Amazon Web Services (AWS), Microsoft Azure, Digital Ocean, Google Cloud, and the local resources of Holland Computing Center at the University of Nebraska-Lincoln. Our work shows that the significance of our study on a project or researcher's budget and the impact it can have on HPC systems. We found that our cost-effective Machine Learning tool can reduce the average cost of running jobs on the cloud by up to 39% for the BEOCAT jobs and up to 47% for the RMACC- Summit resources. Moreover, decrease the average running time to meet the deadlines by 39% for the BEOCAT jobs and up to 52% for the RMACC-Summit resources.

6.2 Limitations and directions for future work

Our future work will include continuing to improve our model by applying additional machine learning algorithms and sophisticated models such as Convolutional Neural Networks

(CNN), Deep Neural Networks (DNN), and Deep Reinforcement Learning (DRL). This will include testing our module in a real HPC system, and a bigger testbed to achieve more accurate results.

In addition, future work on the 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 testbed 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.

Moreover, our future work can include the following:

- **Evaluate our machine learning model using different cluster configurations**

  We assumed our machine learning models are effective and working on 100 % healthy cluster. We meant by healthy cluster is that applications are running using 100 % of the cores and memory capabilities. That means nothing that affects the performance of the cluster such as memory leak or zombie processes running on the background.

  Hence, one potential future work is to examine our machine learning model on different configurations in the sense of anomaly using Cluster configuration tool and LDMS (a low-overhead, low-latency framework for collecting, transferring, and storing metric data on a large distributed computer system)[110].

  The workflow model of our work will be as follows: **i)** Install multiple well known applications such as (CoMD, MiniFE, MiniGhost, Lulish, etc.) on a real cluster **ii)** Run around thousands of jobs using 100 % healthy cluster in order to extract historical

data to be used in our machine learning model. **iii)** Extract the historical data (sacct) from the cluster after finishing all runs. iv) Build a machine learning model for the extracted sacct data via our MARM algorithm and AMPRO-HPCC tool. **v)** Predict the amount of memory and time for all jobs. **vi)** Replace all requested memory and time with predicted values. **vii)** Re-submit all jobs to the cluster using different healthy cluster configurations starting from 100 % healthy cluster; 90% of healthy nodes and 10% of unhealthy nodes; 80% of healthy nodes and 20% of unhealthy nodes; 70% of healthy nodes and 30% of unhealthy nodes; etc. **viii)** Evaluate the performance and the accuracy of our tool for each configuration.

The purpose of this future work is to test the performance and accuracy of our machine learning model using different configurations in real scenarios. This work will involve three stages as the following: **i)** Verifying the prediction of our machine learning model in a real-time system for a single job. **ii)** Verifying the prediction of our machine learning model per system. **iii)** Verifying the prediction of our machine learning model and how the system behaves for a real system in the presence of anomalies.

- **Integrate our ML methodology to Slurm Workload manager**

  Integrate our software based tool AMPRO-HPCC as an application program interface (API) into Slurm workload manager and other HPC schedulers. This will make the process of building the ML models and predicting job resources for the HPC users much easier and smooth.

- **Build ML models for other schedulers such as Tera-scale Open-source Resource and Queue manager (TORQUE), Portable Batch System (PBS), and HTCondor**

  We have developed our research based on Sun Grid Engine and Simple Linux Utility for Resource Management (Slurm Workload Manager). The future work can involve expanding our work to involve more well known HPC schedulers such as Maui Cluster

Scheduler, Tera-scale Open-source Resource and Queue manager (TORQUE), and Portable Batch System (PBS). In addition, build and investigate the performance of our ML model using HTCondor[111] on Open Science Grid (OSG)[112].

Bibliography

[1] Thomas Sterling, Maciej Brodowicz, and Matthew Anderson. *High performance computing: modern systems and practices*. Morgan Kaufmann, 2017.

[2] Earl Joseph and Steve Conway. Major trends in the worldwide hpc market, 2021.

[3] HPC America's. The vital importance of high-performance computing to us competitiveness.

[4] Mohammed Tanash, Brandon Dunn, Daniel Andresen, William Hsu, Huichen Yang, and Adedolapo Okanlawon. Improving hpc system performance by predicting job resources via supervised machine learning. In *Proceedings of the Practice and Experience in Advanced Research Computing on Rise of the Machines (learning)*, pages 1–8. 2019.

[5] Andy B. Yoo, Morris A. Jette, and Mark Grondona. Slurm: Simple linux utility for resource management. In Dror Feitelson, Larry Rudolph, and Uwe Schwiegelshohn, editors, *Job Scheduling Strategies for Parallel Processing*, pages 44–60, Berlin, Heidelberg, 2003. Springer Berlin Heidelberg. ISBN 978-3-540-39727-4.

[6] N.R. Council, D.E.L. Studies, D.E.P. Sciences, and C.P.I.H.E.C.I.F.S. Engineering. *The Potential Impact of High-End Capability Computing on Four Illustrative Fields of Science and Engineering*. National Academies Press, 2008. ISBN 9780309124850. URL https://books.google.com/books?id=2XadAgAAQBAJ.

[7] Chaowei Yang, David Wong, Qianjun Miao, and Ruixin Yang, editors. *Advanced Geoinformation Science*. CRC Press, October 2010. doi: 10.1201/b10280. URL https://doi.org/10.1201/b10280.

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

[9] Maui Scheduler Steering Committee. Maui scheduler open cluster software. URL http://mauischeduler.sourceforge.net/.

[10] Torque resource manager. http://www.adaptivecomputing.com/products/torque/. (Accessed on 02/02/2019).

[11] Pbs professional open source project. https://www.pbspro.org/. (Accessed on 02/03/2019).

[12] Slurm workload manager - documentation. https://slurm.schedmd.com/. (Accessed on 01/07/2019).

[13] 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).

[14] 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. A slurm simulator: Implementation and parametric analysis. In Stephen Jarvis, Steven Wright, and Simon Hammond, editors, *High Performance Computing Systems. Performance Modeling, Benchmarking, and Simulation*, pages 197–217, Cham, 2018. Springer International Publishing. ISBN 978-3-319-72971-8.

[15] Andréa Matsunaga and José A.B. Fortes. On the use of machine learning to predict the time and resources consumed by applications. In *2010 10th IEEE/ACM International Conference on Cluster, Cloud and Grid Computing*. IEEE, 2010. doi: 10.1109/ccgrid.2010.98. URL https://doi.org/10.1109/ccgrid.2010.98.

[16] Warren Smith. Prediction services for distributed computing. In *2007 IEEE International Parallel and Distributed Processing Symposium*. IEEE, 2007. doi: 10.1109/ipdps.2007.370276. URL https://doi.org/10.1109/ipdps.2007.370276.

[17] Rajath Kumar and Sathish Vadhiyar. Identifying quick starters: Towards an integrated framework for efficient predictions of queue waiting times of batch parallel jobs. In Walfredo Cirne, Narayan Desai, Eitan Frachtenberg, and Uwe Schwiegelshohn, editors, *Job Scheduling Strategies for Parallel Processing*, pages 196–215, Berlin, Heidelberg, 2013. Springer Berlin Heidelberg. ISBN 978-3-642-35867-8.

[18] Fenoy García and Carlos. Improving hpc applications scheduling with predictions based on automatically collected historical data, Oct 2014. URL https://upcommons.upc.edu/handle/2099.1/23049.

[19] Eric Gaussier, David Glesser, Valentin Reis, and Denis Trystram. Improving backfilling by using machine learning to predict running times. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis on - SC '15*. ACM Press, 2015. doi: 10.1145/2807591.2807646. URL https://doi.org/10.1145/2807591.2807646.

[20] Josep Ll. Berral, Íñigo Goiri, Ramón Nou, Ferran Julià, Jordi Guitart, Ricard Gavaldà, and Jordi Torres. 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, 2010. doi: 10.1145/1791314.1791349. URL https://doi.org/10.1145/1791314.1791349.

[21] Bruce Bugbee, Caleb Phillips, Hilary Egan, Ryan Elmore, Kenny Gruchalla, and Avi Purkayastha. 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):155–165, February 2017. doi: 10.1002/sam.11339. URL https://doi.org/10.1002/sam.11339.

[22] Beocat. https://support.beocat.ksu.edu/BeocatDocs/index.php/Main_Page. (Accessed on 03/013/2019).

[23] Getting started with scikit-learn for machine learning. In *Python® Machine Learning*, pages 93–117. John Wiley & Sons, Inc., April 2019. doi: 10.1002/9781119557500.ch5. URL https://doi.org/10.1002/9781119557500.ch5.

[24] L. Massaron and A. Boschetti. *Regression Analysis with Python*. Packt Publishing, 2016. ISBN 9781783980741. URL https://books.google.com/books?id=d2tLDAAAQBAJ.

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

[26] Dan Andresen, William Hsu, Huichen Yang, and Adedolapo Okanlawon. Machine learning for predictive analytics of compute cluster jobs. *CoRR*, abs/1806.01116, 2018. URL http://arxiv.org/abs/1806.01116.

[27] Mohammed Tanash, Huichen Yang, Daniel Andresen, and William Hsu. Ensemble prediction of job resources to improve system performance for slurm-based hpc systems. In *Practice and Experience in Advanced Research Computing*, pages 1–8. 2021.

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

[29] Adedolapo Okanlawon, Huichen Yang, Avishek Bose, William Hsu, Dan Andresen, and Mohammed Tanash. Feature selection for learning to predict outcomes of compute cluster jobs with application to decision support. *arXiv preprint arXiv:2012.07982*, 2020.

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

[31] Documentation index. http://www.adaptivecomputing.com/support/documentation-index/. (Accessed on 02/011/2019).

[32] Dror G Feitelson, Dan Tsafrir, and David Krakov. Experience with using the parallel workloads archive. *Journal of Parallel and Distributed Computing*, 74(10):2967–2982, 2014.

[33] Michael Pinedo. *Scheduling*, volume 29. Springer, 2012.

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

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

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

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

[38] Qiqi Wang, Jing Li, Shuo Wang, and Guibao Wu. A novel two-step job runtime estimation method based on input parameters in hpc system. In *2019 IEEE 4th International Conference on Cloud Computing and Big Data Analysis (ICCCBDA)*, pages 311–316. IEEE, 2019.

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

[40] Young-Kyoon Suh, Seounghyeon Kim, and Jeeyoung Kim. Clutch: A clustering-driven runtime estimation scheme for scientific simulations. *IEEE Access*, 8:220710–220722, 2020.

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

[42] Yuping Fan, Zhiling Lan, Taylor Childers, Paul Rich, William Allcock, and Michael E Papka. Deep reinforcement agent for scheduling in hpc. *arXiv preprint arXiv:2102.06243*, 2021.

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

[44] Théo Saillant, Jean-Christophe Weill, and Mathilde Mougeot. Predicting job power consumption based on rjms submission data in hpc systems. In *International Conference on High Performance Computing*, pages 63–82. Springer, 2020.

[45] Omar Aaziz, Jonathan Cook, and Mohammed Tanash. Modeling expected application runtime for characterizing and assessing job performance. In *2018 IEEE International Conference on Cluster Computing (CLUSTER)*, pages 543–551. IEEE, 2018.

[46] Jonathon Anderson, Patrick J Burns, Daniel Milroy, Peter Ruprecht, Thomas Hauser, and Howard Jay Siegel. 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*, pages 1–7. 2017.

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

[48] 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. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011.

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

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

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

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

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

[54] Mohammed Tanash, Daniel Andresen, and William Hsu. Ampro-hpcc: A machine-learning tool for predicting resources on slurm hpc clusters. In *The Fifteenth International Conference on Advanced Engineering Computing and Applications in Sciences ADVCOMP 2021*, pages 20–27. 2021.

[55] Cynthia Bailey Lee, Yael Schwartzman, Jennifer Hardy, and Allan Snavely. Are user

runtime estimates inherently inaccurate? In *Workshop on Job Scheduling Strategies for Parallel Processing*, pages 253–263. Springer, 2004.

[56] Matthias Hovestadt, Odej Kao, Axel Keller, and Achim Streit. Scheduling in hpc resource management systems: Queuing vs. planning. In *Workshop on Job Scheduling Strategies for Parallel Processing*, pages 1–20. Springer, 2003.

[57] tanash1983. Tanash1983/ampro-hpcc: A machine-learning-tool for predicting job resources on hpc clusters. URL https://github.com/tanash1983/AMPRO-HPCC.

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

[59] Bill Nitzberg, Jennifer M Schopf, and James Patton Jones. Pbs pro: Grid computing and scheduling attributes. In *Grid resource management*, pages 183–190. Springer, 2004.

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

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

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

[63] Muhammad Hafizhuddin Hilman, Maria Alejandra Rodriguez, and Rajkumar Buyya. Task runtime prediction in scientific workflows using an online incremental learning

approach. In *2018 IEEE/ACM 11th International Conference on Utility and Cloud Computing (UCC)*, pages 93–102. IEEE, 2018.

[64] Eduardo R Rodrigues, Renato LF Cunha, Marco AS Netto, and Michael Spriggs. Helping hpc users specify job memory requirements via machine learning. In *2016 Third International Workshop on HPC User Support Tools (HUST)*, pages 6–13. IEEE, 2016.

[65] Slurm workload manager. URL https://slurm.schedmd.com/sacct.html. retrieved: 04, 2021.

[66] 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. A slurm simulator: Implementation and parametric analysis. In *International Workshop on Performance Modeling, Benchmarking and Simulation of High Performance Computer Systems*, pages 197–217. Springer, 2017.

[67] EC Amazon. Amazon web services. *Available in: http://aws. amazon. com/es/ec2/(November 2012)*, page 39, 2015.

[68] David Chappell et al. Introducing the azure services platform. *White paper, Oct*, 1364 (11), 2008.

[69] John J JJ Geewax. *Google Cloud Platform in Action*. Simon and Schuster, 2018.

[70] Angelo Bernasconi, Cristiano Beretta, Walter Bernocchi, Giorgio Richelli, et al. *IBM Spectrum Virtualize and IBM Spectrum Scale in an Enhanced Stretched Cluster Implementation*. IBM Redbooks, 2015.

[71] Adam G Carlyle, Stephen L Harrell, and Preston M Smith. Cost-effective hpc: The community or the cloud? In *2010 IEEE Second International Conference on Cloud Computing Technology and Science*, pages 169–176. IEEE, 2010.

[72] Ye Hu, Johnny Wong, Gabriel Iszlai, and Marin Litoiu. Resource provisioning for cloud computing. In *Proceedings of the 2009 Conference of the Center for Advanced Studies on Collaborative Research*, pages 101–111, 2009.

[73] Wenwen Gong, Lianyong Qi, and Yanwei Xu. Privacy-aware multidimensional mobile service quality prediction and recommendation in distributed fog environment. *Wireless Communications and Mobile Computing*, 2018, 2018.

[74] Lei Wu, Ran Ding, Zhaohong Jia, and Xuejun Li. Cost-effective resource provisioning for real-time workflow in cloud. *Complexity*, 2020, 2020.

[75] Maria Alejandra Rodriguez and Rajkumar Buyya. Deadline based resource provisioningand scheduling algorithm for scientific workflows on clouds. *IEEE transactions on cloud computing*, 2(2):222–235, 2014.

[76] Aniruddha Marathe, Rachel Harris, David K Lowenthal, Bronis R De Supinski, Barry Rountree, Martin Schulz, and Xin Yuan. A comparative study of high-performance computing on the cloud. In *Proceedings of the 22nd international symposium on High-performance parallel and distributed computing*, pages 239–250, 2013.

[77] Christian Vecchiola, Suraj Pandey, and Rajkumar Buyya. High-performance cloud computing: A view of scientific applications. In *2009 10th International Symposium on Pervasive Systems, Algorithms, and Networks*, pages 4–16. IEEE, 2009.

[78] Moustafa AbdelBaky, Manish Parashar, Hyunjoo Kim, Kirk E Jordan, Vipin Sachdeva, James Sexton, Hani Jamjoom, Zon-Yin Shae, Gergina Pencheva, Reza Tavakoli, et al. Enabling high-performance computing as a service. *Computer*, 45 (10):72–80, 2012.

[79] Abhishek Gupta, Laxmikant V Kale, Filippo Gioachin, Verdi March, Chun Hui Suen, Bu-Sung Lee, Paolo Faraboschi, Richard Kaufmann, and Dejan Milojicic. The who,

what, why, and how of high performance computing in the cloud. In *2013 IEEE 5th international conference on cloud computing technology and science*, volume 1, pages 306–314. IEEE, 2013.

[80] Renato LF Cunha, Eduardo R Rodrigues, Leonardo P Tizzei, and Marco AS Netto. Job placement advisor based on turnaround predictions for hpc hybrid clouds. *Future Generation Computer Systems*, 67:35–46, 2017.

[81] Marco AS Netto, Renato LF Cunha, and Nicole Sultanum. Deciding when and how to move hpc jobs to the cloud. *Computer*, 48(11):86–89, 2015.

[82] Iman Sadooghi, Jesús Hernández Martin, Tonglin Li, Kevin Brandstatter, Ketan Maheshwari, Tiago Pais Pitta de Lacerda Ruivo, Gabriele Garzoglio, Steven Timm, Yong Zhao, and Ioan Raicu. Understanding the performance and potential of cloud computing for scientific applications. *IEEE Transactions on Cloud Computing*, 5(2):358–371, 2015.

[83] Miguel G Xavier, Marcelo V Neves, Fabio D Rossi, Tiago C Ferreto, Timoteo Lange, and Cesar AF De Rose. Performance evaluation of container-based virtualization for high performance computing environments. In *2013 21st Euromicro International Conference on Parallel, Distributed, and Network-Based Processing*, pages 233–240. IEEE, 2013.

[84] Keith R Jackson, Lavanya Ramakrishnan, Krishna Muriki, Shane Canon, Shreyas Cholia, John Shalf, Harvey J Wasserman, and Nicholas J Wright. Performance analysis of high performance computing applications on the amazon web services cloud. In *2010 IEEE second international conference on cloud computing technology and science*, pages 159–168. IEEE, 2010.

[85] David Bailey, Tim Harris, William Saphir, Rob Van Der Wijngaart, Alex Woo, and

Maurice Yarrow. The nas parallel benchmarks 2.0. Technical report, Technical Report NAS-95-020, NASA Ames Research Center, 1995.

[86] Eduardo Roloff, Matthias Diener, Alexandre Carissimi, and Philippe OA Navaux. High performance computing in the cloud: Deployment, performance and cost efficiency. In *4th IEEE International Conference on Cloud Computing Technology and Science Proceedings*, pages 371–378. IEEE, 2012.

[87] Prakash Murali and Sathish Vadhiyar. Qespera: an adaptive framework for prediction of queue waiting times in supercomputer systems. *Concurrency and Computation: Practice and Experience*, 28(9):2685–2710, 2016.

[88] Rajath Kumar and Sathish Vadhiyar. Identifying quick starters: towards an integrated framework for efficient predictions of queue waiting times of batch parallel jobs. In *Workshop on Job Scheduling Strategies for Parallel Processing*, pages 196–215. Springer, 2012.

[89] Daniel Nurmi, John Brevik, and Rich Wolski. Qbets: Queue bounds estimation from time series. In *Workshop on Job Scheduling Strategies for Parallel Processing*, pages 76–101. Springer, 2007.

[90] Hui Li, Juan Chen, Ying Tao, David Gro, and Lex Wolters. Improving a local learning technique for queuewait time predictions. In *Sixth IEEE International Symposium on Cluster Computing and the Grid (CCGRID'06)*, volume 1, pages 335–342. IEEE, 2006.

[91] Leo T Yang, Xiaosong Ma, and Frank Mueller. Cross-platform performance prediction of parallel applications using partial execution. In *SC'05: Proceedings of the 2005 ACM/IEEE Conference on Supercomputing*, pages 40–40. IEEE, 2005.

[92] Dan Tsafrir, Yoav Etsion, and Dror G Feitelson. Backfilling using system-generated predictions rather than user runtime estimates. *IEEE Transactions on Parallel and Distributed Systems*, 18(6):789–803, 2007.

[93] C Atkeson. Locally weighted regression. *Artificial Intelligence Review*, 1997.

[94] Warren Smith. Prediction services for distributed computing. In *2007 IEEE International Parallel and Distributed Processing Symposium*, pages 1–10. IEEE, 2007.

[95] Maciej Malawski, Kamil Figiela, and Jarek Nabrzyski. Cost minimization for computational applications on hybrid cloud infrastructures. *Future Generation Computer Systems*, 29(7):1786–1794, 2013.

[96] Saurabh Kumar Garg, Rajkumar Buyya, and Howard Jay Siegel. Scheduling parallel applications on utility grids: Time and cost trade-off management. In *ACSC*, volume 9, pages 139–147. Citeseer, 2009.

[97] Nik Bessis, Stelios Sotiriadis, Fatos Xhafa, Florin Pop, and Valentin Cristea. Meta-scheduling issues in interoperable hpcs, grids and clouds. *International Journal of Web and Grid Services*, 8(2):153–172, 2012.

[98] Stelios Sotiriadis, Nik Bessis, and Nick Antonopoulos. Towards inter-cloud schedulers: A survey of meta-scheduling approaches. In *2011 International Conference on P2P, Parallel, Grid, Cloud and Internet Computing*, pages 59–66. IEEE, 2011.

[99] Marcos Dias De Assunção, Alexandre Di Costanzo, and Rajkumar Buyya. Evaluating the cost-benefit of using cloud computing to extend the capacity of clusters. In *Proceedings of the 18th ACM international symposium on High performance distributed computing*, pages 141–150, 2009.

[100] Saeid Abrishami, Mahmoud Naghibzadeh, and Dick HJ Epema. Deadline-constrained workflow scheduling algorithms for infrastructure as a service clouds. *Future Generation Computer Systems*, 29(1):158–169, 2013.

[101] Jiyuan Shi, Junzhou Luo, Fang Dong, and Jinghui Zhang. A budget and deadline aware scientific workflow resource provisioning and scheduling mechanism for cloud. In

Proceedings of the 2014 IEEE 18th International Conference on Computer Supported Cooperative Work in Design (CSCWD), pages 672–677. IEEE, 2014.

[102] Amandeep Verma and Sakshi Kaushal. Cost-time efficient scheduling plan for executing workflows in the cloud. *Journal of Grid Computing*, 13(4):495–506, 2015.

[103] Wei Zheng, Yingsheng Qin, Emmanuel Bugingo, Dongzhan Zhang, and Jinjun Chen. Cost optimization for deadline-aware scheduling of big-data processing jobs on clouds. *Future Generation Computer Systems*, 82:244–255, 2018.

[104] URL https://cloud.google.com/products/calculator.

[105] Pricing microsoft azure, . URL https://azure.microsoft.com/en-us/pricing/details/virtual-machines/linux/.

[106] . URL https://www.digitalocean.com/pricing.

[107] URL https://cloud.ibm.com/vpc-ext/provision/vs.

[108] URL https://calculator.aws/#/createCalculator/EC2.

[109] Priority access pricing, . URL https://hcc.unl.edu/priority-access-pricing.

[110] Anthony Agelastos, Benjamin Allan, Jim Brandt, Paul Cassella, Jeremy Enos, Joshi Fullop, Ann Gentile, Steve Monk, Nichamon Naksinehaboon, Jeff Ogden, et al. The lightweight distributed metric service: a scalable infrastructure for continuous monitoring of large scale computing systems and applications. In *SC'14: Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, pages 154–165. IEEE, 2014.

[111] HTCondor Team. Htcondor, Jan 2020. URL https://zenodo.org/record/3595387#.YXmektnMKbs.

[112] Ruth Pordes, Don Petravick, Bill Kramer, Doug Olson, Miron Livny, Alain Roy, Paul Avery, Kent Blackburn, Torre Wenaus, Frank Würthwein, et al. The open science grid. In *Journal of Physics: Conference Series*, volume 78, page 012057. IOP Publishing, 2007.