Optimizing high performance computing system's, resource utilization and throughput by leveraging machine learning

by

Brandon Dunn

B.Sc., Kansas State University, 2021

A THESIS

submitted in partial fulfillment of the
requirements for the degree

MASTER OF SCIENCE

Department of Computer Science
Carl R. Ice Collage of Engineering

KANSAS STATE UNIVERSITY
Manhattan, Kansas

2021

Approved by:

Major Professor
Dan Andresen

Copyright

Abstract

High Performance Computing (HPC) facilitates a significant portion of research and analytics across many different fields, industries, and education. HPC is implemented using supercomputers, which can be comprised of a few servers to tens to thousands. HPC systems typically use a scheduler - such as Slurm - to manage the execution of tasks on the system. Schedulers typically have hundreds of configuration parameters. With such diverse workflows and hardware the question becomes: how do we adapt these HPC schedulers so that we keep a high utilization and throughput on the systems? Our research focuses on optimizing the SLURM scheduler by adapting its configuration options based on the type of hardware in the High Performance Computing system and types of workflows, utilizing Semi-supervised Machine Learning.

Table of Contents

List of Figures

List of Tables

Acknowledgments

I would like to acknowledge my Dan Andresen, Not only has he been a great Major professor but he is a good friend. He was there to offer encouragement and sometimes giving me stern push. I would also like to show my appreciation for the beocat team (Adam, Kyle, and Dave) for always being there when I need a pick me up when I thought I would fail. I would also like to acknowledge Mohammad Tanash and Huichen Yang for always being willing to help me find answers to questions I had. Finally thank you to all the friends and family that supported me through this time.

Dedication

To my loving wife Laura,

You have always been there to lift me up and spur me on when I doubted that I would make it through. You stayed up late and sacrificed sleep to make sure I never came home to dark and solemn house after long hours of school. You never failed to express the deepest affection and gratitude for the time I had to sacrifice with you and our wonderful boys in order to see this through. I wish that I was capable of expressing how completing this thesis was only made possible through your devotion and strength, for without this my strength would have failed long before the end of this road.

To my amazing sons Johnathon and Ethan,

I know that I missed some of important things as you grew because I was seeking a means to give you a better life than I was given, as is the dream of all parents. Please know that while I may have been physically absent for a good portion of time my thoughts were always of making your lives happier and fuller than mine can ever be. I hope that I have inspired you to become the best possible men that you can. I want you to shoot for the stars and know they are reachable.

With all my love,

Brandon Dunn

Chapter 1

Introduction

The integration of High Performance Computing (HPC) in academics and companies around the globe for research and analysis has proven that we can solve many problems we thought would take lifetimes. With this adoption of HPC many companies and academic facilities are learning that running a supercomputer is not like your traditional server setup. The difficulty of managing these systems takes exponentially more effort as the system grows in scale and complexity. Resource management is one of the more complicated parts to implement and maintain. The first problem is choosing the resource manager that works for you system and team. Then you need to configure that resource manager to fit your hardware and workflows that your system will be handling. As time goes on workflows shift and change then these resource managers need to be reconfigured, but often this does not happen for many years if ever during the life of the system. The main reason for this is the sheer number of configuration options that have to be adjusted with every change in workflow. For instance SLURM has about one thousand tune-able parameters for its configuration file. Each parameter can have a different level of impact on scheduling, back-filling, etc. The other issue is the type of cluster that is being used, for example a heterogeneous cluster that can have many different types of nodes that change over time as servers are added, replaced, or removed. These types of clusters typically buy nodes in small batches and added the to the current infrastructure and only remove nodes when the are no longer operable. Usually

these nodes are from different vendors and with completely different hardware configurations. Then you have the types of systems that are bought in their entirety from a single vendor where all the hardware is configured exactly the same. With the heterogeneous clusters you have to deal with both changing hardware and workflows over time whereas with the other it's just changing workflows. The goal of my research is to utilize Machine Learning to adjust these parameters thus increasing the system utilization and throughput. A secondary side effect is that system administrators of these supercomputers can run this tool as often as they need to adapt their system to the changes in workflows. For this thesis I will be focusing on the viability of this research.

1.1 Research Questions

1.1.1 Optimizing Utilization

Can the overall performance of HPC resource managers/schedulers be optimized, particularly in the areas of resource utilization and throughput by leveraging the capabilities of machine learning? That is to say can we make HPC systems more efficient with respect to scheduling jobs on the available resources and as a side effect can we get more jobs through the system?

1.1.2 Percentage increase

If the HPC system's utilization and throughput are in fact increased, then what is the percentage of said increase? Is this percentage of increase adequate enough to warrant further research?

1.1.3 Generalized tool

Can we design a a significantly generalized tool that can be utilized by supercomputers regardless of heterogeneity of the system.

1.2 Limitations and Delimitations

As we strive to make our research as thorough as possible limitations are inevitable. With this in mind let's discuss some of the primary limitations that were encountered while conducting the research for this thesis. Delimitation's are included at the end of each subsection.

1.2.1 Availability of Systems

Biggest limitation was the number of systems available to test the generalized ML model on. With guaranteed access to one supercomputer and the possibility of access to one other system. I would like to get this information for more systems a do a broader testing to make an the generalized ML model more refined in the future.

1.2.2 System Types

The next limitation for testing the generalized ML model was the diversity of the types of systems. The system I had access to was a heterogeneous cluster. This means that the cluster consists of my different types of nodes, with different hardware configurations. There was no access available to a non-heterogeneous system where the nodes are all exactly the same. These differences to play into how each system get optimized. Future work would include getting access to these types of systems for testing and refinement of the generalized ML model.

1.2.3 Resource Managers/Schedulers

As with all things, there are many different resource managers/job schedulers available for HPC clusters. The system I had access to used the SLURM workload manager. Future work would include adding more of these resource managers/ job schedulers to the ML tool for optimizing.

1.3 Assumptions

With limited access to different resource managers/schedulers I had to make a base assumption that all the different ones available operated in relatively the same way. This is a critical assumption if this research is to be expanded past just the Slurm workload manager to others like PBS, MOAB, SGE, etc.

I had to assume that the results from the simulator were accurate, as time to run all of the test jobs on any real HPC cluster would take months or even years.

Finally I had to assume the data about each user and job that came from the different HPC clusters, that I was given was recorded accurately by the systems that they came from.

1.4 Definition of terms

High Performance Computing - The use of a supercomputer(s) to solve computational problems that are generally to large for a single computer or workstation.

Supercomputer - A computer with extremely high performance compared to general-use computers. Performance generally measured in Floating-point operations per second.

Machine Learning - Using artificial intelligence to allow a system to learn and improve through experience rather than being told explicitly how to run.

HPC - High performance Computing

ML - Machine Learning

Chapter 2

Related Material

As ML saturates all fields of research, it is no surprise that researchers would want to utilize it to optimize the code that they run on supercomputers. While much of the research focuses on the code some researchers want to optimize aspects of the actual HPC system its self. In the next few subsections we will outline some of the aspects of the HPC systems that are being researched as of the writing of this paper.

2.1 Optimization for User Job Submissions

Much of the research that surrounds the use of machine learning for the purpose of optimizing high performance computing is centered around how many resources users request when submitting their jobs to the scheduler. This is important research as the cost of setting up and maintaining these super computers is extremely high, with smalls cluster ranging from a couple million dollars to larger cluster costing tens of million. This puts pressure to get a high throughput and utilization from the supercomputer so as to get a good Return on Investment (ROI) of these systems[1].

The issue that currently causes these system to be under utilized decreases throughput is that the current versions of resource managers/job schedulers rely on the users to inform it how much resources there job will utilize. The issue with this approach is that the vast

majority of users do not fully understand how to estimate the correct amount of resources such as ram, number of CPU's, number of nodes, etc, often time they do not even know which queue they are submitting their job to.[2].

The primary reason for this deficiency is that the majority of the researchers using these HPC systems are not programmers, they are scientist trying to solve a complex problem. The lack of proficiency in programming and knowledge of how code uses computer resources has resulted in users being encouraged, often times by others users to over-estimate the needed resources for their jobs, ensuring that they will not be killed[3].

There is another reason that contributes to this deficiency, which is that the systems scheduler has to be told what resources are needed. The resource manager/scheduler allow for user and system administrators to look at what resources are being used while the job is running. However, they do not assist with determining the correct amount of resources that each job should use. Currently no software exists that fills the the need of automating the resource allocation for these jobs[4].

This is all to say that much research is being poured into finding a software solution to allocating resources or finding a way to adjust and optimize the users requested resources. The majority of approaches are using machine learning to predict if a job will fail based on the users requested resources[4][5][3][2][1].

2.2 Optimization for Energy Efficiency

With the HPC systems progressing in scale and density, the energy cost of supporting these systems are expected to grow exponentially. This means that the growing energy consumption is quickly becoming a primary constraint for HPC systems, especially as they approach the exascale level. These systems are predicted to have a 20 MW operating budget[6].

Chapter 3

Design and Methodology

This Chapter we will discuss the design choices that was used during this research.

3.1 Research Workflow

This section discuss the workflow that I used during my research. The workflow will cover what software and libraries used, methods for processing the data used in the experiments, and the environments used to do my research. The process that experiments will follow are shown in figure 3.1, the process will be repeated for ML algorithm. Then the results will be compared and I will select the best model to use in the software My team and I are building.

3.1.1 Software development

This subsection discusses the different programming languages, open source software libraries, custom built libraries and changes made to the simulator.

Programming Languages

The primary language used in for this research is Python. The choice to use Python was made with respect to the two factors. The first being that the majority of machine learning is done using Python, so there are many well developed libraries for implementing the algorithms

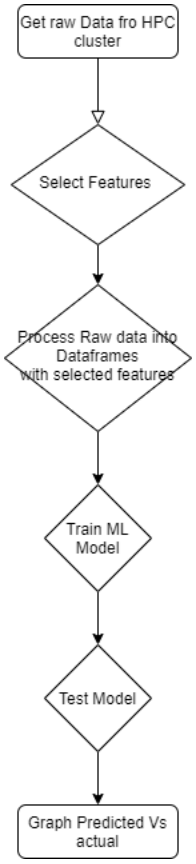

Figure 3.1: *Research Process per ML algorithm*

used in this research. The second reason is that I am very familiar with Python which makes it easier to develop and implement changes.

The next language is C. This is because the slurm resource manager is written in C. During my research I need to make modifications to the slurm simulator API hooks. The modification of these API hooks will be discussed in greater detail in the upgrading the simulator section.

Upgrading simulator

Shortly after we began working with the simulator we ran into some issues with compatibility of CentOS versions, as well as over need to manually install the simulator. The other area of frustration was the need to use R to generate the trace file need to run a simulation. This prompted me to write portions of the simulator to allow for automated installation of the simulator via Python scripts. These scripts not only made installations much easier but also allowed for checking of OS version and adapting the installation based on this information. The other upgrade made to the simulator was to re-write the trace file creation in Python and to have it output the trace file in JSON format. Removing the R component and changing the file format meant that the simulator's API hooks would need to be updated as well to accept the new format. The API hooks were updated to take the JSON format. This changed improved the the overall time it takes to get a simulation running as the new trace file creation was faster using the Python and JSON combination then the previous R and trace format. The R trace file creation was also limited in terms of being able to create random and bulk jobs in a trace file where as the Python can be used to create a wide variety of job types in the trace file.

3.1.2 Data processing

The raw data was retrieved from a supercomputer called Beocat and was in the CSV file format. The file was read into a Python script using the pandas library. Then it was cleaned by removing any erroneous data and the number of columns reduced from one hundred and

five parameters to thirteen. These thirteen were chosen based on the data needed by the simulator being used. Then the newly cleaned data was run through another Python script that created a trace file that the simulator knew how to read.

3.2 Resource Managers and Job Schedulers

This section covers the resource mangers and job schedulers that we used during our research, as well as a brief description of how they work. Resource mangers are responsible for tracking a clusters total hardware resources both what is currently available for use and what is currently in use. This information is then passed to the scheduler so that the scheduler can decide what job in the queue can fit where. The scheduler is responsible for taking the information from the resource manager and then adds the information such as requested run time, memory, CPU's, Network type, given by the job requests and determines what jobs in the queue can be fit in the system given available resources. The resource mangers and schedulers we used were coupled together as part of one software package, however there are other configurations where these are separate software packages. For our research we use the following Resource Mangers/Schedulers

3.2.1 Slurm Workload Manager

For this research we used the Slurm workload manager as it was what was used on the systems available to me for. Slurm is a job scheduler and cluster management system for Linux based clusters. The three key functions of Slurm are to allocate resources to users; provide a means for starting, stopping, and monitoring jobs; and it manages a queue for resource contention. This allows users to request the number of resources and schedule a job to run on the system. Slurm takes the users job request that contains some number of requested resources compares it to the current queue and available resources and determines where it should go in the queue and if it can run immediately. Slurm periodically checks the queue, currently running jobs, and available resource and back-fills jobs as resource become

available[7]. This is the scheduler that this research will be focusing on optimizing.

3.2.2 Sun Grid Engine

Sun Grid Engine is a grid computing software for executing UNIX batch jobs most commonly in the for of shell scripts, within a pool of cooperating systems (serves, workstations, etc). Jobs are placed in queued then get executed on the cluster at times when those systems would otherwise be idle or only lightly loaded. The work load is distributed among the workstations in the cluster corresponding to the load situation of each machine and the resource requirements of the jobs[8]. For this research we only used data about users and jobs submitted not for optimizing the scheduler.

3.3 Simulation

This section describes the why we used simulation and the simulator we used for the simulations. For this research we needed to be able to rapidly test effectiveness of different configurations of the Slurm scheduler to see if our machine learning models were effective. This meant that we would need to be able to schedule and run thousands maybe tens of thousands of job with each different configuration and measure the the throughput and resource utilization. This would not be achievable through standard means of changing a clusters configuration file and running jobs through a cluster that was already in use, so we decided that a simulator would best server our needs. With a simulator we could simulate a cluster and run simulated jobs and compare it to the actual system we were emulating to see if our ML models gave us a better result. The other useful thing about the simulator was that we were able to make singularity containers to deploy on a supercomputer to run several simulations with different configurations simultaneously. For this research we use the following simulator.

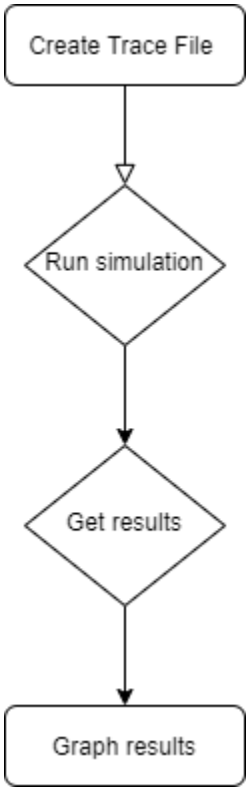

Figure 3.2: *Slurm Simulator workflow*

3.3.1 Slurm Simulator

We decided on using the the Slurm simulator that was developed by University of Buffalo's Center for Computational Research (UBCCR), as Slurm was the chosen scheduler for this research. The Slurm simulator was also chosen because it is designed to simulate an entire cluster on a single workstation or server and to run with time acceleration. The Slurm simulator can simulate approximately 17 days per hour depending on the complexity and scale of the jobs. The simulator provides the capability of rapid testing of new Slurm plugins and configuration options[9]. With the aforementioned capabilities the Slurm simulator gave us a method of testing the new configuration files without need of an actual HPC system. Figure 3.2 shown below shows the high level operations of the simulators workflow.

3.4 Viability

in this section we discuss testing the viability of this research. To do this I started of by manually changing a couple of configuration options to see if they actually affected the throughput and queue wait times. The goal for this was to just see if the path we were on makes sense to keep going down. For the purposes of this thesis I decided to look at three of the configuration options: PriorityWeightAge, PriorityCalcPeriod, and MaxJobCount. The details of these factors and the choices to use them is detailed in the subsections below. The goal of this work to show that using machine learning is a viable option for improving the Slurm scheduler and possibly other using machine learning. for this thesis this will be done through some manual viability testing, however there will be a paper with the ML models.

3.4.1 Priority Weight Age

This subsection discusses what the Priority Weight Age (PWA) factor is, how it affects the system, and why is was chosen for viability testing. The PWA is the scaling factor that is used to scale the age factor in the multi-factor priority plugin. The Age factor is the a value of the time the job has been waiting in the queue. Since the age factor is set buy the scheduler i choose to use the PWA as it is used to scale the age. The multi-factor priority plugin uses the PWA in the job priority calculation show here:

```
    Job_priority =
site_factor +
(PriorityWeightAge) * (age_factor) +
(PriorityWeightAssoc) * (assoc_factor) +
(PriorityWeightFairshare) * (fair-share_factor) +
(PriorityWeightJobSize) * (job_size_factor) +
(PriorityWeightPartition) * (partition_factor) +
(PriorityWeightQOS) * (QOS_factor) +
SUM(TRES_weight_cpu * TRES_factor_cpu,
TRES_weight_<type> * TRES_factor_<type>,
```

```
    ...)
- nice_factor
```

[7] The above calculation shows several more factors then what are being used for this paper, however these factors are being considered and will be talked about in the future work subsection of the conclusion section.

3.4.2 Priority Calculation Period

In this subsection we will talk about priority calculation period factor, what it is and how it might be able to increase the performance of the Slurm scheduler. The priority calculation period a value in minuets, of how often the priority decay half-life gets recalculated. To understand how this effects the schedulers we will also look at what the priority decay half-life does in the scheduler. These factors are also part of the multi-factor priority plugin for Slurm. The priority decay half-life factor determines how much of the historical usage has on the composite usage value. a larger number, the more past usage affects fair-share. if the value is set to 0 then no decay is applied. I did not know enough about the system's data I was using priority for there users to know if this should be adjusted, this is why I chose to use the calculation period for this instead.

3.4.3 Max Job Count

This subsection discusses the what the max job count factor is, how it is used by the system's scheduler, and how it can impact performance of the scheduler. The max job count is the max number of jobs that the Slurm scheduler can have in it's active jobs database. This value will can cause the scheduler to use more memory then available on the server if set to high, however it will also cause job submissions to fail if set to low. The default value for this set to 10,000. for the viability testing I will be varying this value form and using the default as the baseline.

3.5 Machine learning

This sections covers the machine learning algorithms that I will trying to use to predict the aforementioned factor values. I will move forward with the machine learning if the viability test show that this research is worth pursuing. I plan on using a few simple algorithms initially and then move to more advanced later. The advanced algorithms will be discussed in future work section. The simple algorithms are discussed in the subsections below.

3.5.1 Linear Regression

Linear Regression is a supervised machine learning algorithm that uses independent variable to predict continuous dependent variables, represented as a constant slope. Because this algorithm is simple to implement and works for predicting a value within a continuous range it is often the first algorithm used to test if machhine learning is an option. Since much of machine learning research starts by using the standard linear regression algorithm, I decided to use it for the initial prediction of the priorityweightage, prioritycalcperiod, and the maxjobcount factors.

Chapter 4

Results

4.1 Initial Manual Tests

In this section I go over the results of my initial testing, where I manually adjusted each of the configuration parameters discussed in the subsections below. These tests showed that just a few parameters each had an impact on the schedulers performance. While each individual impact was not a massive game changer, figures 4.1 4.2 show that we can adjust the performance of the HPC when compared to the default scheduler configurations show in figures 4.3 4.4. Theses result also suggest the possibility of even higher performance gains by changing more than a single parameter at a time. All of these tests were done using a trace file with 100,000 jobs of varying amounts of CPU, memory, and number of nodes. The results shown in the above figures may have more of an impact if the same number of jobs were run on a smaller cluster or if more jobs were scheduled on this cluster. I would have run a larger amount of jobs, but the number of jobs in the data set without errors was limited.

4.1.1 Priority Weight Age

Let's discuss how the system utilization and the wait time in queue were affected by the changes to the PWA configuration parameter for SLURM. I varied the PWA parameter through six different values shown in the table 4.1. This table also gives the mean and

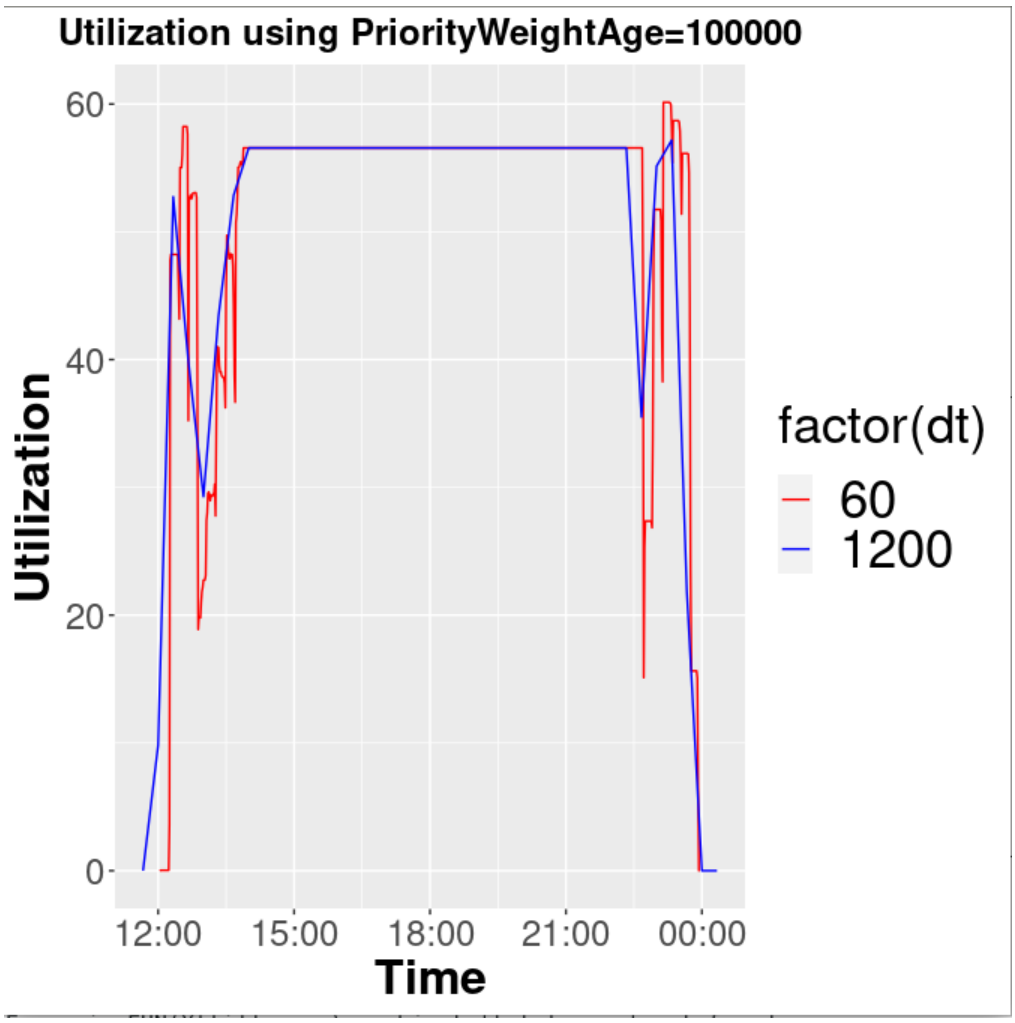

Figure 4.1: *Utilization with PWA set to 100000*

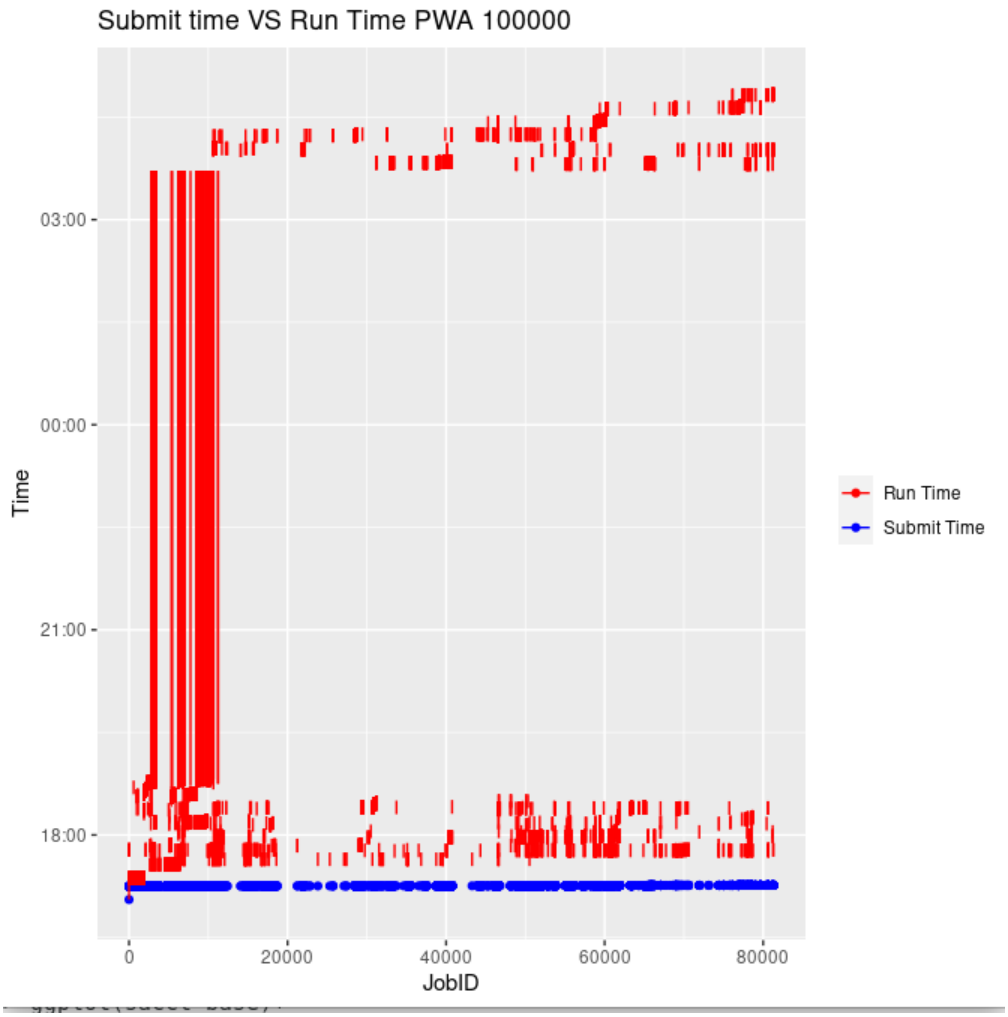

Figure 4.2: *Wait time with PWA set to 100000*

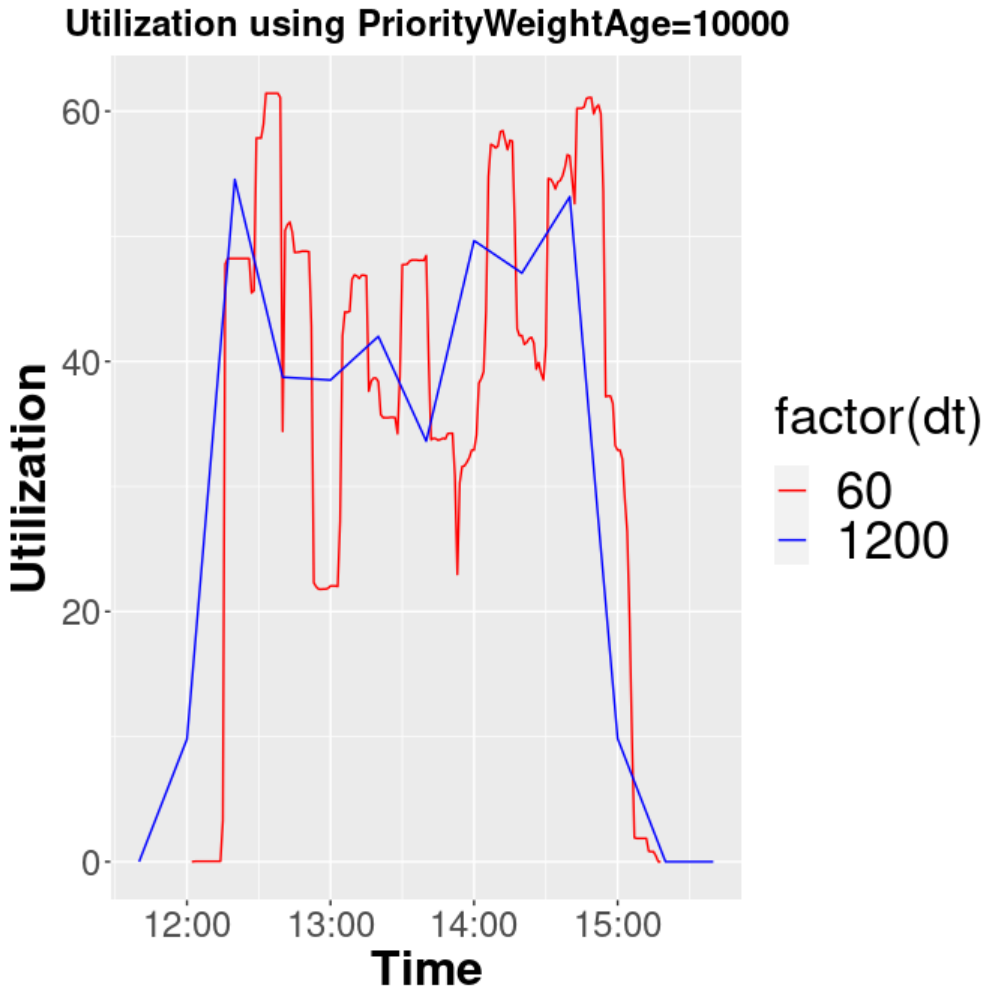

Figure 4.3: *Utilization with Default PWA value of 10000*

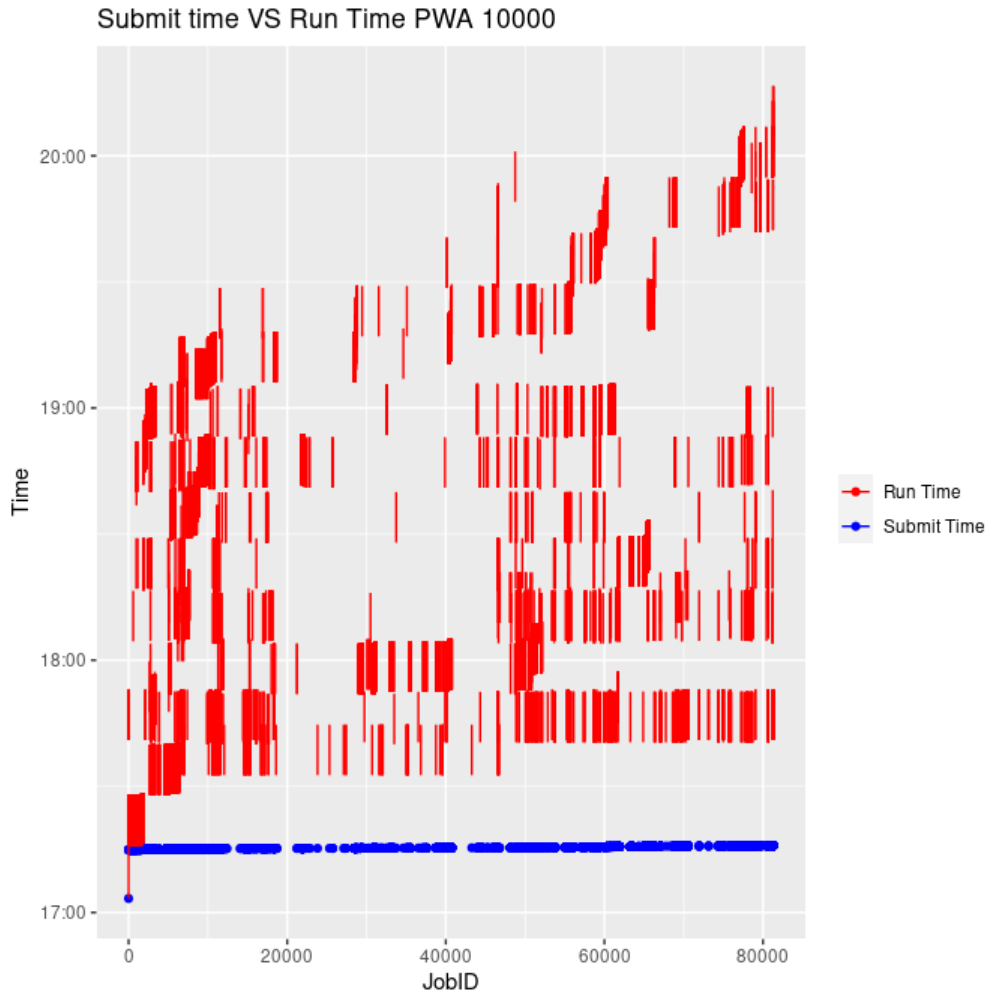

Figure 4.4: *Default wait time with PWA value of 10000*

Table 4.1: *PriorityWeightAge*

PriorityWeightAge	Wait Time (hrs)
1000	1.16
5000	1.18
10000	1.13
25000	1.15
50000	1.08
100000	1.13
Mean	1.14
STD	0.034

standard deviation. This table show that changing this parameter can give us a slight increase to overall scheduler performance by increasing utilization and decreasing the time jobs spend waiting in the queue. It is possible to see higher gains for this parameter by using ML to find the global optima for the system the scheduler is running on.

4.1.2 PriorityCalcPeriod

As discussed in 3 the PCP is the time in minutes until the half-life decay for all jobs in the queue is recalculated. For this test the PWA value was held at the default values used by beocat and the PCP was adjusted to values between 1 and 30 as shown in table 4.2. The results show that while small there is an impact to the amount of time that jobs spend in the queue. This is also evident when looking at figure 4.5 compared to the default in figure 4.6. With a the lower interval The utilization is a little higher for a bit longer then with the default. This does not include the entire range of values for this parameter, so this does leave room for improvement by using an ML algorithm to find an optima amongst all possible values.

4.1.3 MaxJobCount

The results of this test surprised me as I expected this to have more of an impact on the schedulers performance than what is show in table 4.3. Given the time to get information

Table 4.2: *PriorityCalcPeriod*

PriorityCalcPeriod	Wait Time (hrs)
1	1.18
5	1.18
10	1.18
15	1.13
20	1.17
30	1.1
Mean	1.16
STD	0.031

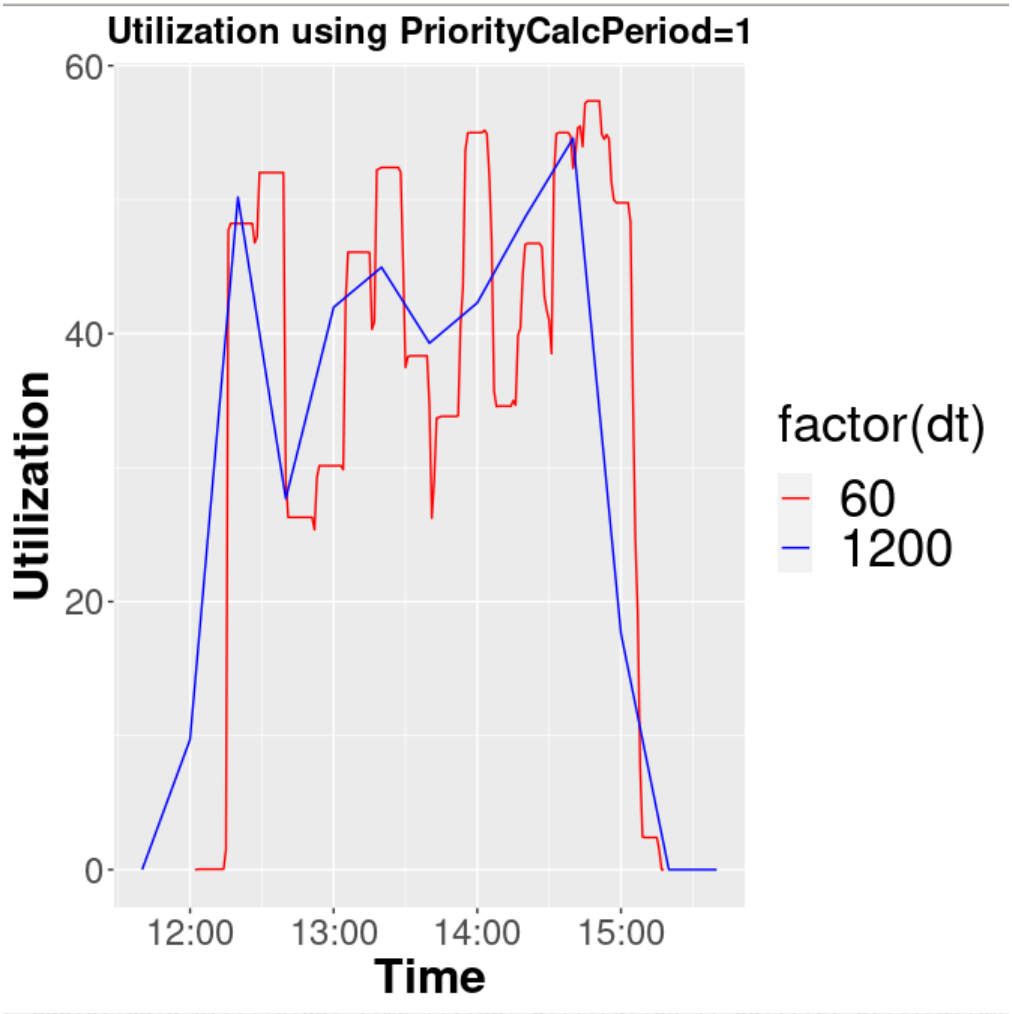

Figure 4.5: *Wait time with PCP set to 1*

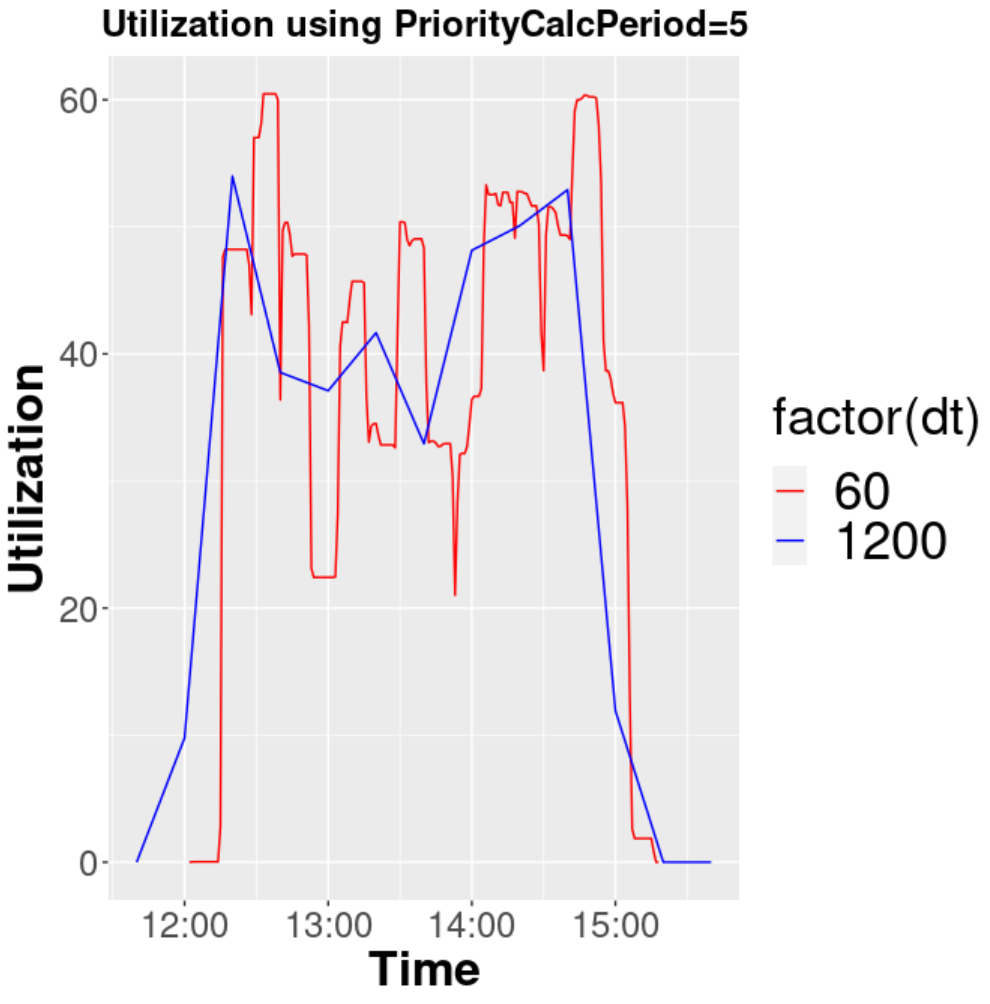

Figure 4.6: *Default wait time with PCP set to 5*

Table 4.3: *MaxJobCount*

MaxJobCount	Wait Time (hrs)
10000	1.14
50000	1.28
100000	1.26
500000	1.26
1000000	1.27
10000000	1.27
Mean	1.25
STD	0.048

from a database then run an algorithm to determine best fit of all jobs in the database I had logically concluded that allowing more jobs would have had a larger impact on wait time than it did. However the results show that it is nearly the same from one hundred thousand to ten million. The utilization seemed to follow my assumptions that a smaller amount of jobs in the database would mean lower utilization. This can be seen in figure 4.7 compared to the default in figure 4.8. Again this did not cover the full range of values that this parameter can take, so there is room for potentially higher gains. Before moving on to the ML models section I think it is worth mentioning that combining the changes theoretically should yield a greater performance gain than any single parameter change. More about this in future work.

4.2 Machine Learning Results

Since the viability of this was shown in the previous section I moved to building a ML model for prediction. I decided to start with a simple linear regression algorithm for this model. I started trying to predict the PriorityWeightAge value, However I ran into issues with generating the model. The biggest issue is that there are so many variables that the simplicity of the linear regression did not generate good results. I decided to drop the failed model and move forward with the results above. The results of this have made it clear that more sophisticated ML methods are needed to address this particular optimization problem.

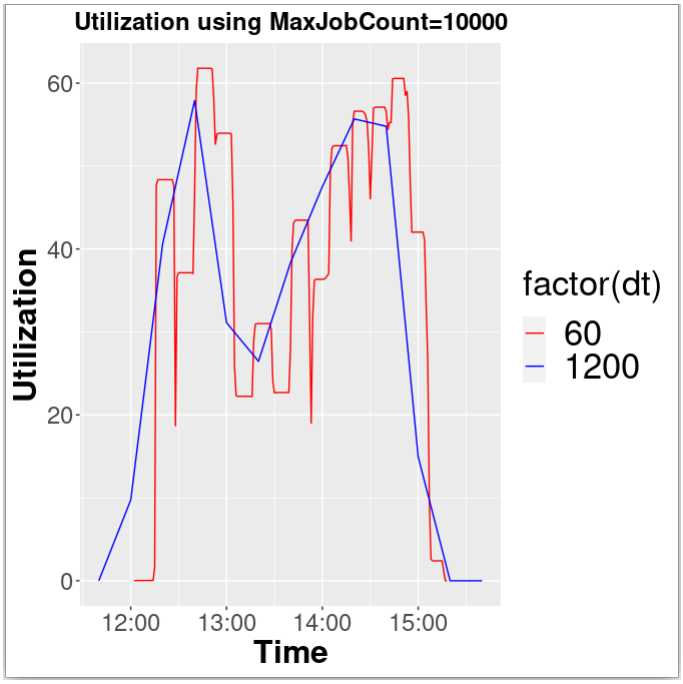

Figure 4.7: *Utilization with MaxJobCount set to 10000*

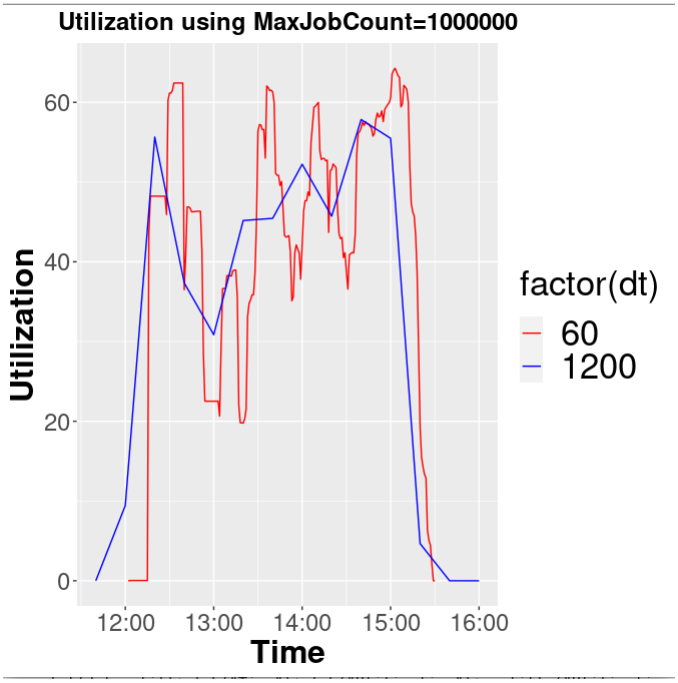

Figure 4.8: *Utilization with default MaxJobCount of 1000000*

Chapter 5

Conclusion

When this paper started I was seeking to answer the question can HPC schedulers be optimized through the use of machine learning techniques. If the answer to this question is yes then how much can we optimize it. Finally can we create a tool that HPC system administrators or anyone who wants to run a cluster of any size can use. In the next few paragraphs I hope show that these questions have an answer and that they are worth pursuing farther.

For the question of optimizing HPC schedulers I believe that the results in this paper (while still in their infancy), show that each parameter has an impact on the performance of the scheduler. While I had issues with the simple variations of ML algorithms, the testing done shows the viability of more complex algorithms. While the failure of the simpler ML algorithms might seem like a set back it, it is in reality a great opportunity to learn and expand our understanding of things. Having spent time think as to why the simpler methods failed, it occurred to me that sometimes in life things that seems simple really are not and we may need a broader view of the situation. This thought has lead me into researching more complex ML method's that can capture the wider scope needed for this optimization question.

While the previous question was not answered in it's entirety. I do think that a sufficient enough yes can be gleaned from the results. I will say that for this portion the percentage

increases are lower than expected, however there still was improvement. I feel the research results support a reasonable assumption that these gains can be further increased by combining these adjustments together and applying more advanced techniques.

Finally addressing the building a generalized tool. In order to conduct this research I have written a good portion code that will be the make up a large portion of the building blocks of this tool. I have written a library for Python that is designed to parse HPC information like users, accounting, and configuration files. As shown this tool is forming and taking shape with each day and research task that passes.

5.1 Future Work

Let's talk about the future. There is so much more potential in this area and the surface has barley been scratched. I know that moving forward on the machine learning side I want to explore better methods to use starting with using more sophisticated models like Convolutional Neural Networks(CNN), Deep Neural Networks(DNN), and Deep Reinforcement Learning(DRL), then using multivariate versions of these models. Finally putting these all together in an ensemble method, for example Taking the output from a CNN then sending it through an Long short-term memory(LSTM) this is shown in figure 5.1. The goal with using these better models and methods is to be able to predict several configuration parameters by including information like the systems hardware configuration, network topology and most common type of workload run on the system.

I will continuing to develop a software based tool that can be used to periodically update these configurations as the system and workloads change overtime. This software will include an updated version of the slurm simulator, an application program interface(API), as well as a GUI. This should provide a means for the HPC community to improve the systems they are using and this software.

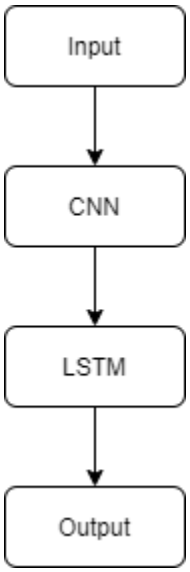

Figure 5.1: *Ensemble Method using CNN and LSTM*

Bibliography

[1] V. Ranganath and D. Andresen. Why do users kill hpc jobs? In *2018 IEEE 25th International Conference on High Performance Computing (HiPC)*, pages 276–283, 2018. doi: 10.1109/HiPC.2018.00039.

[2] Eduardo R. Rodrigues, Renato L. F. Cunha, Marco A. S. Netto, and Michael Spriggs. Helping hpc users specify job memory requirements via machine learning. In *Proceedings of the Third International Workshop on HPC User Support Tools*, HUST '16, page 6–13. IEEE Press, 2016. ISBN 9781509038749.

[3] 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)*, PEARC '19, New York, NY, USA, 2019. Association for Computing Machinery. ISBN 9781450372275. doi: 10.1145/3332186.3333041. URL https://doi-org.er.lib.k-state.edu/10.1145/3332186.3333041.

[4] Andresen Dan, Hsu William, Yang Huichen, and Okanlawon Adedolapo. Machine learning for predictive analytics of compute cluster jobs. In *Proceedings of the International Conference on Scientific Computing (CSC)*, pages 40–46. The Steering Committee of The World Congress in Computer Science, Computer . . . , 2018.

[5] 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.

[6] Connor Imes, Steven Hofmeyr, and Henry Hoffmann. Energy-efficient application re-

source scheduling using machine learning classifiers. In *Proceedings of the 47th International Conference on Parallel Processing*, ICPP 2018, New York, NY, USA, 2018. Association for Computing Machinery. ISBN 9781450365109. doi: 10.1145/3225058.3225088. URL https://doi-org.er.lib.k-state.edu/10.1145/3225058.3225088.

[7] Slurm workload manager - overview. https://slurm.schedmd.com/overview.html, 2015. (Accessed on 11/30/2020).

[8] Sge manual pages. http://gridscheduler.sourceforge.net/htmlman/manuals.html. (Accessed on 01/22/2021).

[9] 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.