SIMULATION AND ANALYSIS OF THE SEAWULF JOB SCHEDULER

Matthew Cimino **Caleb Kofahl** **Amanda Prechtl** **Adil Soubki**

December 10, 2021

ABSTRACT

In this report we simulate the job scheduler for Stony Brook's high performance computing cluster, SeaWulf, and use the results to make recommendations for improvement. This is done by modeling the cluster as a series of multi-server queueing systems, with a separate queue assigned to each of the cluster's major node partitions. Data from several thousand real jobs was collected by using the scheduler's `squeue` tool, which provided the submit time, start time, end time, partition, and number of nodes requested for each job. This data is then used to fit input distributions for a simulation program. After comparing baselines to ensure the simulation is similar to the observed data, we are then able to change the number of nodes in each partition and observe the effects to average delay and partition utilization. These results are then used to make qualitative recommendations to either reduce the number of nodes on a partition in order to save power or increase the number of nodes on a partition to reduce delay time. All code, graphs, and figures created for this project are available on GitHub.

1 Introduction

SeaWulf is a high performance computing cluster utilized by the students and faculty of Stony Brook University for research purposes. The cluster consists of several partitions, with each partition recieving a certain allocation of CPU or GPU nodes. A breakdown of the cluster's partitions can be seen in Figure 1 below. These nodes provide the resources neceessary for users to complete jobs relating to their research field. As jobs are requested by users, SeaWulf's Slurm Workload Manager allocates the nodes necessary to complete the job. However, if the nodes requested under a particular partition are not currently available, the job scheduler places the user into a queue. Therefore, SeaWulf's job scheduler can be thought of as a large, multi-server queueing system.

The primary goal of this project was to model the SeaWulf's Job Scheduler as a multi-server queueing system for each major partition, and then simulate the system under various conditions to ultimately find ways to improve the scheduler's current operations. The key performance measures to be observed are the average delay of a user for each partition, as well as the overall server utilization. In particular, we explored how these performance measures may change under the effects of altering the number of nodes available on each partition.

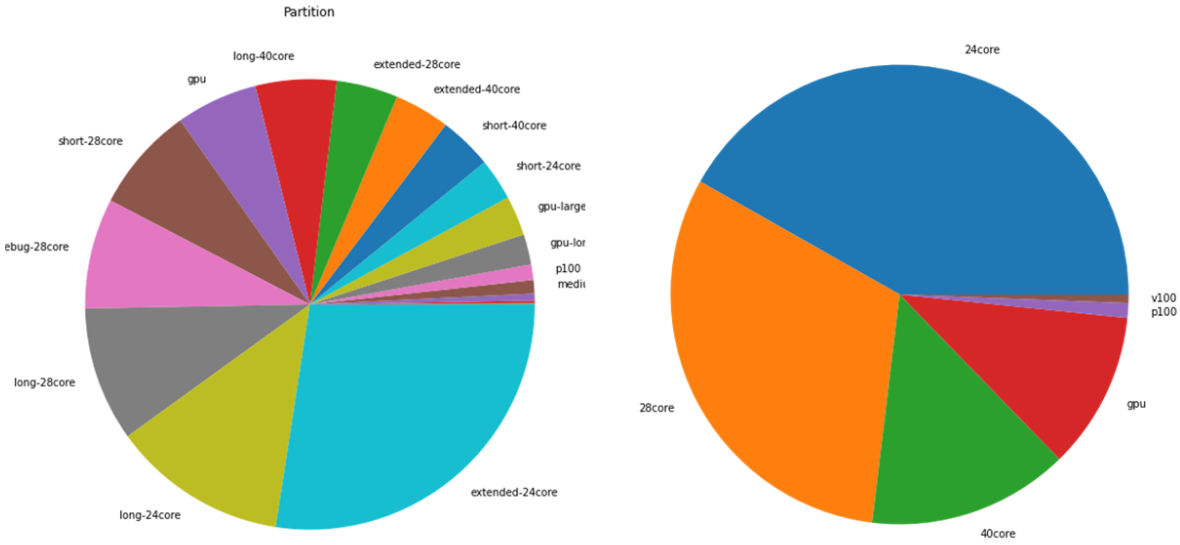

Figure 1: The chart on the left shows the breakdown of the different partitions available on the SeaWulf computing cluster, while the chart on the right shows a more generalized partition grouping used by our simulation model to account for the fact that partitions share underlying computing nodes. Note p100 and v100 were too small to be considered for this project.

2 Data Collection

The information most interesting for our purposes includes time spent waiting to start (delay), time spent running (service time), and the number of computing nodes required on a per job basis. While this data does exist somewhere in a log file written by the scheduler, it was not something we had access to.

Fortunately, there are other ways to collect the data needed for our simulation model. Slurm provides a number of tools for users of the cluster to submit and monitor jobs to the cluster. One such tool is called `squeue`, which provides a snapshot of information regarding jobs currently in the scheduler queue. We wrote a daemon to poll the output of this tool, merge data to avoid repetition, and record the results persistently to a file. By running this continuously over several weeks we were able to build a dataset containing information on thousands of jobs submitted to SeaWulf. This dataset would provide the foundation for fitting the input distributions to our simulation.

3 Input Analysis

Once the data was collected, we used Excel to convert it into useable data points. We filtered out the jobs that were pending, configuring, or running so that we were left with only jobs that had `STATE == COMPLETE`. This left us with just over 4,000 data points that we then filtered into the different partitions: 24core with approximately 1,800 points, 28core with approximately 1,200

points, 40core with approximately 550 points, and GPU with approximately 450 points. For the interarrival times - that is, the time between successive `SUBMIT_TIMES` - we converted the times into seconds, and for the service times - `END_TIME` minus `START_TIME` - we used hours.

3.1 Interarrival Times

The interarrival times varried greatly, from zero seconds between multiple jobs submitted by the same user at the same time, to over 24 hours between two jobs submitted to the 40core partition. However, in all partitions, the data was extremely skewed to the right. In fact, in the 24core partition, even the median is equal to zero. The other partitions were close, with all of them having a median of 240 seconds or less. Unfortunately, there was no clear partition between day and night so dividing the data into two groups to account for the large ranges was not practical.

Instead we focused on finding a theoretical distribution that best represents the data as a whole. We first tried an exponential distribution but it was far too steep a curve when overlayed with the data's histogram. So, we tried heavy-tailed distributions such as the log-normal distribution shown in Figure 2.

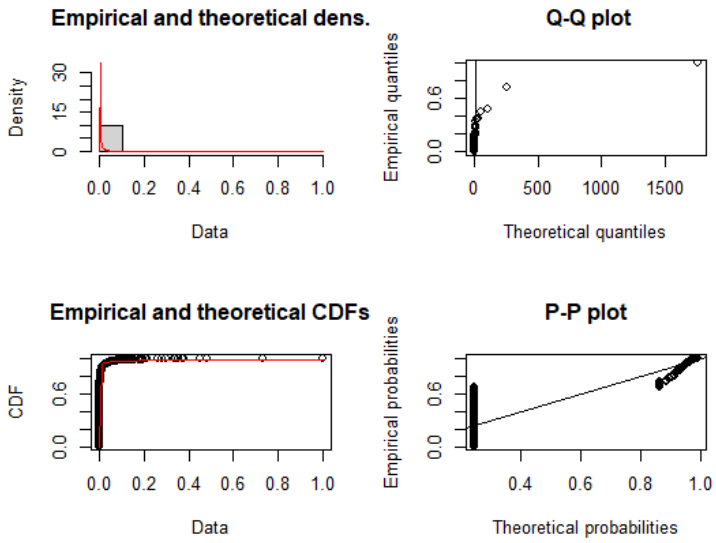

Figure 2: 24core Log-Normal Distribution

From the plots we can clearly see that a log-normal distribution is not a good fit for the 24core partition. It was also not a good fit for the 28core, 40core, or GPU partitions. The Weibull distribution also failed. We then used a Cullen and Frey Graph (Figure 3) to exempt some distributions by the parameters of skewness and kurtosis.

The beta and gamma distribution both showed some promise when looking at their plots (see Figure 4 for the 24core Beta Distribution), but neither passed any of the goodness of fit tests. In fact, the log-normal distribution had a higher Kolmogorov-Smirnov statistic than the beta and gamma distributions for all of the partitions despite having worse P-P and Q-Q plots.

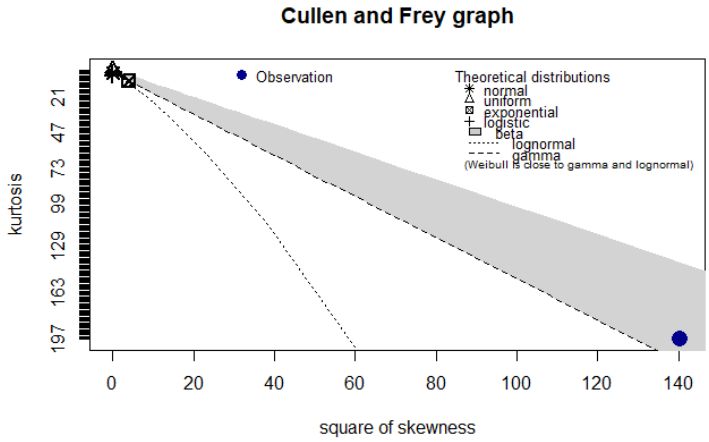

Figure 3: 24core Cullen and Frey Graph

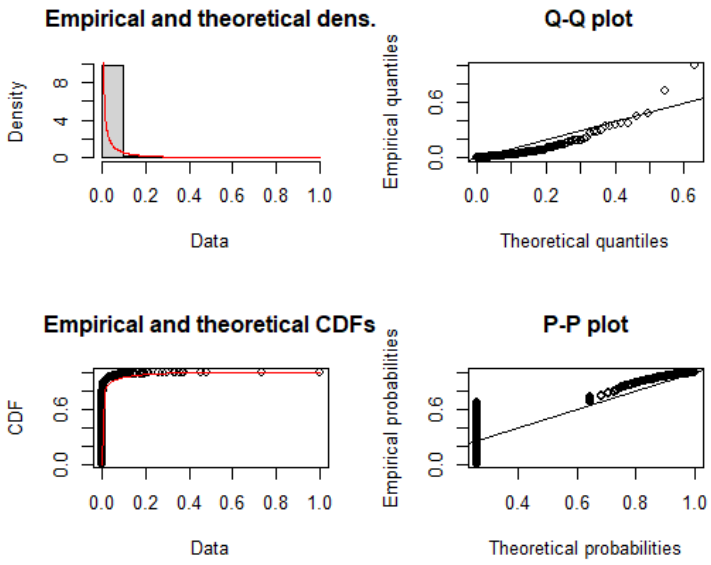

Figure 4: 24core Beta Distribution

Since the data did not fit any of the theoretical distributions, we made the decision to use the data to fit an empirical distribution. That is, we used a kernel densitity estimation (KDE) distribution to represent the interarrival times in the simulation. A downside of this is that because the distribution must be truncated, we potentially limit the range of possible sample values. But, as shown in Section 4, the KDE distribution performed better than expected in that it naturally mirrored the highs/lows of day/night.

3.2 Service Times

To analyze the distribution of service times for each partition, a script was written in Matlab that extracts the needed data for each partition, creates an intitial histogram, and then fits a certain

distribution based on the histogram. The fitted distribution then undergoes a chi-squared goodness of fit test, and is analyzed on a Q-Q plot. Finally, the script creates a sample data set using the fitted distribution and plots it alongside a normalized histogram of the original data for comparison. At first, several different theoretical distributions were attempted for each partition, including exponential, Weibull, Pareto, gamma, and log-normal. However, none of these distributions provided a decent fit, and could not pass the chi-squared test performed by the script.

One major issue common to all partitions was a significant spike in frequency for jobs taking over 160 hours to complete. After looking into these spikes, it was found that a significant number of jobs had issues which caused them to "hang" in service indefinitely. These jobs were eventually kicked out of the system after 7 days (168 hours). Therefore, we discovered that the system has a maximum service time set in place, which allowed for a more justified use of KDE distributions compared to the interarrival times. Seeing as the theoretical distributions still did not fit well after removing these "hanging" jobs, it was decided to use KDE distributions for each partition.

To ensure the KDE distributions provided the desired linear Q-Q plot structure, very small service times had to be removed from the data set. The thresholds for minimum service times for each partition are: 7.2 minutes for 24 core, 4.2 minutes for 28 core, 2.4 minutes for 40 core, and 1.2 minutes for GPU. Figures 5 through 8 below show the fitted histogram for each partition along with its respective Q-Q plot.

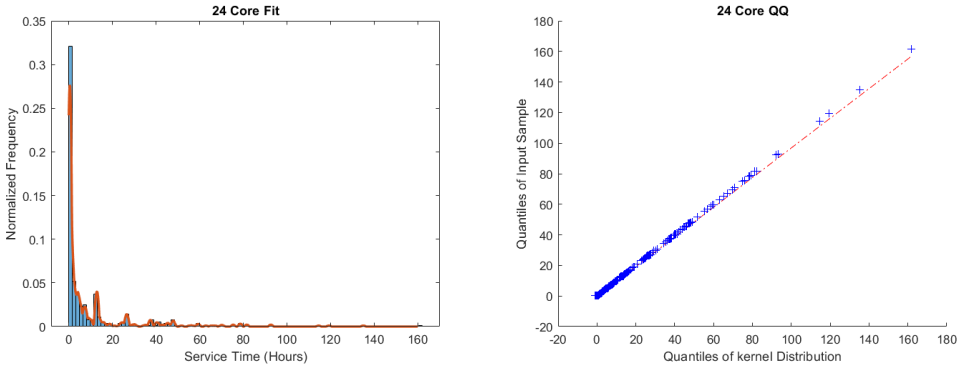

Figure 5: 24core Service Time Fit and Q-Q plot

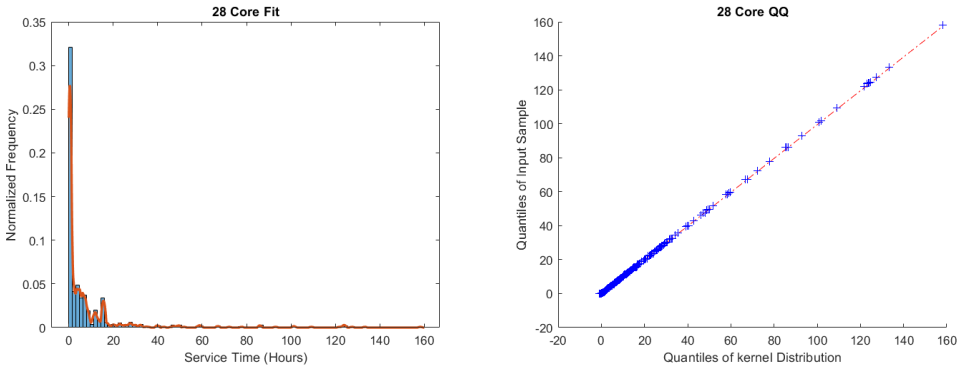

Figure 6: 28core Service Time Fit and Q-Q plot

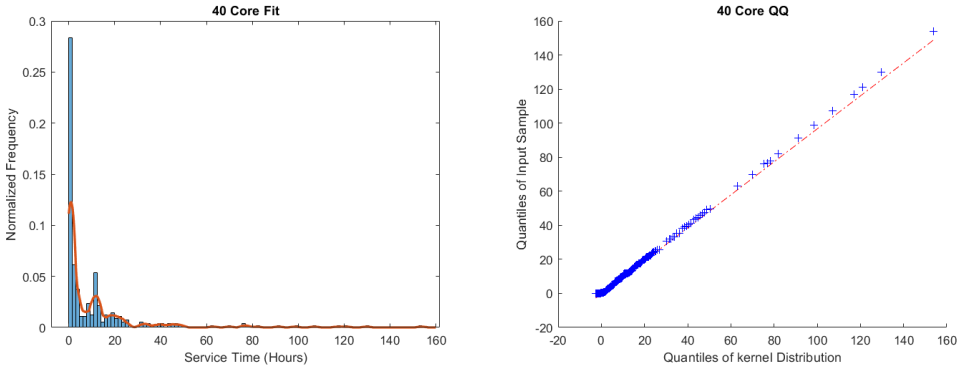

Figure 7: 40core Service Time Fit and Q-Q plot

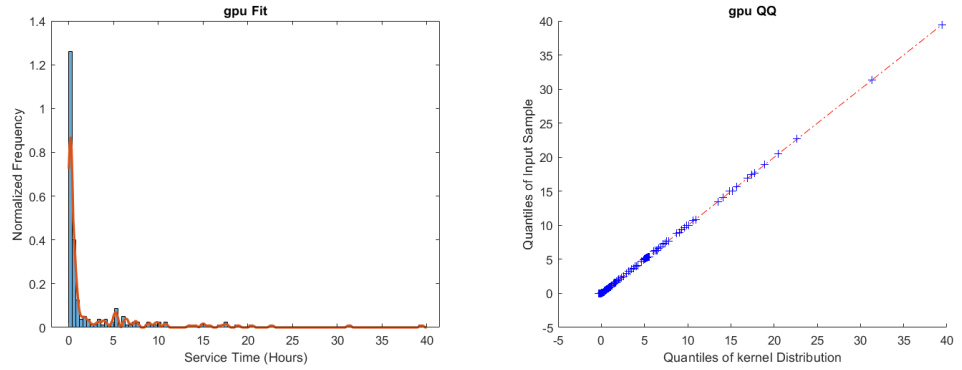

Figure 8: GPU Service Time Fit and Q-Q plot

3.3 Nodes Requested

A distribution for the number of nodes requested by a particular job was not initially used as the data collected from `squeue` did not seem to indicate it was necessary. However, upon closer inspection it became clear that there was an issue with collection. The documentation notes this strange behavior.

> "In the case of a `COMPLETING` job, the list of nodes will comprise only those nodes that have not yet been returned to service." [1]

Because our tool was polling for the latest data, this meant what we had initially collected was not representative. After modifying the tool to account for this and gathering new data it became clear that this information would need to be used as an input distribution to the simulation.

Similar to the service time distribution, no theoretical distribution matched the data collected for any partition. However, since there is a relatively small range of values for the number of nodes that can be requested on any partition, we mitigate one of the biggest downsides to using an empirical distribution. As a result, we fit an empirical distribution and linearly interpolate between values. Each time a job is created in the simulation, we will draw from this distribution to determine the resources that job will request.

[1] https://slurm.schedmd.com/squeue.html

4 Simulation Construction

We broke the simulation code into two sections; samplers and the simulator. For sampling the interarrival times, service times, and number of requested nodes distributions, we used the KDE from the `statsmodels` Python library. We then interpolated a sample distribution using `scipy`'s interpolate library. The mean, minimum, and maximum values for the gathered and sampled data can be seen in Figure 9. In addition, we calculated the server utilization and average number of nodes in use for each dataset. From these values, it is clear that the sampled data closely mimics that of the originally gathered data. Thus, the samplers will provide decent approximations during our simulation runs.

```
Partition: 24core
Capacity: 99
mean/min/max interarrival time: 0.19974 | 0.0 | 20.916
mean/min/max service time: 4.6066 | 0.0 | 168.01
mean/min/max delay time: 1.1352 | 0.0 | 138.89
mean/min/max number of requested nodes: 1.9834 | 1 | 16
Average nodes in use: 45.742072102066956
Average utilization: 0.4620411323441107

Partition: 28core
Capacity: 146
mean/min/max interarrival time: 0.26512 | 0.0 | 28.044
mean/min/max service time: 8.7831 | 0.0 | 168.01
mean/min/max delay time: 1.4535 | 0.0 | 166.95
mean/min/max number of requested nodes: 1.9834 | 1 | 16
Average nodes in use: 65.7056300843766
Average utilization: 0.45003856222175753

Partition: 40core
Capacity: 64
mean/min/max interarrival time: 1.0403 | 0.0 | 288.41
mean/min/max service time: 12.988 | 0.0 | 168.01
mean/min/max delay time: 6.4177 | 0.0 | 465.68
mean/min/max number of requested nodes: 1.9834 | 1 | 16
Average nodes in use: 24.76161257648534
Average utilization: 0.38690019650758345

Partition: gpu
Capacity: 9
mean/min/max interarrival time: 0.64185 | 0.00027778 | 19.021
mean/min/max service time: 2.5811 | 0.0 | 48.006
mean/min/max delay time: 0.29077 | 0.0 | 8.3258
mean/min/max number of requested nodes: 1.9834 | 1 | 16
Average nodes in use: 7.975904286649329
Average utilization: 0.886211587405481
```

```
Partition: 24core
Capacity: 99
mean/min/max interarrival time: 0.19998 | 0.0 | 20.875
mean/min/max service time: 4.5368 | 8.7826e-07 | 168.01
mean/min/max number of requested nodes: 1.9689 | 1 | 16
Average nodes in use: 44.66601374343163
Average utilization: 0.45117185599425885

Partition: 28core
Capacity: 146
mean/min/max interarrival time: 0.26676 | 0.0 | 28.04
mean/min/max service time: 8.7484 | 0.0 | 168.01
mean/min/max number of requested nodes: 1.9768 | 1 | 16
Average nodes in use: 64.82777146483713
Average utilization: 0.4440258319509392

Partition: 40core
Capacity: 64
mean/min/max interarrival time: 1.0697 | 0.0 | 288.38
mean/min/max service time: 12.882 | 0.0 | 168.01
mean/min/max number of requested nodes: 1.9702 | 1 | 16
Average nodes in use: 23.72511723551543
Average utilization: 0.3707049568049286

Partition: gpu
Capacity: 9
mean/min/max interarrival time: 0.63725 | 0.00027986 | 19.002
mean/min/max service time: 2.613 | 6.2106e-07 | 48.006
mean/min/max number of requested nodes: 1.8184 | 1 | 9
Average nodes in use: 7.456043405249636
Average utilization: 0.8284492672499595
```

Figure 9: Mean, minimum, maximum, average nodes in use, and average utilization of the collected data (left) and sampled data (right).

After researching simulation libraries, we decided to utilize the `simpy` Python library to simulate the job scheduler.[2] `simpy` allowed us to create a simulation object containing all the functionality needed to simulate our desired queueing system. This simulation object contains "resources", in our case compute nodes, that would act as the servers in our queueing system. Given a specific time horizon, the simulator generates an interarrival time, service time, and requested number of nodes, all drawn from our previously designed samplers. In addition, the simulation was designed such that the next job in the queue cannot begin until there are enough nodes available on that partition to service that job.

[2] https://simpy.readthedocs.io/en/latest/

5 Output Analysis and Results

Once the simulated scheduler was written, we began analyzing the resulting queueing system data. We used the Replication/Deletion Approach by removing the first 1,000 time steps from our simulation runs. From this data, it was apparent that the 24core and 28core partitions are not utilizing all available resources. Thus, it may be prudent to reduce energy costs by shutting down some of the nodes that aren't being fully utilized. Figures 10 and 11 show the resulting simulation plots using the actual number of available nodes and the case where that number is reduced by 14 nodes and 31 nodes for the 24core and 28core partitions, respectively.

In addition, we calculated the same statistics as before to compare the two simulations (see figure 12). In the case of the 24core partition, we went from a delay time of 2 minutes up to a delay time of 24 minutes, which is still somewhat reasonable. For the 28core case, in the initial simulation no queue ever formed. From the user's perspective this is great, but from the provider's perspective, that is a lot of under utilized resources that they are paying money for. In the simulation with 31 fewer nodes, we see a delay time of just over 15 minutes – very reasonable considering we removed 21% of the available nodes.

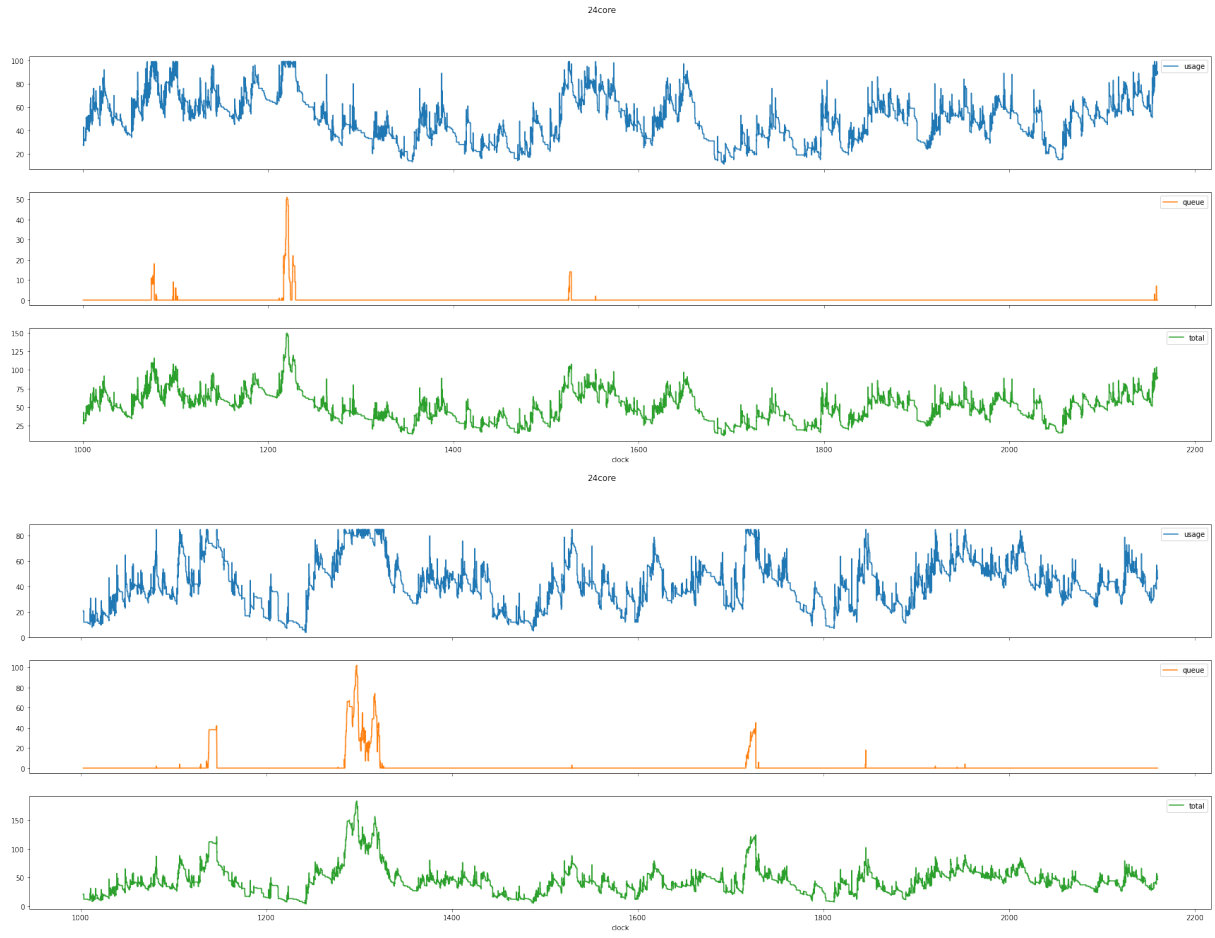

Figure 10: Plots of utilization (blue/top), queue size (orange/middle), and total (green/bottom) for the original simulation (top) and the simulation with reduced resources (bottom) for the 24core partition.

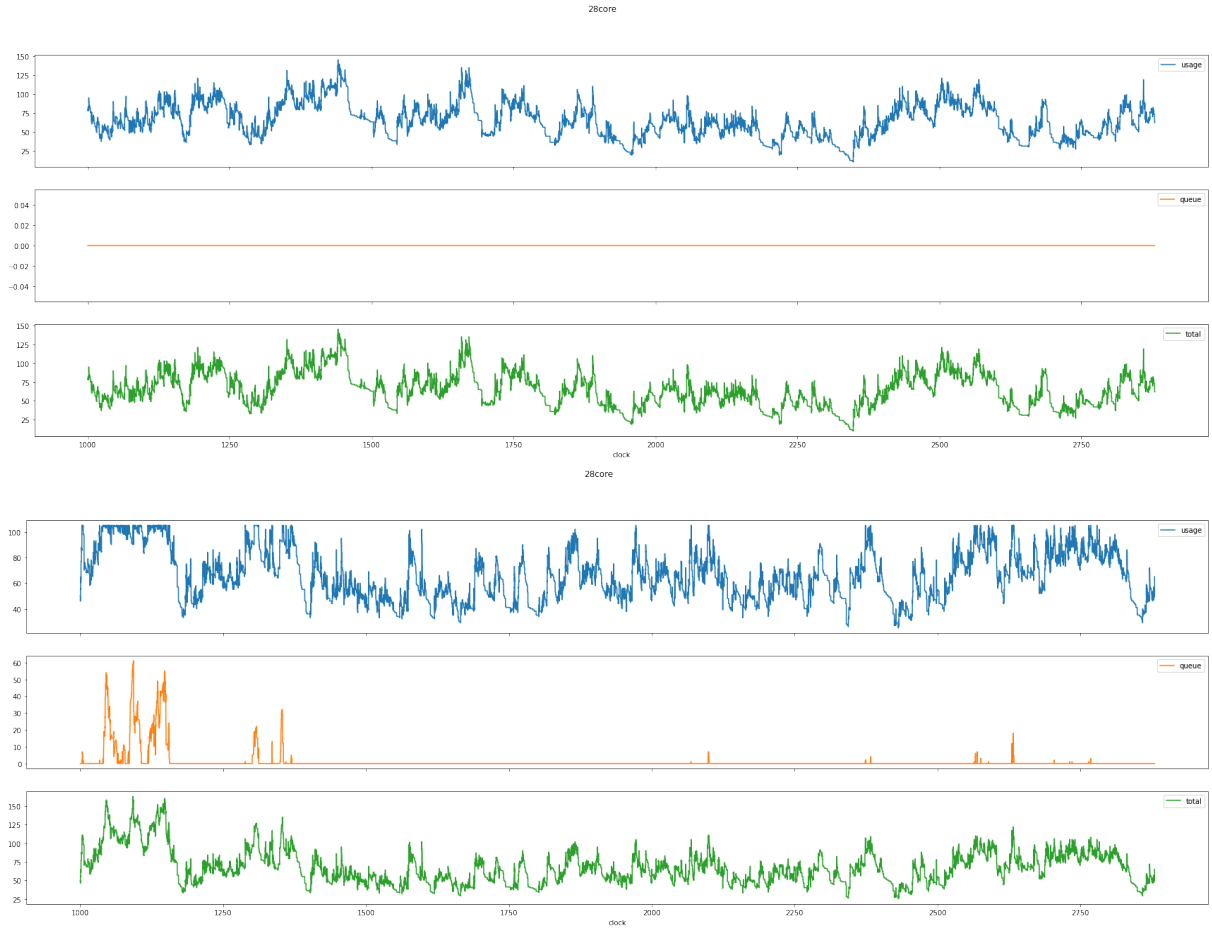

Figure 11: Plots of utilization (blue/top), queue size (orange/middle), and total (green/bottom) for the original simulation (top) and the simulation with reduced resources (bottom) for the 28core partition.

```
Partition: 24core
Original Simulation (n=10682):
Capacity: 99
--------------------------------------------------
mean/min/max interarrival time:         0.2029 | 0.0 | 20.624
mean/min/max delay time:                0.035982 | 0.0 | 5.0848
mean/min/max service time:              4.783 | 2.1566e-05 | 168.01
mean/min/max number of requested nodes: 1.951 | 1 | 16
Average nodes in use: 45.99132271476446
Average utilization: 0.464558815300651
--------------------------------------------------
Modified Capacities Simulation (n=10584):
Capacity: 85
--------------------------------------------------
mean/min/max interarrival time:         0.20423 | 0.0 | 20.806
mean/min/max delay time:                0.39783 | 0.0 | 12.115
mean/min/max service time:              4.5108 | 3.294e-05 | 168.01
mean/min/max number of requested nodes: 1.9545 | 1 | 16
Average nodes in use: 43.16846846967984
Average utilization: 0.5078643349374098
```

```
Partition: 28core
Original Simulation (n=10596):
Capacity: 146
--------------------------------------------------
mean/min/max interarrival time:         0.27408 | 0.0 | 28.007
mean/min/max delay time:                0.0 | 0.0 | 0.0
mean/min/max service time:              8.5559 | 0.0 | 168.01
mean/min/max number of requested nodes: 1.9811 | 1 | 16
Average nodes in use: 61.844283916161785
Average utilization: 0.4235909857271355
--------------------------------------------------
Modified Capacities Simulation (n=10798):
Capacity: 105
--------------------------------------------------
mean/min/max interarrival time:         0.26679 | 0.0 | 27.969
mean/min/max delay time:                0.25826 | 0.0 | 10.567
mean/min/max service time:              9.3321 | 0.0 | 168.01
mean/min/max number of requested nodes: 1.9837 | 1 | 16
Average nodes in use: 69.38845496395072
Average utilization: 0.6608424282281021
```

Figure 12: Statistics from runs of our simulation with the actual resources vs. reduced resources for the 24core (left) and 28core (right) partitions.

In contrast, for the case of the 40core partition we see that the average delay is over 90 minutes. By simply adding 11 nodes to this partition, the delay can be reduced to just under 15 minutes. For the GPU partition however, the original simulation had a queue that increased to infinity. To reduce the

average delay down to just over 15 minutes we had to increase the number of available nodes to 30, an increase of 233%. This data shows that the GPU partition is grossly overused and to properly service the current demand, more than twice as many nodes have to be added. Figures 13 and 15 show the plots of the original simulation and the one with increased resources for the 40core and GPU partitions, respectively.

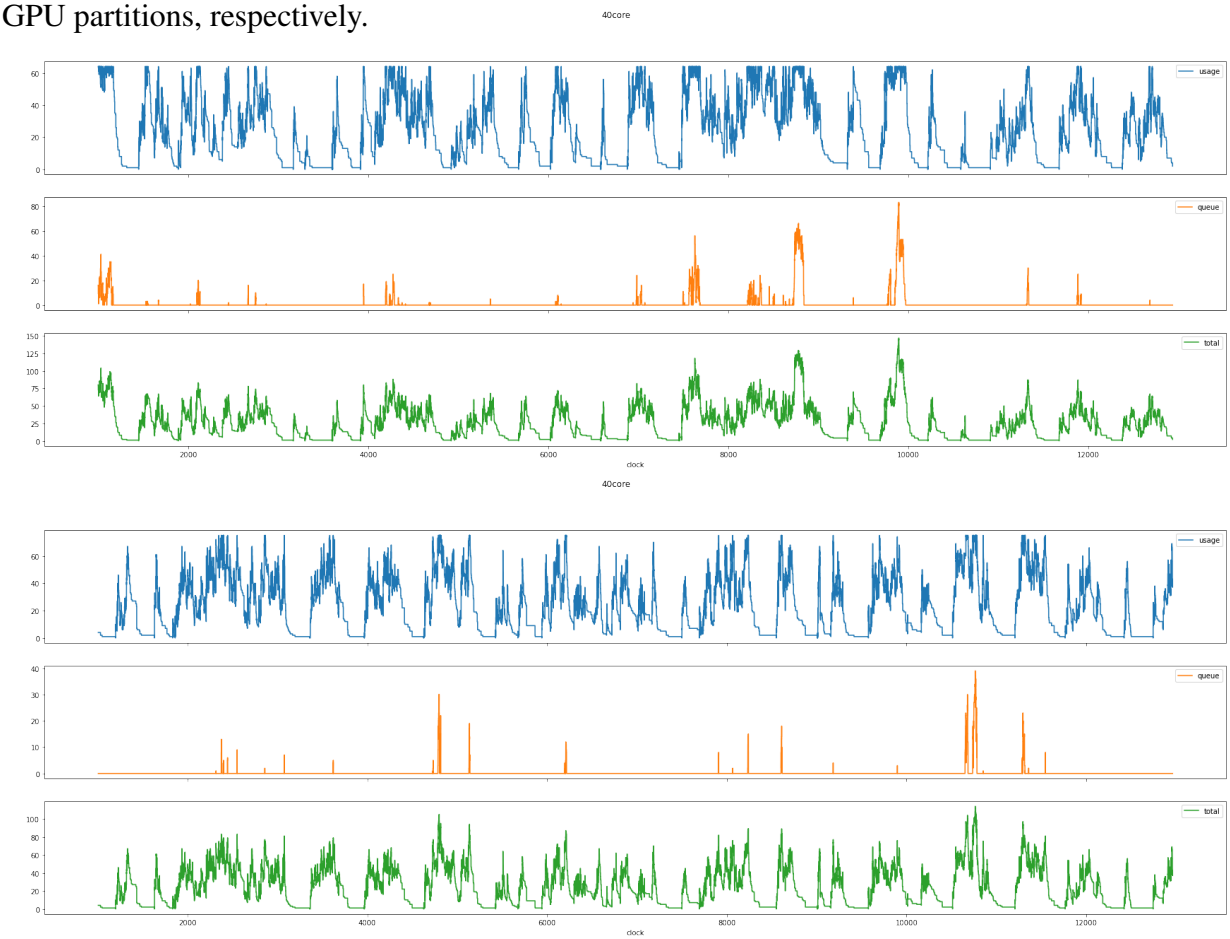

Figure 13: Plots of utilization (blue/top), queue size (orange/middle), and total (green/bottom) for the original simulation (top) and the simulation with increased resources (bottom) for the 40core partition.

```
Partition: 40core
Original Simulation (n=11613):
Capacity: 64
--------------------------------------------------
mean/min/max interarrival time:         1.1267 | 0.0 | 287.88
mean/min/max delay time:                1.6137 | 0.0 | 38.916
mean/min/max service time:              13.153 | 0.0 | 168.01
mean/min/max number of requested nodes: 1.9597 | 1 | 16
Average nodes in use: 22.87760292284022
Average utilization: 0.35746254566937846
--------------------------------------------------
Modified Capacities Simulation (n=11056):
Capacity: 75
--------------------------------------------------
mean/min/max interarrival time:         1.1723 | 0.0 | 288.19
mean/min/max delay time:                0.24207 | 0.0 | 23.036
mean/min/max service time:              13.189 | 0.0 | 168.01
mean/min/max number of requested nodes: 1.959 | 1 | 16
Average nodes in use: 22.041534101282302
Average utilization: 0.2938871213504307
```

```
Partition: gpu
Original Simulation (n=11366):
Capacity: 9
--------------------------------------------------
mean/min/max interarrival time:         0.63369 | 0.00028216 | 19.006
mean/min/max delay time:                748.04 | 0.0 | 1381.0
mean/min/max service time:              2.6263 | 1.5868e-06 | 48.006
mean/min/max number of requested nodes: 1.8127 | 1 | 9
Average nodes in use: 7.512605692924325
Average utilization: 0.8347339658804805
--------------------------------------------------
Modified Capacities Simulation (n=11018):
Capacity: 30
--------------------------------------------------
mean/min/max interarrival time:         0.65349 | 0.0002968 | 18.924
mean/min/max delay time:                0.25421 | 0.0 | 20.345
mean/min/max service time:              2.5947 | 2.3701e-06 | 48.006
mean/min/max number of requested nodes: 1.9765 | 1 | 16
Average nodes in use: 7.8477584131536045
Average utilization: 0.2615919471051202
```

Figure 14: Statistics from runs of our simulation with the actual resources vs. reduced resources for the 40core (left) and GPU (right) partitions.

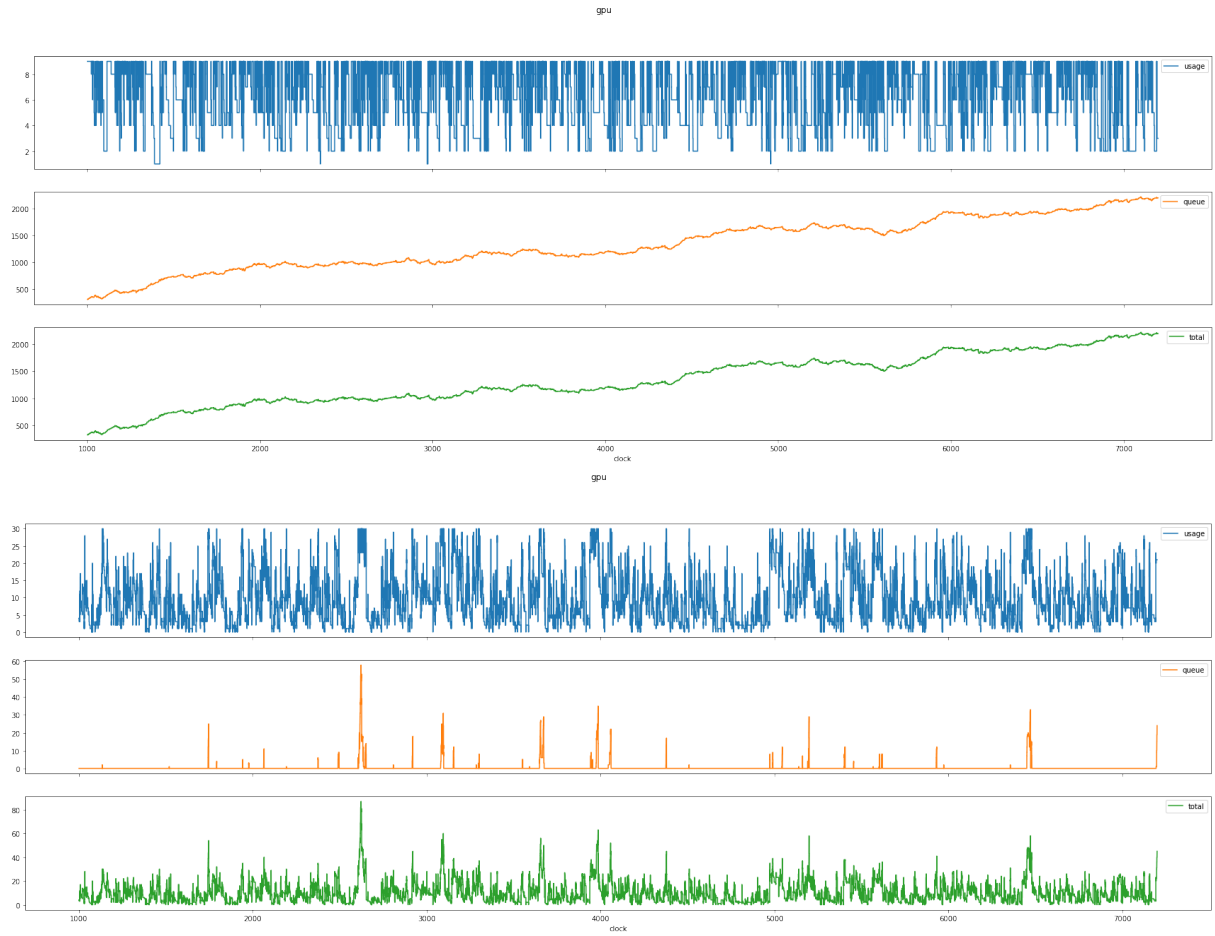

Figure 15: Plots of utilization (blue/top), queue size (orange/middle), and total (green/bottom) for the original simulation (top) and the simulation with increased resources (bottom) for the GPU partition.

6 Conclusion

After simulating SeaWulf's job scheduler by modeling the cluster as a series of multi-server queueing systems, with each of the major partitions (24core, 28core, 40core, and GPU) having their own queues, we were able to formulate the following qualitative recommendations:

1. For the 24core and 28core partitions, some nodes should be removed or shutdown to minimize energy costs.

2. For the 40core and GPU partitions, nodes should be added to decrease the delay time to a far more reasonable range.

These were formulated by collecting data from over 4,000 real jobs to create input distributions for a simulation program. Using this simulator, we were able to adjust the number of nodes, observe the effects, and form the proposed modifications presented in Section 5.

6.1 Future Improvements

There are several improvements to this project that we were unable to complete due to time constraints. Ideally, we would have been able to perform more replications for each capacity simulated on the various partitions. This would improve the accuracy of our results at the cost of additional computation. Similarly, simulating more capacities for each partition would allow us to plot how our output parameters, like average delay time and utilization, vary at a more granular level. Another improvement would be to perform more thorough statistical testing, perhaps by an ANOVA test, to ensure our simulation is producing output statistics which are not significantly different from the empirical data. Finally, we would like to implement variance reduction techniques to improve the statistical efficiency of our program.