Analysis of Historical HPC Job Data

Bachelor Thesis

University of Basel
Faculty of Science
Department of Mathematics and Computer Science
HPC Research Group

Examiner: Prof. Dr. Florina M. Ciorba
Supervisor: Thomas Jakobsche

Author: Katja Keller
Email: katja.keller@stud.unibas.ch

December 30, 2022

Acknowledgements

First of all, I would like to thank Prof. Dr. Florina Ciorba for allowing me to write my bachelor thesis in her research group. I am extremely grateful for the manner in which she supervised the team and for sharing her thoughts and insights regarding my work. Next, I want to thank my supervisor Thomas Jakobsche for his exceptional work. I am thankful for all the time and advice he offered me, for never hesitating to do research whenever I encountered a problem he did not know the solution to, and for his general support throughout this thesis. Furthermore, I would like to thank the entire HPC Research Group, as the biweekly meetings and their insights were always very helpful. Last but not least, I want to thank my family and friends, who encouraged and supported me during the whole process.

Abstract

HPC systems are designed to give scientific applications the computing capacity they need to address scientific issues. They are preferred to run smoothly and efficiently because they are expensive and use a lot of energy. To improve and maintain performance, resilience, and energy efficiency of HPC systems, operators need to understand how users are interacting with the system, specifically job scheduling and resource allocation. For instance, how accurate are time limit estimates, when are jobs submitted, and how many of these jobs get completed? In this thesis, HPC Slurm job data is statistically examined using graphs and tables to address the above-mentioned and other research questions. We analyze three months of historical job data for the miniHPC system operated for teaching and research purposes by the HPC Group of the University of Basel. We identify potential ways to enhance job scheduling, job anomaly detection, and how to provide user feedback to improve and maintain performance, resilience, and energy efficiency of HPC systems.

Contents

Chapter 1

Introduction

1.1 Motivation

In order to solve scientific problems faster, HPC practitioners and researchers need efficient HPC systems. To achieve this, one has to know how the users utilize these systems. The most efficient way to do this is to analyze the data of the jobs executed by the users. Since the Slurm sacct data is vast and looking at individual jobs is not feasible, we need to have statistical plots and tables that focus on specific aspects, such as job arrivals times and exit codes. By doing this, we can investigate particular possibilities for improving HPC job scheduling and job anomaly detection while also providing job user feedback.
Analyses like this have been made for numerous other HPC systems. But these analyses are only of limited use in our case since job data analysis is often system specific.

1.2 Goal

The goal is to improve the usage of the HPC system to make it run more efficiently. We analyze the miniHPC's job data in order to find patterns, which can then be used to improve the HPC system usage. This includes making job scheduling easier so that the resources can be used more efficiently, providing user feedback so that the users know why their jobs are constantly failing or timing out, which then minimizes the occurrence of failures and time outs, and predicting the jobs' end states when using a particular submit line, which helps not only with job scheduling but also with saving resources.

1.3 Research Questions

In order to achieve the goal, the HPC job data has to be visualized and analyzed. To know what charts and tables have to be created, we need to determine what exactly we want to know.
We ask the following research questions to reach this thesis' goal:

- What are the most used job sizes?
- Which user runs jobs of what size?

- At what time do the jobs arrive?
- Which exit codes appear how often?
- What user's jobs fail with which exit code?
- How good is the time limit usage?
- How long are the jobs' average and overall job execution times when looking at different users and job sizes?
- Are there any consistent job characteristics (jobs with the same name have a similar job execution time or jobs executed using a particular submit line terminate in the same state)?

1.4 Solution

To answer these questions, we first extract the Slurm HPC job data by using the sacct command. This is done once for every month (September, October and November of the year 2022) and then once again for the three of them combined. This data then gets stored in text files which can be imported into Excel. Once imported, a few things have to be added manually. For example, we add a column stating whether a job was executed on a weekday or the weekend. There are also a lot of entries that have to be deleted, as they are faulty or just not important for this thesis. After these steps, the result are four tables, with the biggest one of them made out of twelve columns and 8181 rows for the whole three months.
With these tables as the basis, numerous pivot tables and charts are created, combining different aspects of the job data. For example, when looking at the users and the job sizes, a pivot chart showing which user executed jobs of what sizes can be generated. Afterwards, these tables and graphs can be analyzed and used to answer the research questions asked in section 1.3.

1.5 Outline

After the introduction, this work will contain a chapter discussing the works related to this thesis. The next chapter, chapter 3, consists of the methods used to complete this work. Before explaining why we used Microsoft Excel to create pivot tables and charts, we will talk about the Slurm Workload Manager, the sacct command, and the job accounting data we used in this thesis. The last part of this chapter will contain the experimental setup and parameters. The results will be displayed in chapter 4 and then discussed in chapter 5. In the last chapter, which is Chapter 6, we will draw a conclusion and discuss future research possibilities.

1.6 HPC

To solve advanced computing problems at high speed, high performance computing (HPC) is used. Thanks to its ability to use parallel data processing, complex calculations can be performed, and the computing performance is improved. This leads to lower costs, as time and money can be saved. [1]

1.6.1 miniHPC

miniHPC is the name of a small high-performance computing cluster funded by the University of Basel. It is used for teaching students parallel programming and as a fully-controlled experimental platform for conducting cutting-edge HPC scientific research. miniHPC contains four different types of nodes. The Intel Xeon E5-2640 nodes include 22 computing nodes, one login node, and one node for storage. There are four Intel Xeon Phi computing nodes, while both the Intel Xeon Gold 6258 nodes and the AMD EPYC 7742 nodes are computing nodes. [2]

Chapter 2

Related Work

2.1 Workload Modeling for Computer Systems Performance Evaluation

In his book *Workload Modeling for Computer Systems Performance Evaluation*, Dror G. Feitelson explains that workload modelling relies on the analysis of workload data. And analyzing is exactly what he does. One of his examples demonstrates how he analyzes an accounting log. Based on this log, he examines the activity on the 128-node NASA Ames iPSC/860 hypercube supercomputer. He illustrates various aspects with graphs, including the relationship between job runtimes and size and the distribution of job sizes. In his analysis of these plots, he gives them significance.

This thesis is heavily based on Feitelson's work. But, while we use his examples as an inspiration and basis for our visualizations, he also analyzes some aspects that are of no interest to us. One example would be his analysis of the distribution of job sizes according to type (direct interactive jobs versus NQS batch jobs). What we add are user-specific visualizations, as we want to know every user's behaviour on the system. [3]

2.2 HPC Job-Monitoring with SLURM, Prometheus and Grafana

For his thesis *HPC Job-Monitoring with SLURM, Prometheus and Grafana*, Pascal Kunz installed and configured Prometheus[1] and Grafana[2]. Prometheus is a tool to monitor time-series data, while Grafana can be used to visualize data. By doing this, he granted the members of the HPC research group a way to monitor the miniHPC. He created software that collects data from Slurm and cleans it. This data then gets added to the Prometheus database. He then further ensured that this data could be visualized in Grafana and that job anomalies were studied, so they could then get captured. [4]

[1] https://prometheus.io/
[2] https://grafana.com/

2.3 Visual Analysis of Job Accounting Information of High Performance Computing Systems

In his bachelor thesis, *Visual Analysis of Job Accounting Information of High Performance Computing Systems*, Aldi Dyla created a platform named *HPC Workload Visual Analysis Platform*, which grants users a way to visualize their job accounting data to simplify the analysis of said data. He made use of various metrics (simple and derived metrics), such as system size, job turnaround time, waiting time (per user/group), and a lot more. In chapter 5, he presents the platform's results by uploading two log data files, which are then visualized and analyzed. [5]
Even though his work is about visualizing and analyzing accounting or log data, it is very different from what we are doing in this bachelor thesis. While he searched for a way to visualize the data, we are actually interested in the results of this visualization, which we want to use to enhance the HPC system's performance. Additionally, we are also analyzing different metrics than he did.

Chapter 3

Methods

3.1 Slurm Workload Manager

The name Slurm is an acronym for "Simple Linux Utility for Resource Management". It is frequently referred to as simply Slurm. The Slurm Workload Manager is a free and open-source Linux and Unix-like kernel task scheduler, which is used by many supercomputers and computer clusters.

Besides granting people temporary exclusive or non-exclusive access to resources and providing a framework for initiating, carrying out, and keeping track of work, it also manages a queue of open jobs to arbitrate resource contention.
Slurm uses a best-fit approach based on fat-tree network topology or Hilbert curve scheduling to maximize the locality of task assignments on parallel computers.

In this work, Slurm is used to collect the HPC job data we want to analyze. [6]

3.2 sacct Command

Slurm either logs accounting information for jobs it launches in the job accounting log file or stores it in the Slurm database. For one's study, the sacct command presents job accounting data in several formats that are saved in the job accounting log file or Slurm database. By default, the sacct command shows details about jobs, exit codes, and more. [7]

Here, we work with the sacct command of SLURM to display accounting data from the past.

3.3 Used Job Accounting Data

Table 3.1 depicts the used job status fields, with corresponding examples and descriptions.

Job Accounting Data		
Job Status Field	Example	Description
JobID	858490	The identification number of the job or job step.
User	user2	The user name of the user who ran the job.
State	COMPLETED	Displays the job status, or state.
Start	2022-10-01T17:56:22	Initiation time of the job.
Elapsed	00:00:01	The job's elapsed time.
AllocCPUS	40	Count of allocated CPUs.
Timelimit	01:00:00	What the timelimit was/is for the job.
ExitCode	0:0	The exit code returned by the job script.
JobName	nas	The name of the job or job step.
SubmitLine	sbatch jobscript	The full command issued to submit the job.
WorkDir	/users/stud/j/user2	The directory used by the job to execute commands.

Table 3.1: Table showing the Used Job Accounting Data [8]

3.4 Microsoft Excel Pivot Table and Chart

The tables and plots used in this thesis were created using Microsoft Excel Pivot Tables and Microsoft Excel Pivot Charts. A pivot table is an interactive table composed of rows, columns, pages, and data fields. It can easily be used on already existing tables, which in our instance, is the table storing the HPC job data. The user has a lot of flexibility with these pivot tables, including the ability to program equations and move around the data. It is then effortless to compare the information and analyze the data. Furthermore, it is possible to manage vast amounts of data with ease. When creating a pivot table, the user has the option to build a pivot chart based on the table's data. Every change applied to the table is immediately reflected in the chart.

3.5 Experimental Setup and Parameters

In this thesis, the HPC job data from all the users within the months of September, October, and November of the year 2022 is being analyzed. All the plots exist on a monthly basis and as a combination of the three months. To ensure data security, the users' real names have been changed to user1, user2, and so on up until user16.

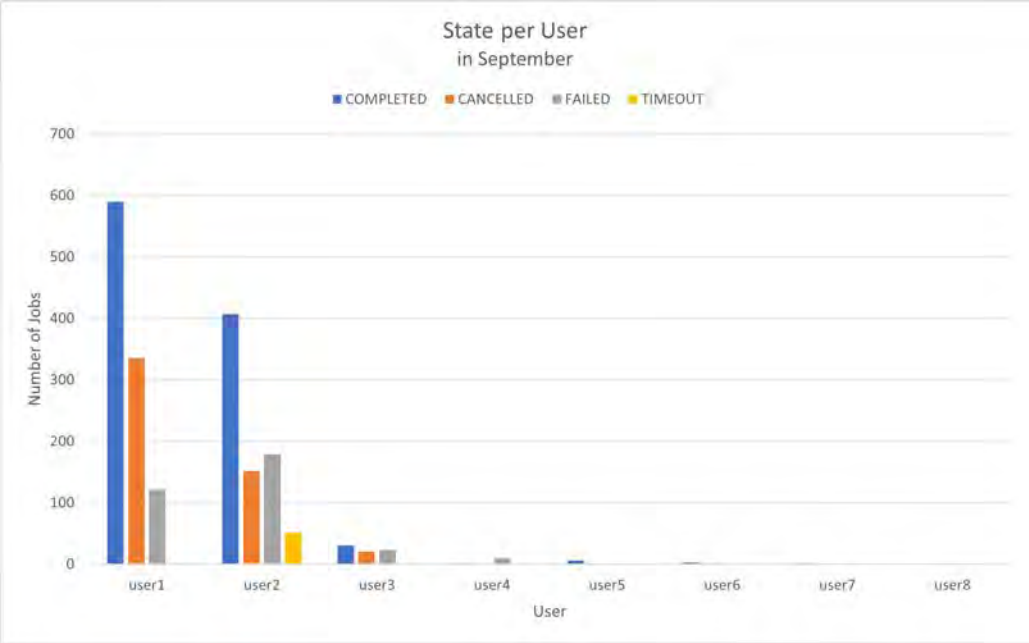
Chapter 4

Results

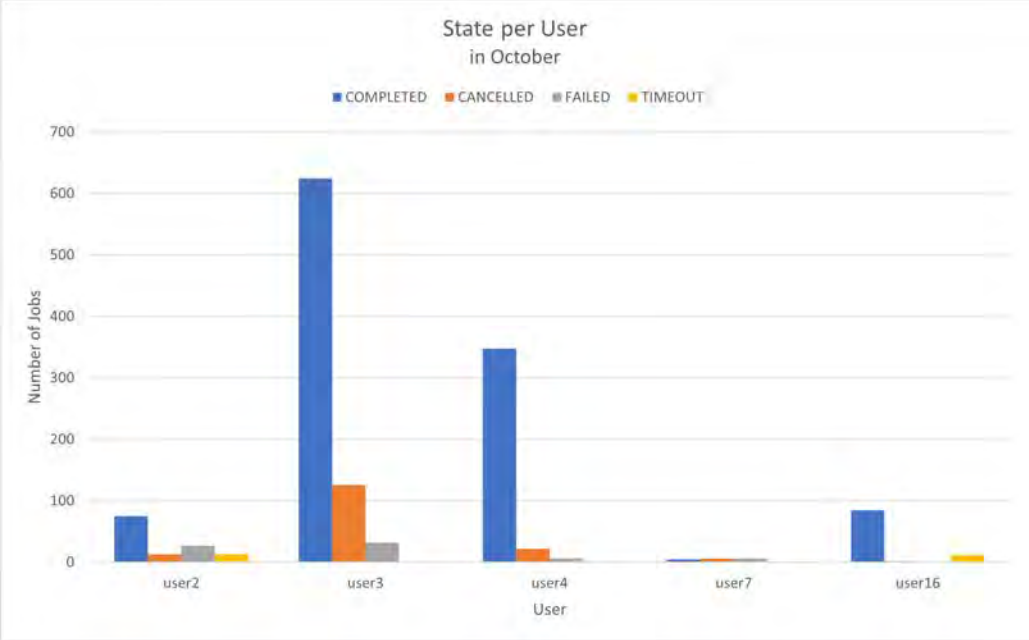
In this chapter, the results are displayed as charts and tables.

4.1 State per User

Figure 4.1 depicts the state of user-executed jobs. The horizontal axis identifies the various users, while the vertical axis indicates the number of jobs. As seen in (a), in September, most jobs were run by user1 and user2, with most of their jobs being completed.
In October, user3 executed the most jobs. Out of 783 jobs, only 32 failed, while not a single one timed out. User4 has even more impressive statistics, with only six failures out of 376 completed jobs. This is shown in (b).
As (c) demonstrates, not a single executed job timed out in November. User3 ran the most jobs once again; however, this month, almost a third of them resulted in a fail.
An overview of the three mentioned months can be seen in (d). This overview clarifies how many more jobs user3 executed compared to the other 15 users.

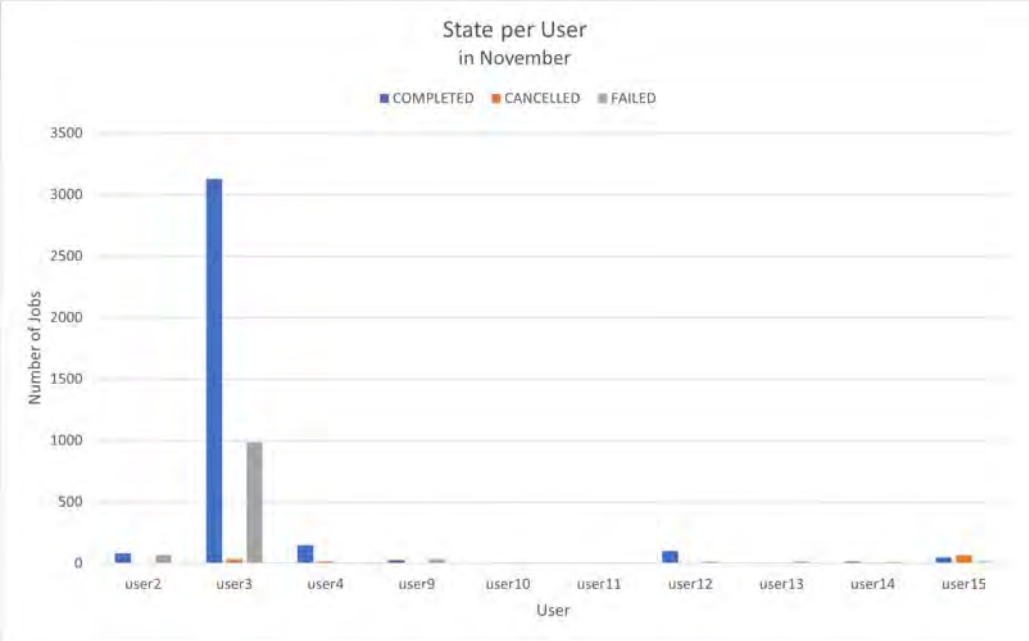


(a) State per User in September

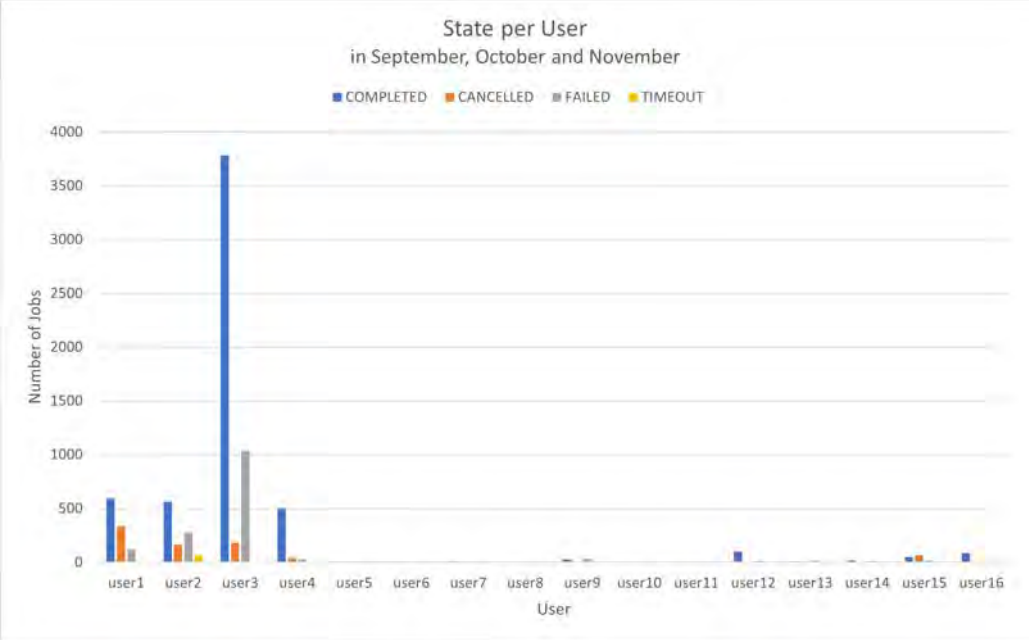


(b) State per User in October

Figure 4.1: Charts showing State per User



(c) State per User in November

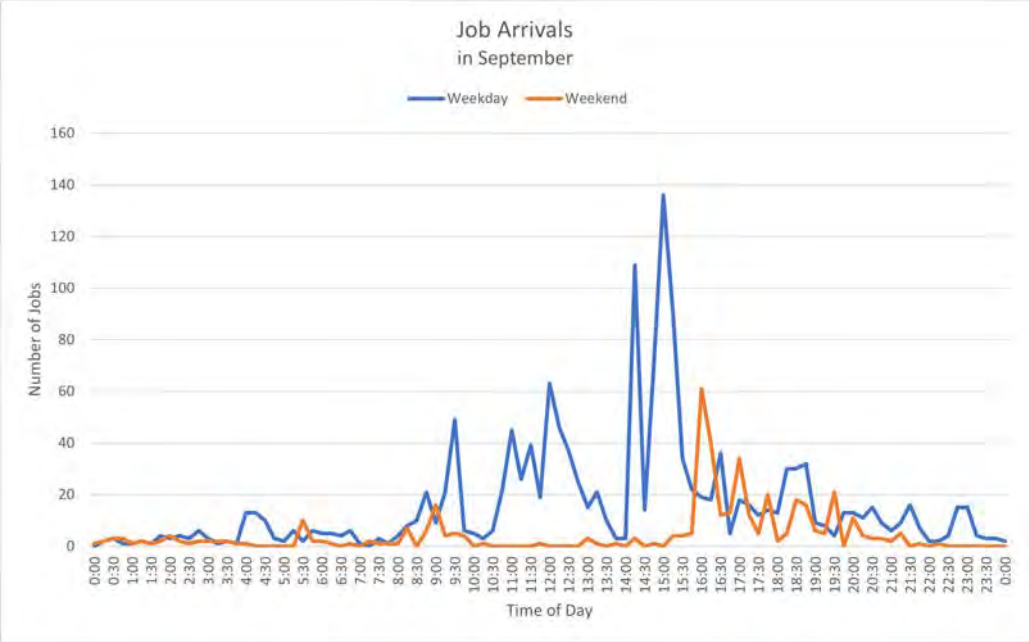


(d) State per User during all 3 months

Figure 4.1: Charts showing State per User

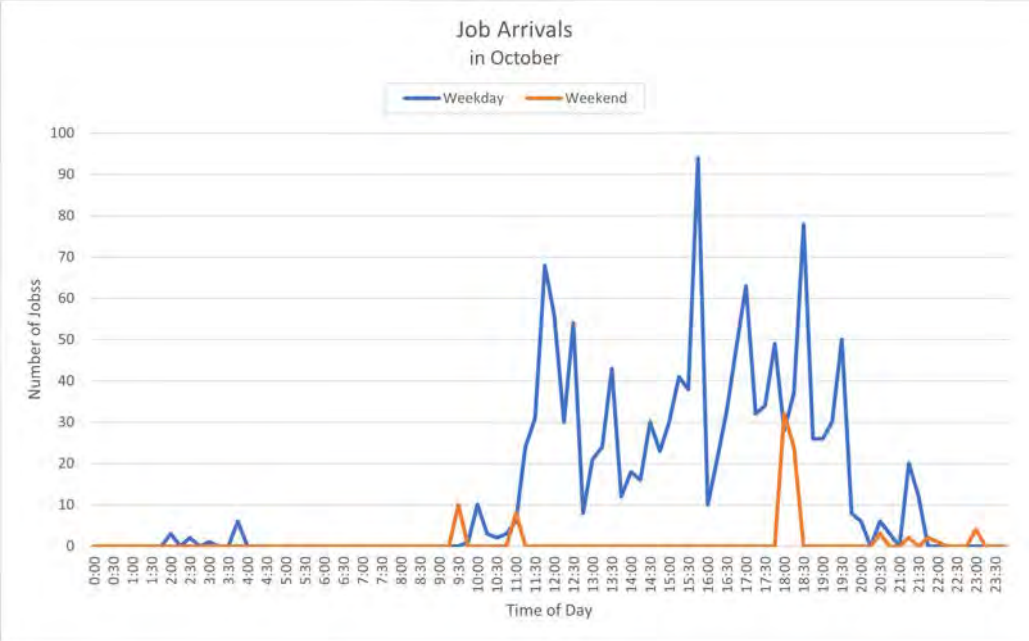
4.2 Job Arrivals

Figure 4.2 shows the daily job arrivals. In these four charts, one can see how many jobs arrived at what time of day. Once more, the vertical axis counts the number of jobs, and on the horizontal axis, the time of day is displayed. The blue line represents jobs executed Monday through Friday, whereas the orange line reflects weekend jobs.
In September, most of the jobs arrived in the afternoon. There are two outstanding lows, the first at 10 am and the second at 2 pm.
Looking at (b) and (c), on weekdays in October and November, almost all the jobs were run after 11 am and before midnight. On the weekends in November, most jobs were executed at night or in the morning.
Overall, most of the jobs arrived between 11 am and 7 pm on weekdays. On weekends, the jobs were executed the whole day. This can be seen in (d).

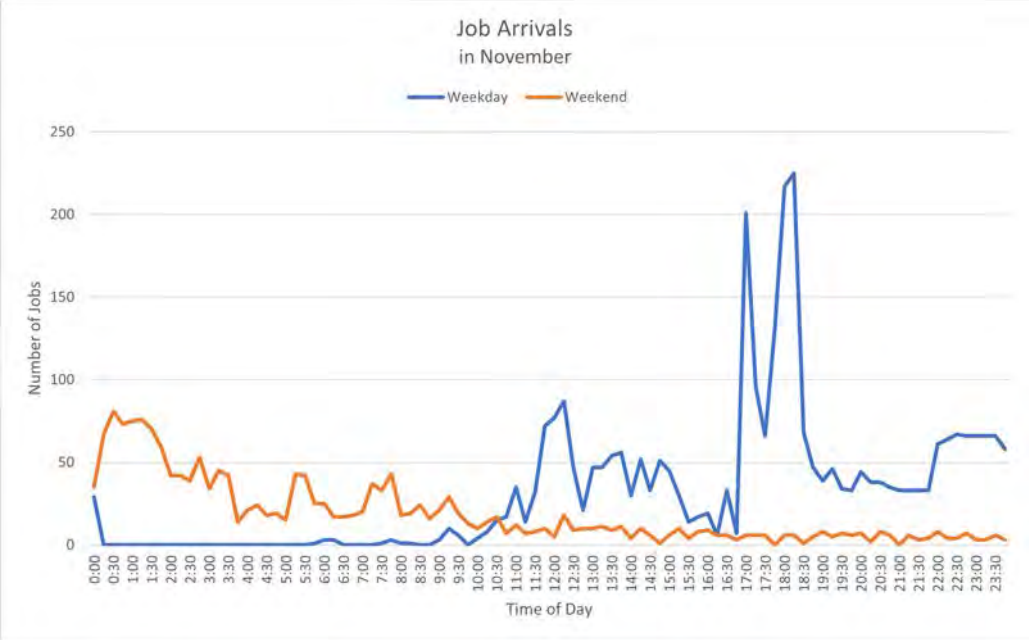


(a) Job Arrivals in September

Figure 4.2: Charts showing Job Arrivals

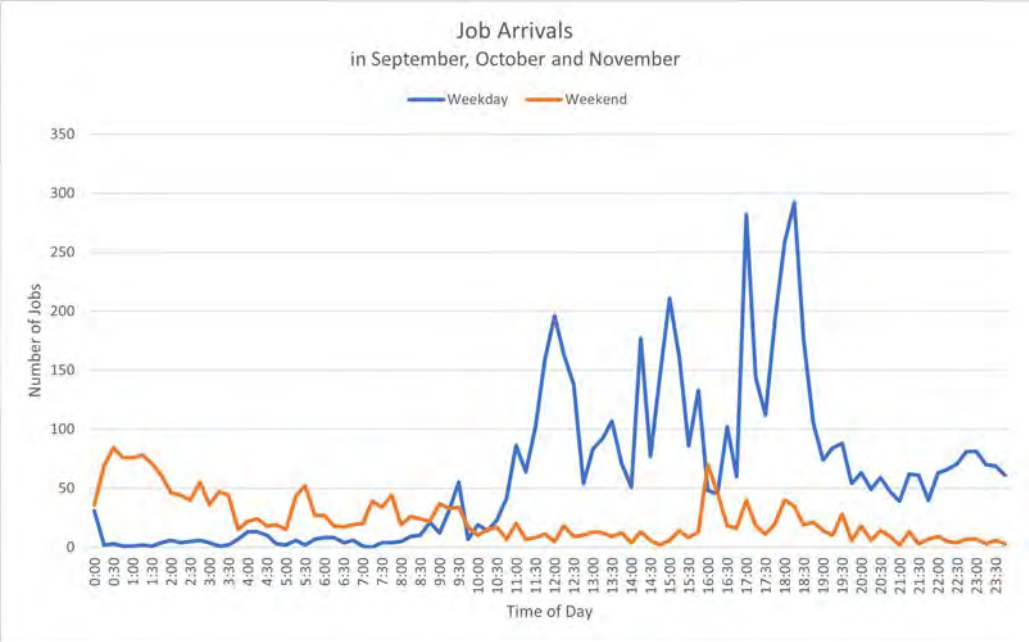


(b) Job Arrivals in October



(c) Job Arrivals in November

Figure 4.2: Charts showing Job Arrivals



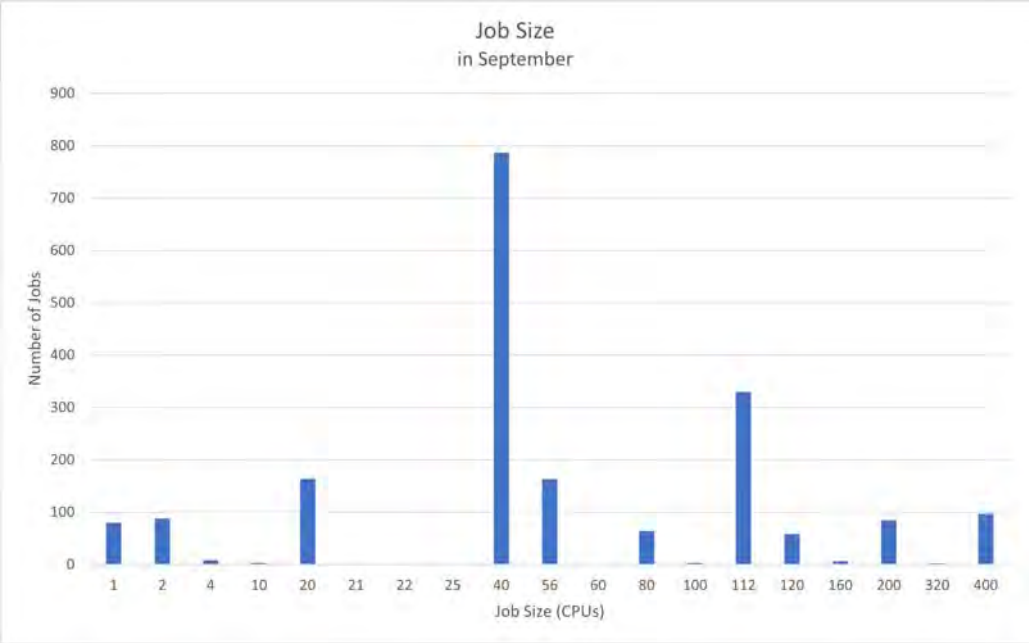
(d) Job Arrivals during all 3 months

Figure 4.2: Charts showing Job Arrivals

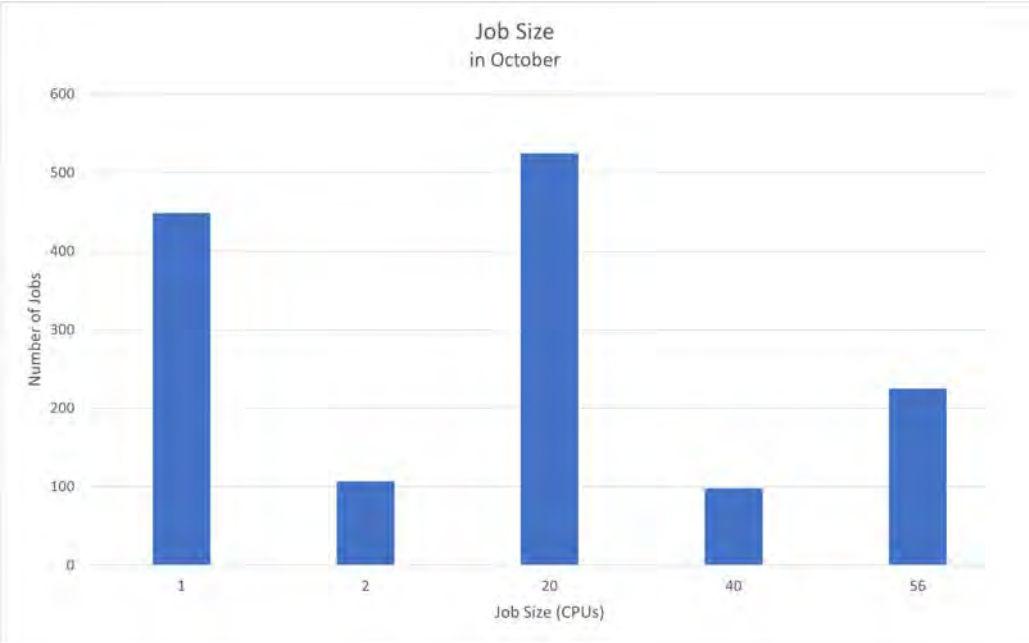
4.3 Job Size

4.3.1 Job Size

Figure 4.3 shows the sizes of the executed jobs. Again, the vertical axis represents the number of jobs, while the horizontal axis displays the different job sizes. Wherever the number of jobs appears to be zero, the number was just too small to be displayed in these charts.
As seen in (a), the most common job sizes in September were 40 and 112, implying that the majority of jobs utilized either 40 or 112 CPUs.
(b) shows us that in October, only five different sizes were used, namely 1, 2, 20, 40 and 56. Here the main one is the size 20, followed by 1.
In November, the disparities become much more pronounced. Almost all jobs were either size 20 or size 56. This can be seen in (c).
Overall, the sizes 20 and 56 were used the most frequently, which agrees with the date from November.

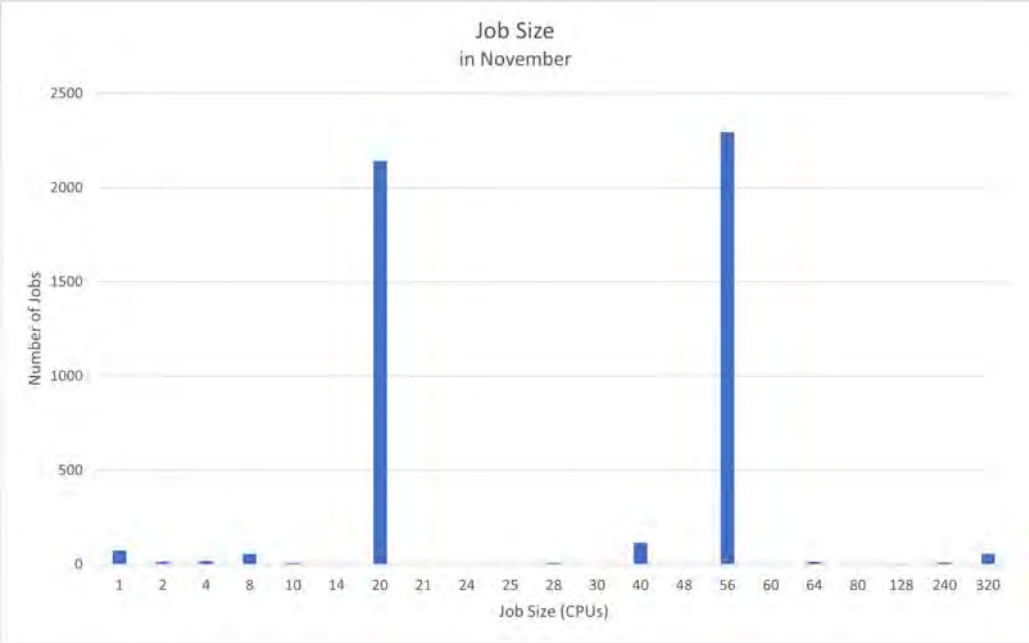


(a) Job Size in September

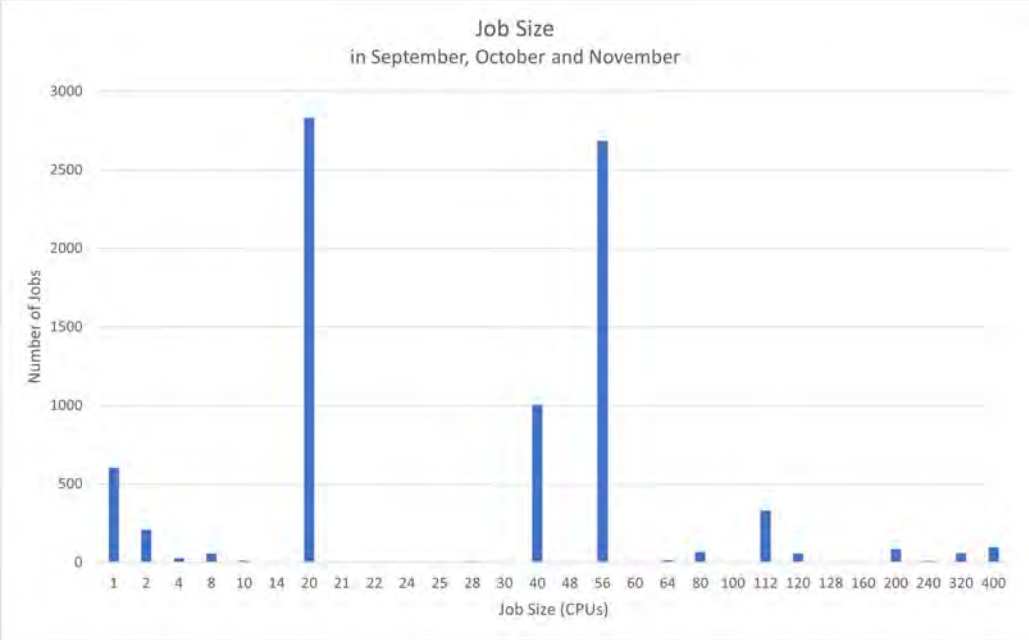


(b) Job Size in October

Figure 4.3: Charts showing Job Size



(c) Job Size in November

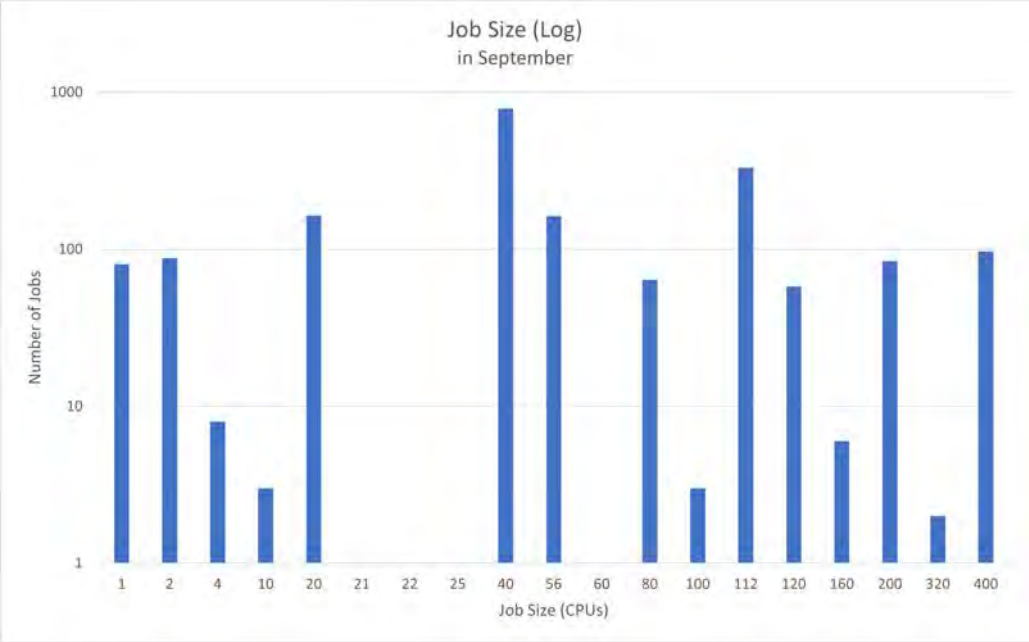


(d) Job Size during all 3 months

Figure 4.3: Charts showing Job Size

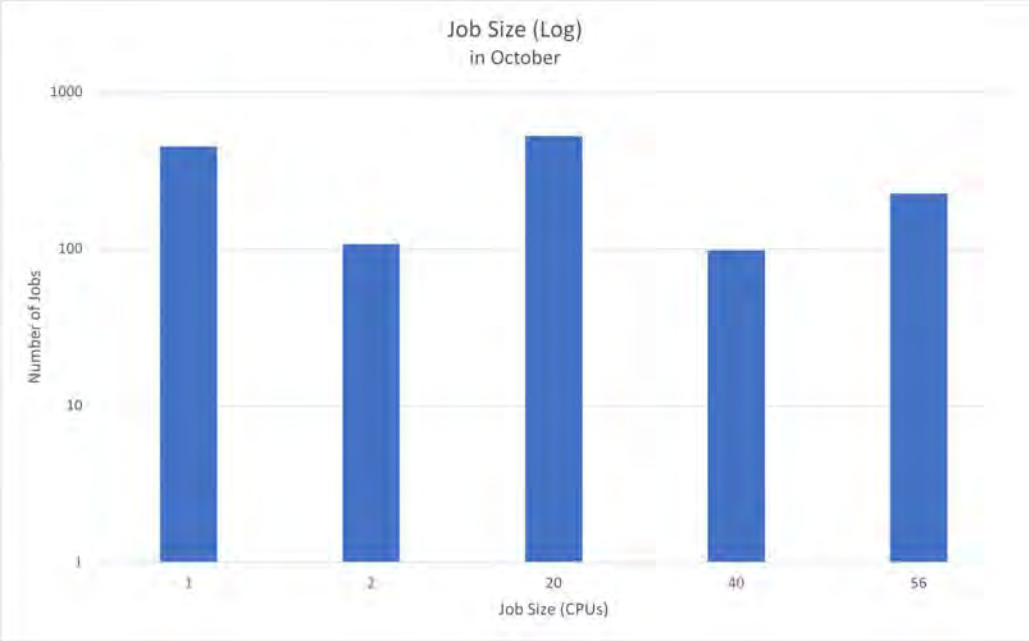
4.3.2 Job Size (Log)

The charts in Figure 4.4 display the same data as those in Figure 4.3. The difference is the vertical axis. To not have as many values appear to be zero, a logarithmic scale was used for these plots. Of course, the insights gained from these charts stay the same.
(d) shows that a lot of different job sizes were used during the three months. While the job sizes 22, 24, 30 and 48 were only utilized once in November, sizes 20 and 56 were the clear favorites, followed by size 40.

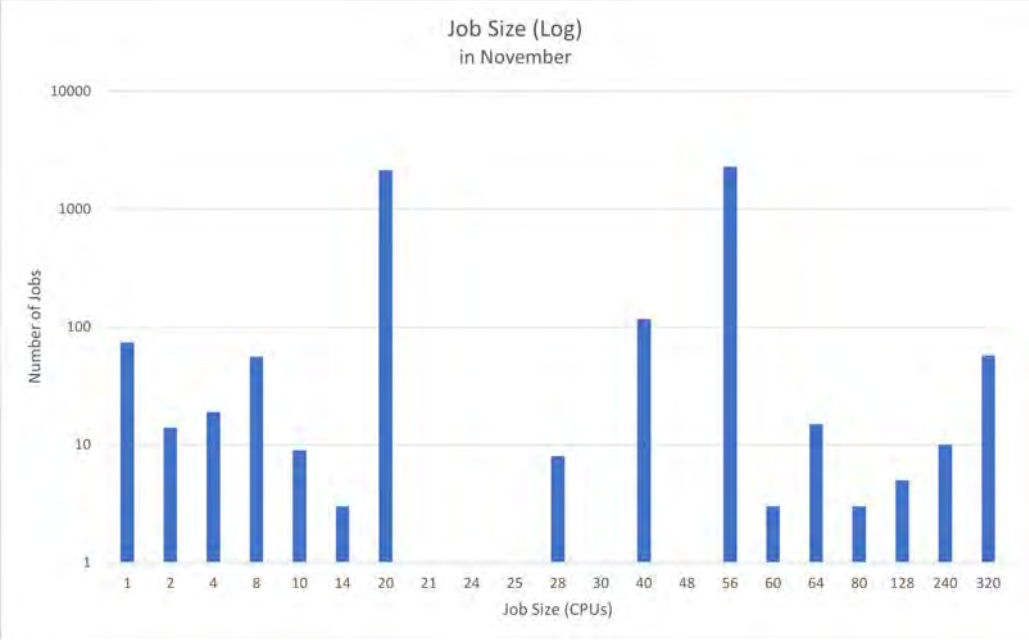


(a) Job Size (Log) in September

Figure 4.4: Charts showing Job Size (Log)

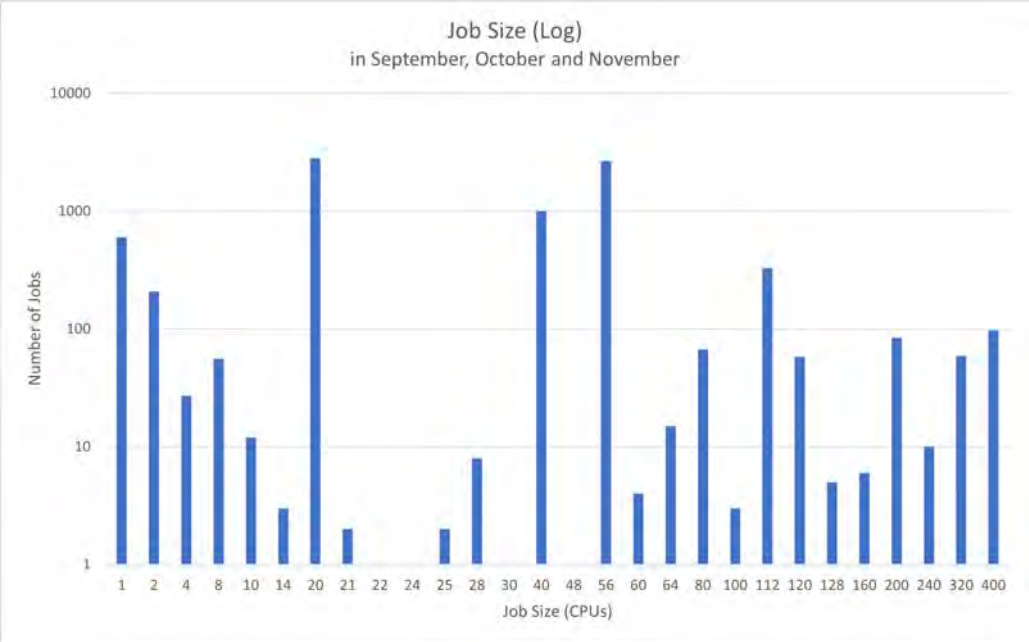


(b) Job Size (Log) in October



(c) Job Size (Log) in November

Figure 4.4: Charts showing Job Size (Log)

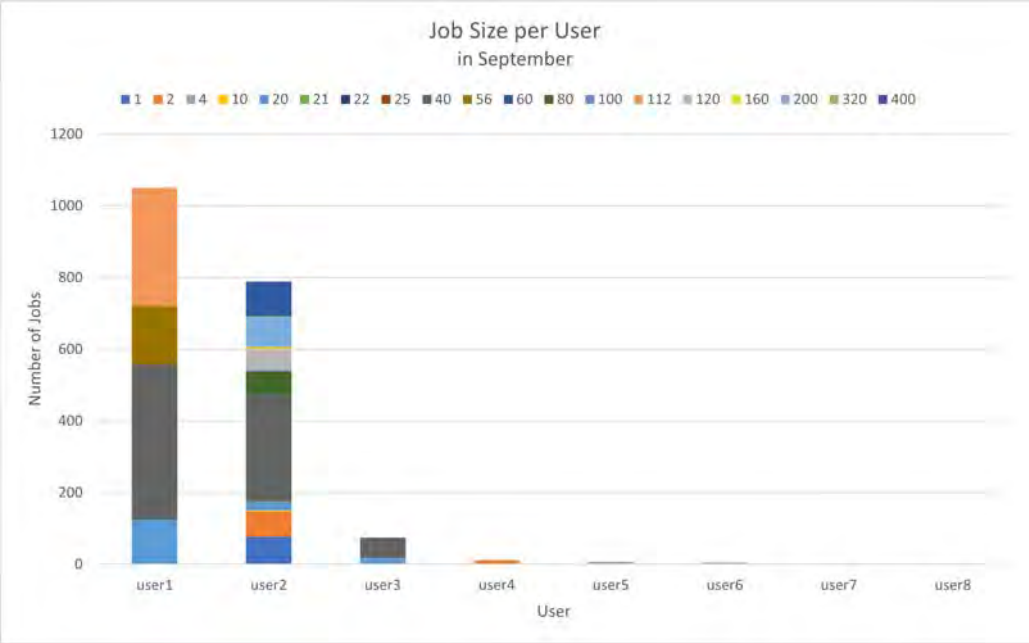


(d) Job Size (Log) during all 3 months

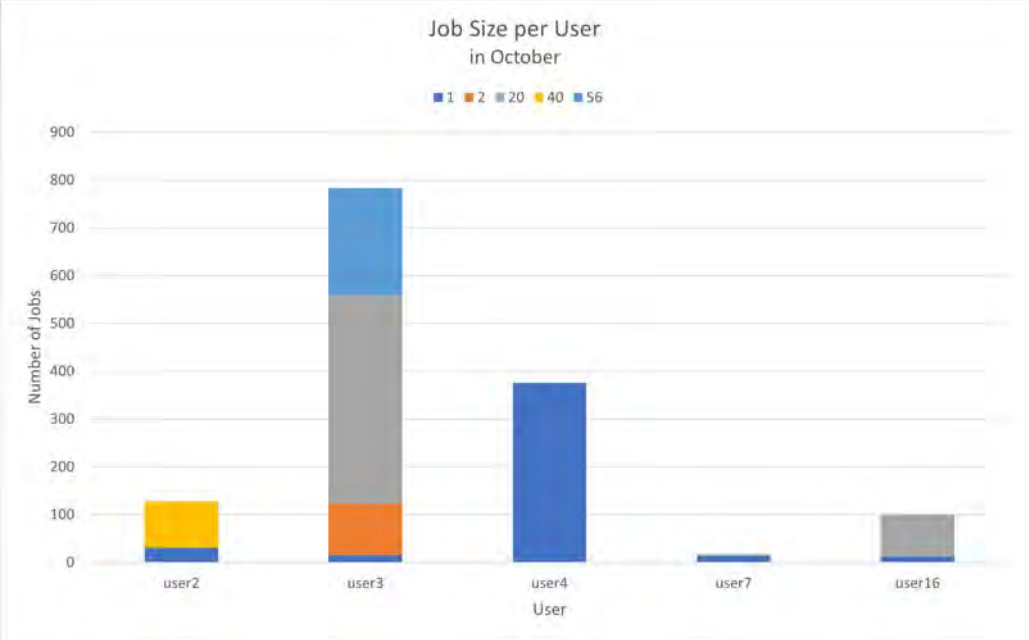
Figure 4.4: Charts showing Job Size (Log)

4.3.3 Job Size per User

To not only know the job sizes overall but also which user executed how many jobs of which size, the charts shown in Figure 4.5 were created.
As (a) shows, no user used only one job size in September. Especially user2 utilized many different job sizes. The same applies to user1, although not to the same extent.
It looks different in October. There, user4 only executed jobs of size 1. This can be seen in (b).
In November, we again did not have a user that used only one size.
Looking at the overview over the three months, we see that while user4 only used one size in October, he used different sizes during the whole three months. This means that no user made use of the same job size during the entire three months.

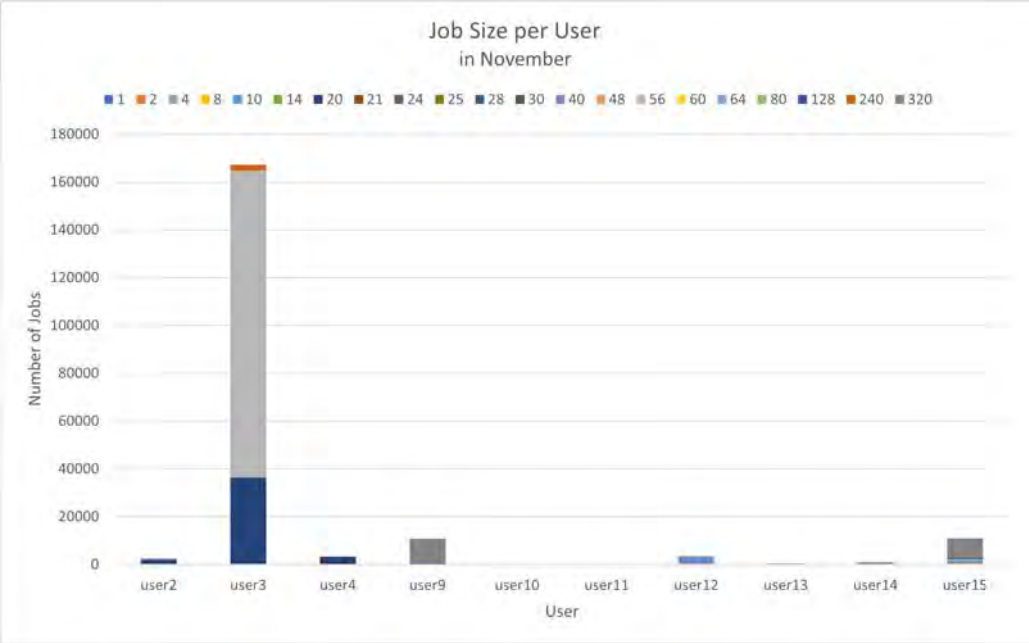


(a) Job Size per User in September

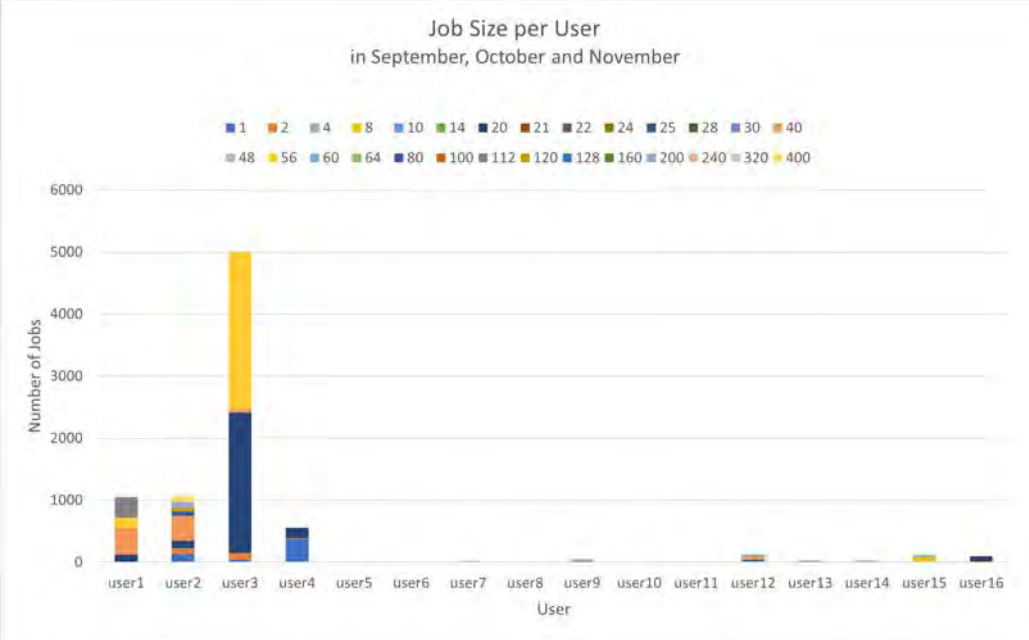


(b) Job Size per User in October

Figure 4.5: Charts showing Job Size per User



(c) Job Size per User in November

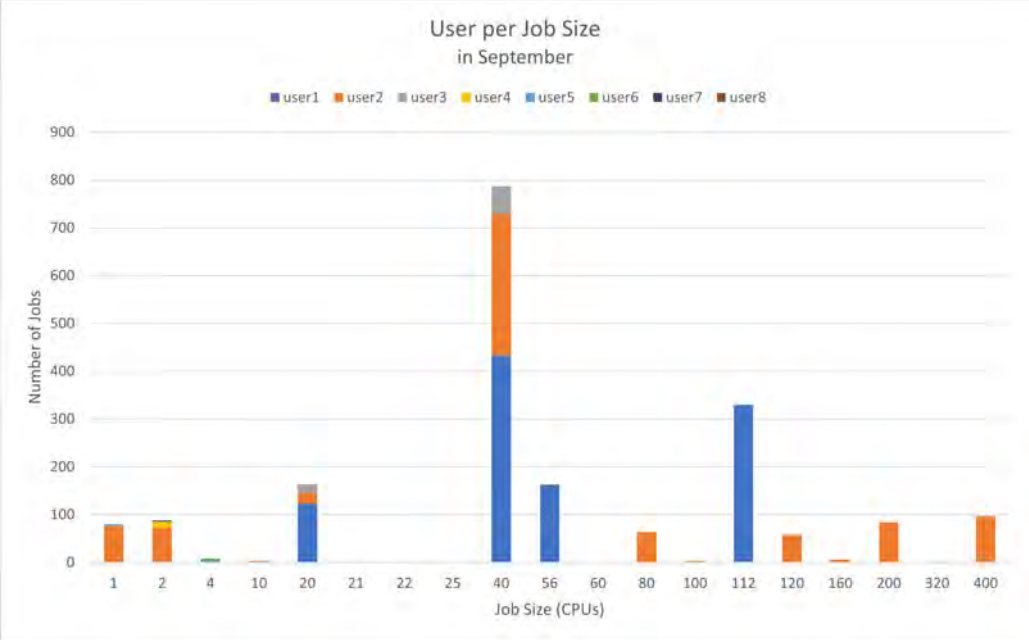


(d) Job Size per User during all 3 months

Figure 4.5: Charts showing Job Size per User

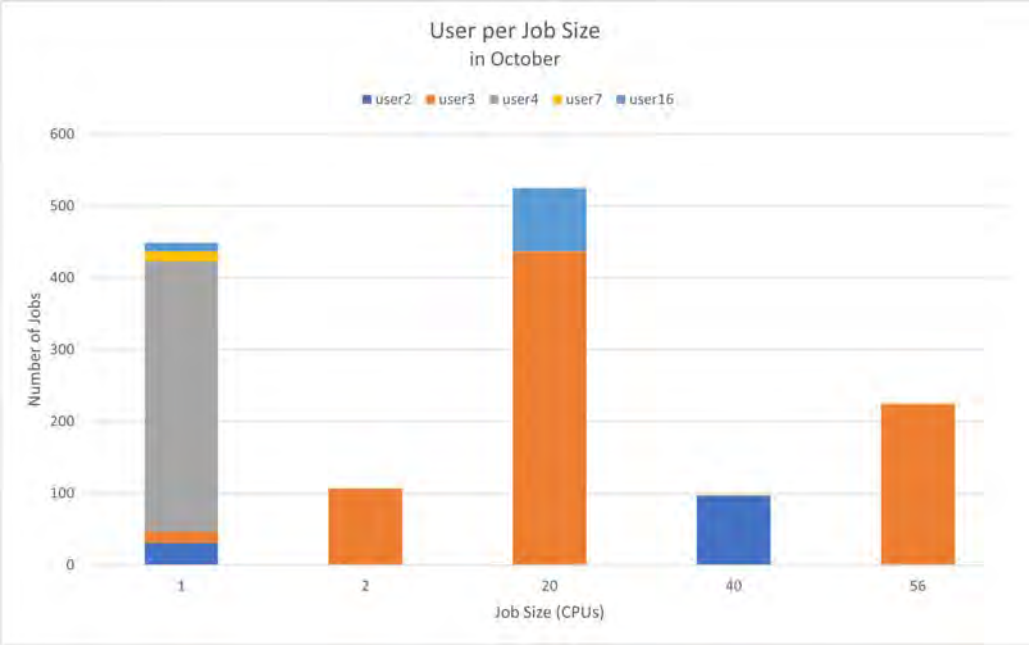
4.3.4 User per Job Size

Figure 4.6 shows the counterpart of Figure 4.5. Here, the User per Job Size is displayed.
(a) shows that in September, there were some job sizes that were only used by one user. For example, size 112 was only utilized by user1, while the sizes 60, 80, 120, 160, 200, 320 and 400 were only used by user2.
As shown in (b), there was one job size that was only used by one user in October, and that was job size 2, with the only user being user3.
In November, there were numerous sizes only used by one user. One of them was size 80, which was only used by user12. Unfortunately, since some sizes were used hundreds of times, the chart is not able to display the really small bars.

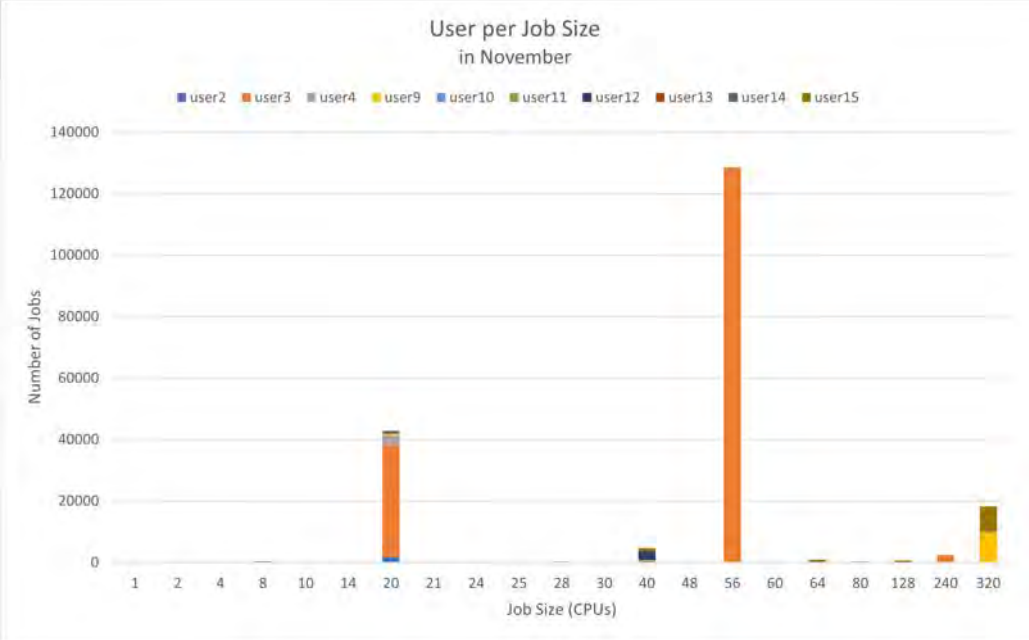


(a) User per Job Size in September

Figure 4.6: Charts showing User per Job Size

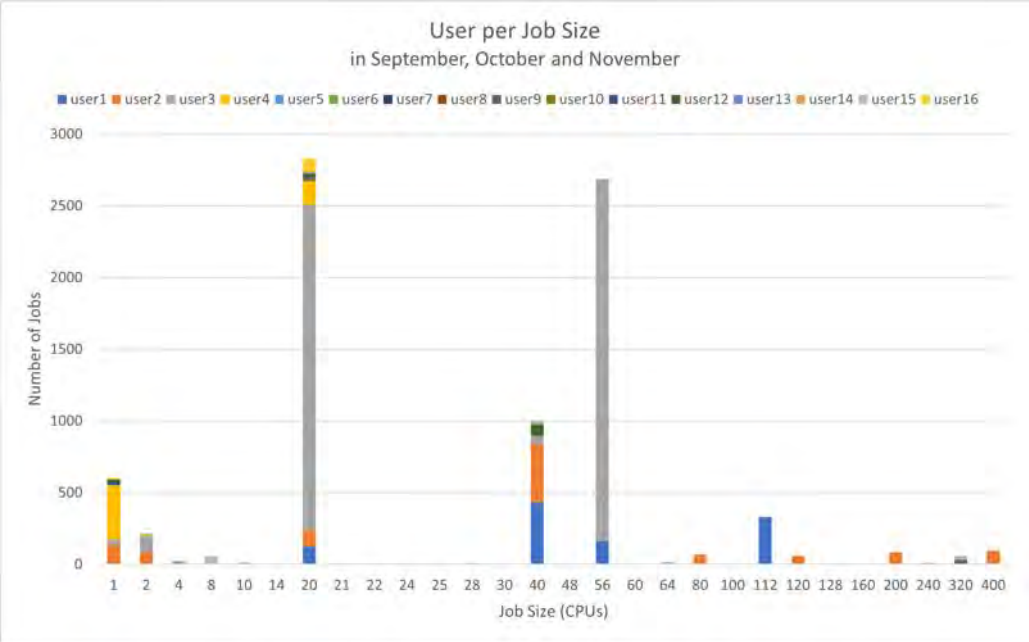


(b) User per Job Size in October



(c) User per Job Size in November

Figure 4.6: Charts showing User per Job Size



(d) User per Job Size during all 3 months

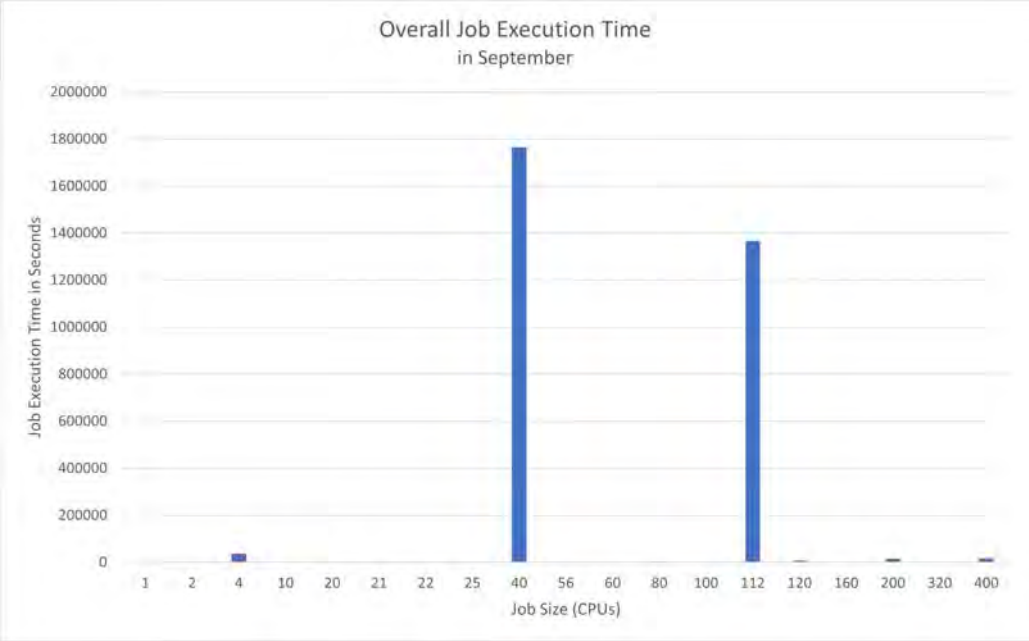
Figure 4.6: Charts showing User per Job Size

4.4 Job Execution Time

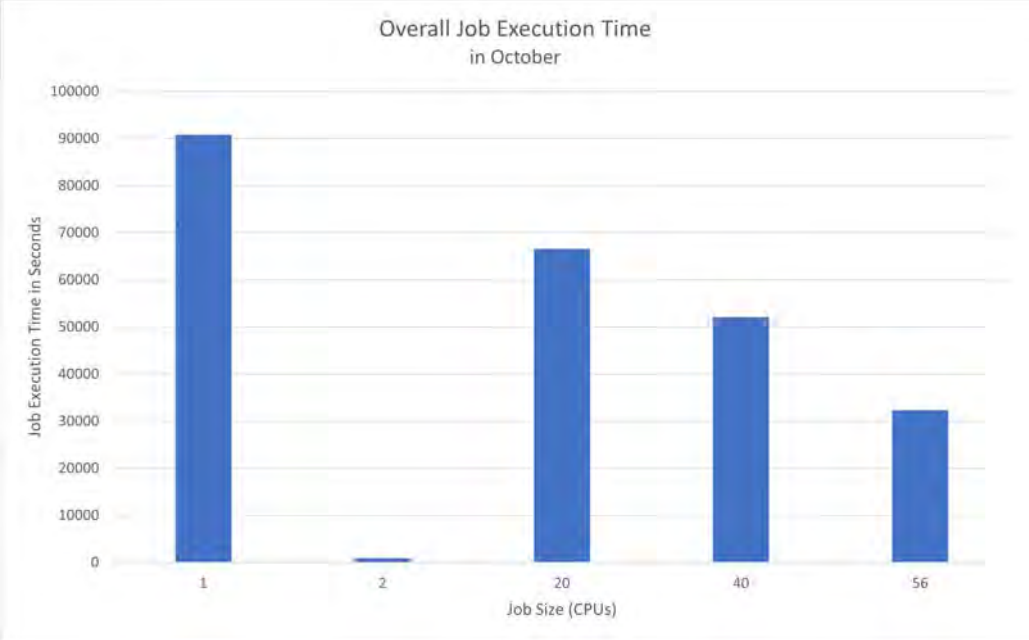
The job execution time is the time the jobs are running. In the HPC job data fields, this parameter is called Elapsed.

4.4.1 Overall Job Execution Time

In 4.4.1, the overall job execution time is displayed. This means that all the execution times of a specific job size get added together. As seen in Figure 4.7 (a), the jobs of the sizes 40 and 112 ran for the longest time by a lot in September. Size 40 jobs were running for almost 1'800'000 seconds while numerous other values appear to be zero. This, once again, is due to the chart's inability to display such small numbers when huge ones exist. October looks a lot different, with job size 1 running the longest. Here, the longest execution time is only about 90'000 seconds, with the lowest being barely visible at 951 seconds. This can be seen in (b). The chart displaying the November data (c) looks similar to the one from September, with the job sizes 20 and 56 being the clear peaks. The difference is that the longest execution time, belonging to the size 20, is about 280'000 seconds, which is quite the difference from the almost 2 million seconds in September. What we can see in (d) is the fact that the values from September were that high, so the chart containing the data from the three months looks similar to the one from September. The only differences are the small bars at sizes 20 and 56, which can be traced back to the data from November.

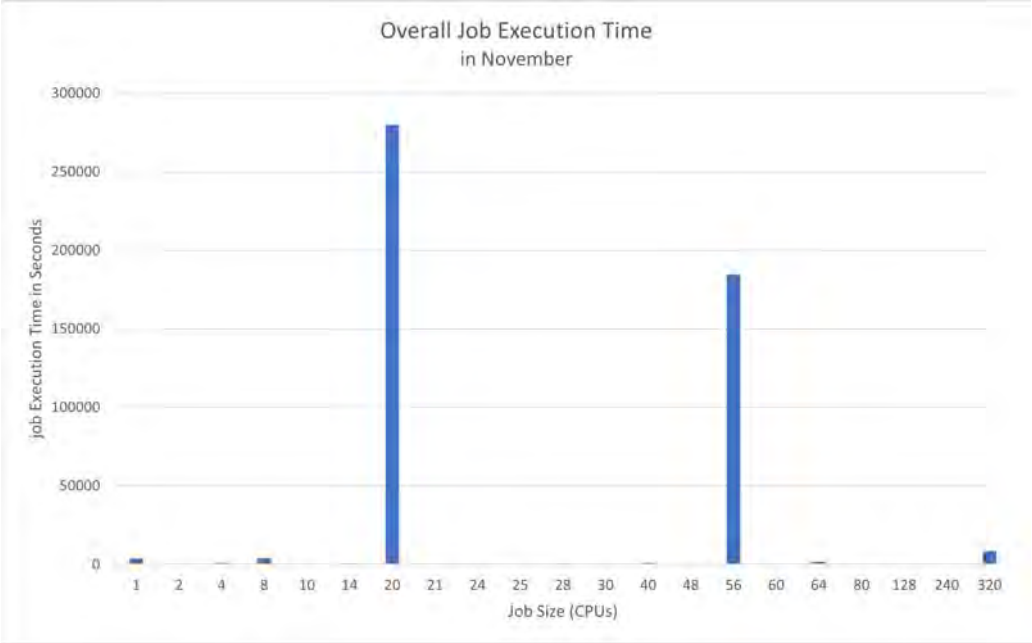


(a) Overall Job Execution Time in September

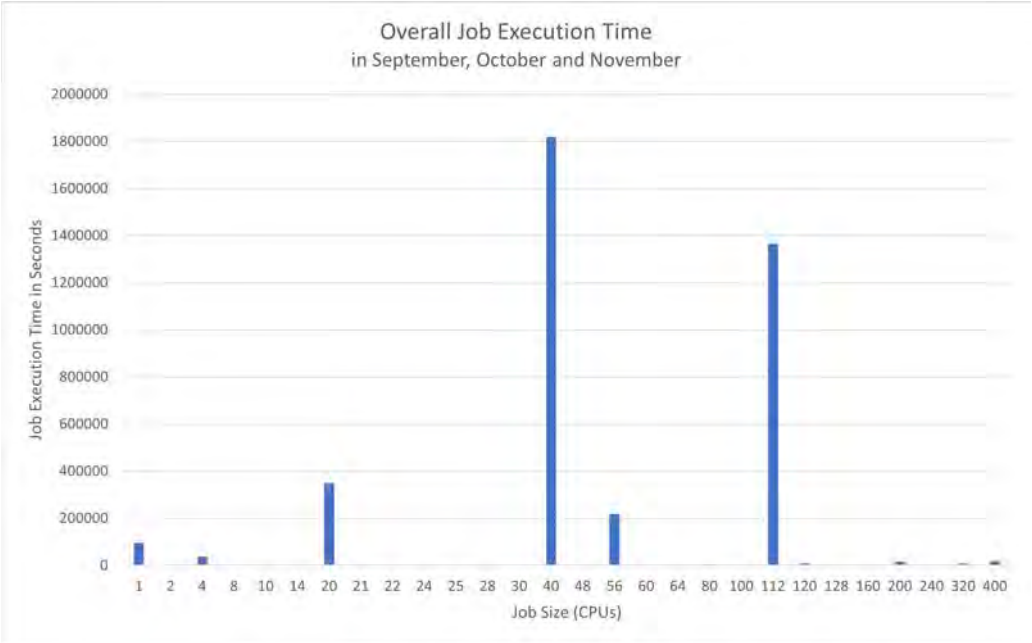


(b) Overall Job Execution Time in October

Figure 4.7: Charts showing Overall Job Execution Time



(c) Overall Job Execution Time in November

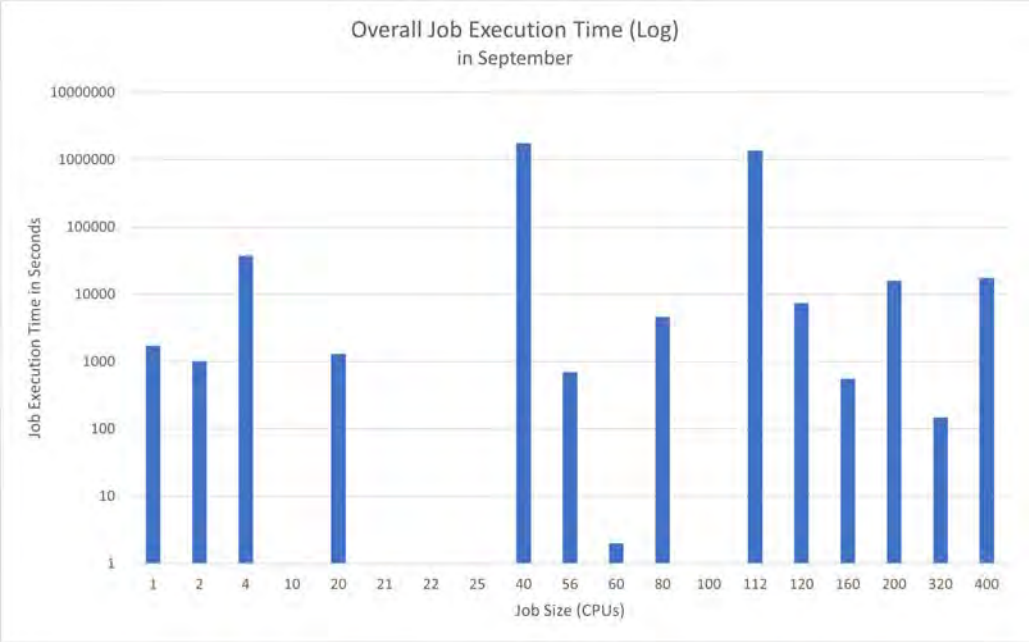


(d) Overall Job Execution Time during all 3 months

Figure 4.7: Charts showing Overall Job Execution Time

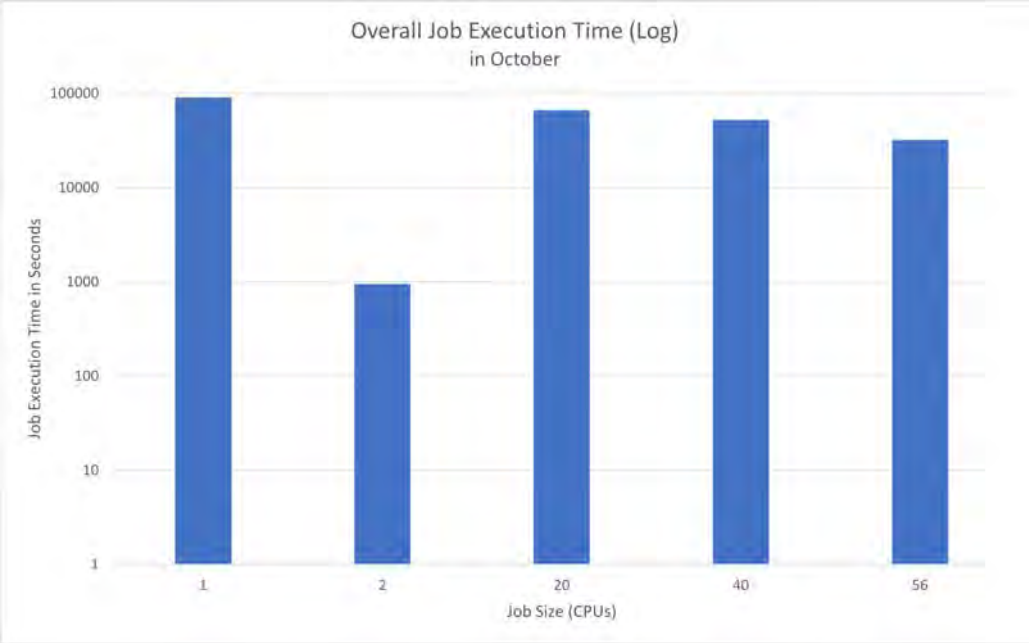
4.4.2 Overall Job Execution Time (Log)

In this subsection, the charts display the same data as the ones in Figure 4.7. The difference, once again, is the use of a logarithmic vertical axis. Using this, we want to avoid the values to appear to be zero. As can be seen in (a), this does not work for all the values. The values for the sizes 10, 21, 22, 25 and 100 all lay between zero and two seconds.

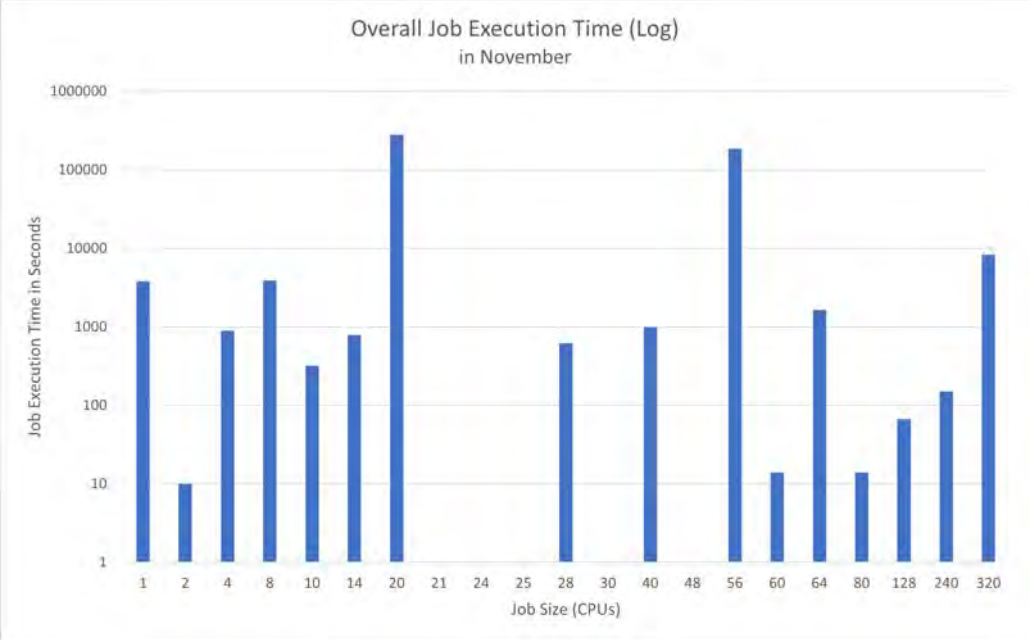


(a) Overall Job Execution Time (Log) in September

Figure 4.8: Charts showing Overall Job Execution Time (Log)



(b) Overall Job Execution Time (Log) in October



(c) Overall Job Execution Time (Log) in November

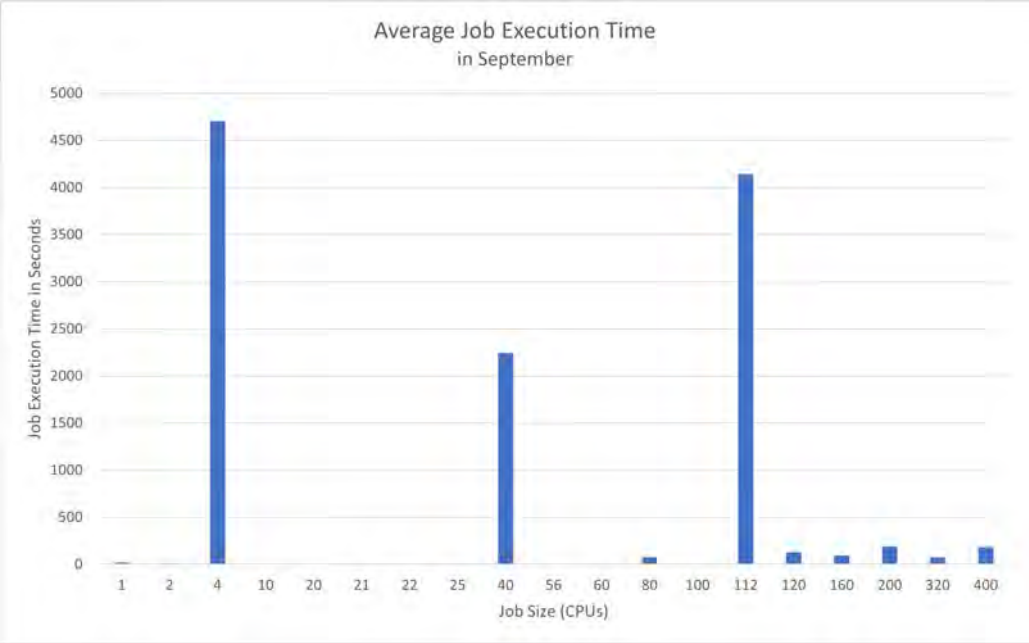
Figure 4.8: Charts showing Overall Job Execution Time (Log)

(d) Overall Job Execution Time (Log) during all 3 months

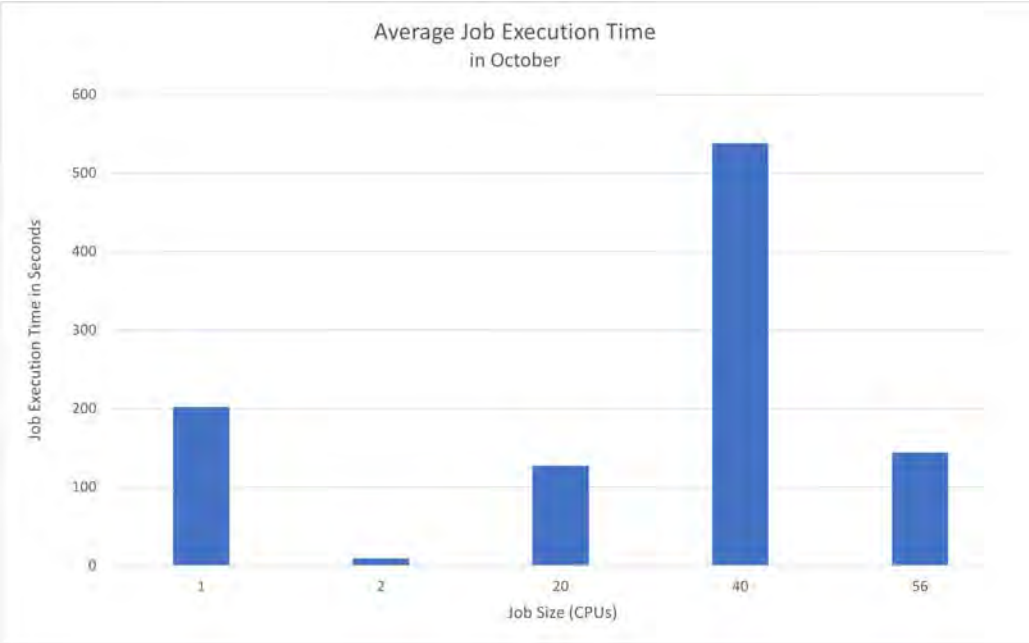
Figure 4.8: Charts showing Overall Job Execution Time (Log)

4.4.3 Average Job Execution Time

In this subsection, we do not look at the overall job execution time but rather the average job execution time. This means that we want to know how long a job of a specific size runs on average.
In September, as seen in (a), the average job execution time for jobs of size 4 was about 4'700 seconds, and for jobs of size 112 about 4'150 seconds.
In October, we have another front-runner, namely job size 40, with an average job execution time of 538 seconds. Notably, the other jobs all ran for less than 210 seconds on average.
What is even more interesting is that the highest average job execution time in November was only 262 seconds. This was achieved by jobs of size 14, which can be seen in (c).
Looking at the three months combined, there is a clear leader. Since jobs of size 112 were only executed in September, the average job execution time over the three months is still around 4'150 seconds. Job sizes 40 and 4 have the next highest average job execution time.
All of this is displayed in Figure 4.9.

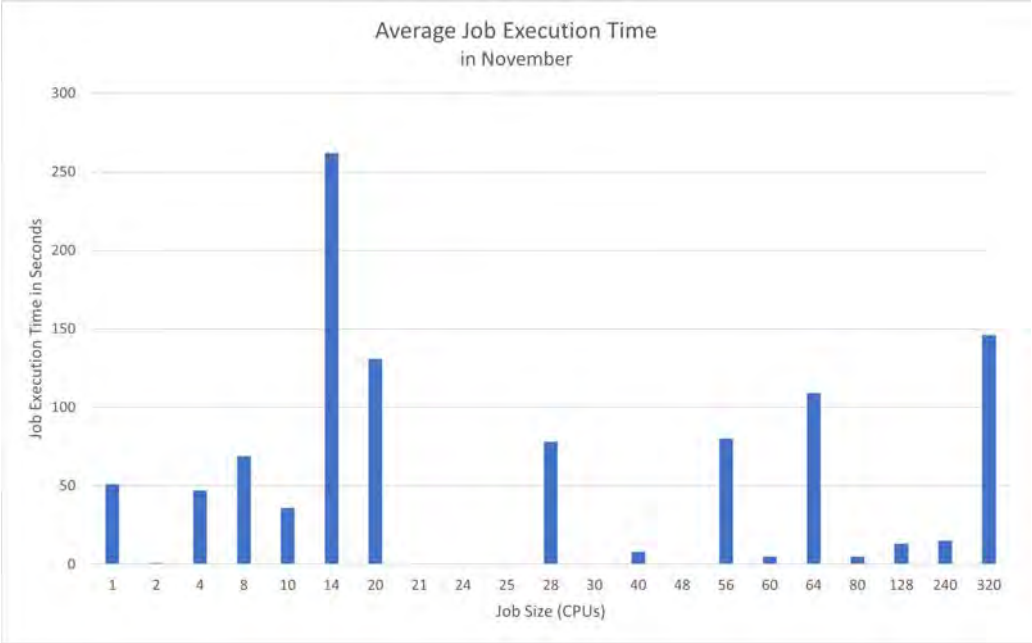


(a) Average Job Execution Time in September

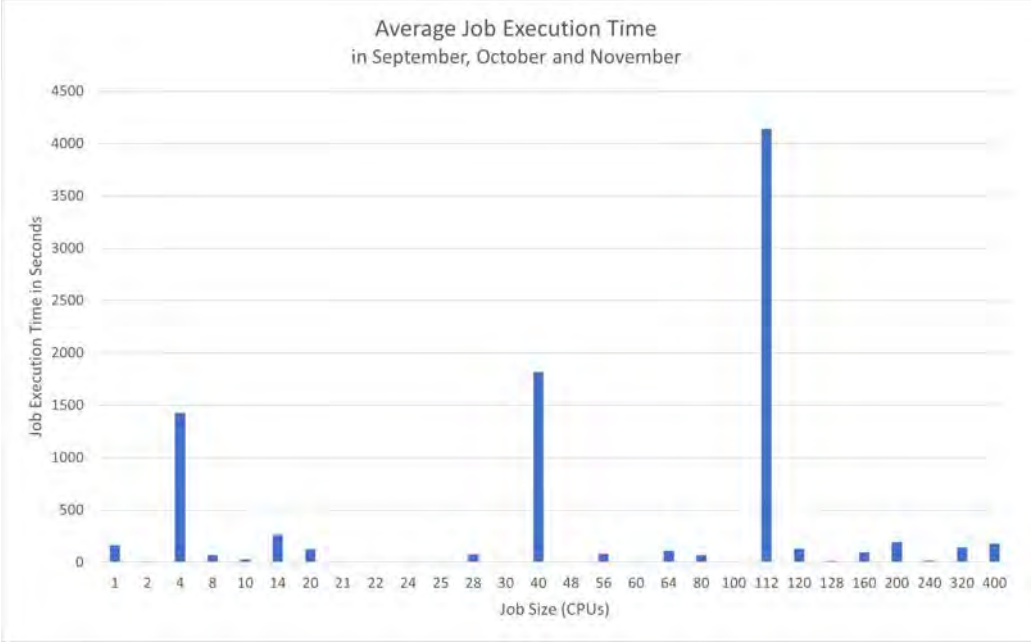


(b) Average Job Execution Time in October

Figure 4.9: Charts showing Average Job Execution Time



(c) Average Job Execution Time in November



(d) Average Job Execution Time during all 3 months

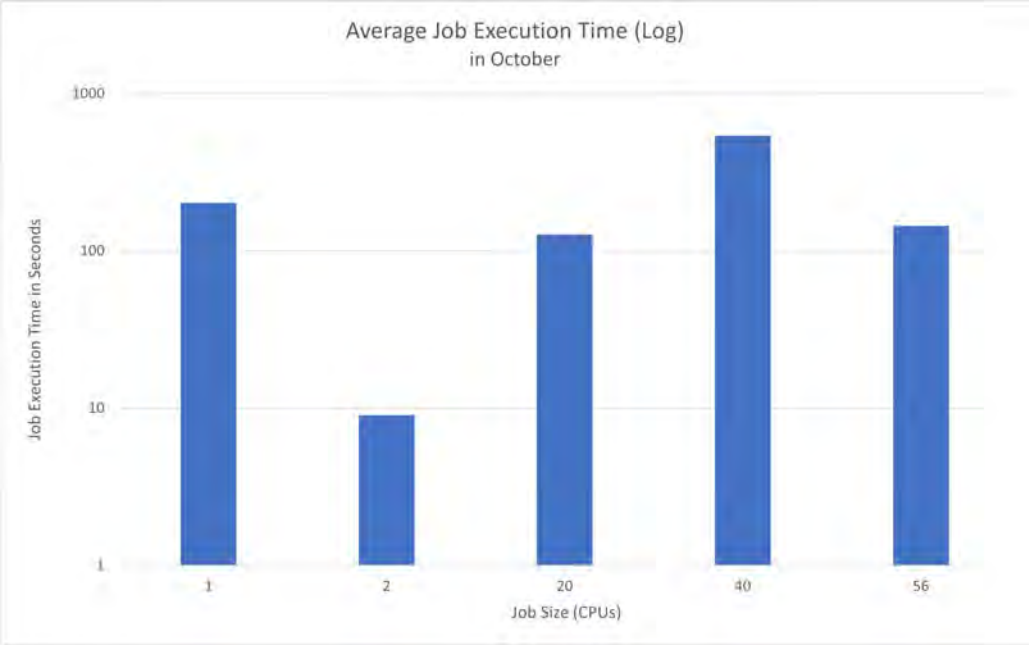
Figure 4.9: Charts showing Average Job Execution Time

4.4.4 Average Job Execution Time (Log)

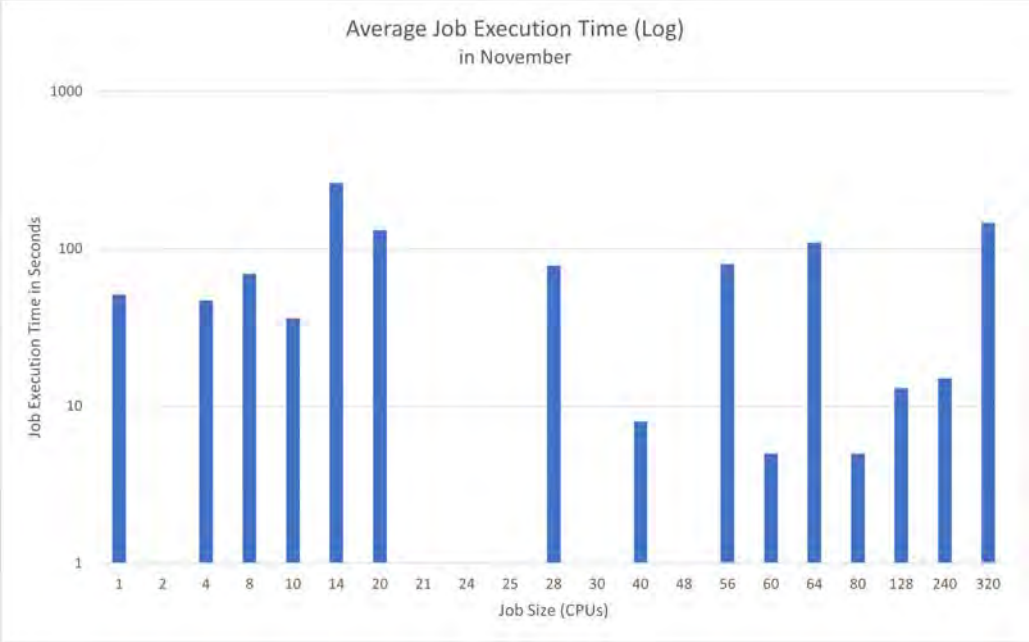
Once again, we are making use of the logarithmic scale on the vertical axis to be able to display small values. The data is the one that was also used in Figure 4.9.

(a) Average Job Execution Time (Log) in September

Figure 4.10: Charts showing Average Job Execution Time (Log)

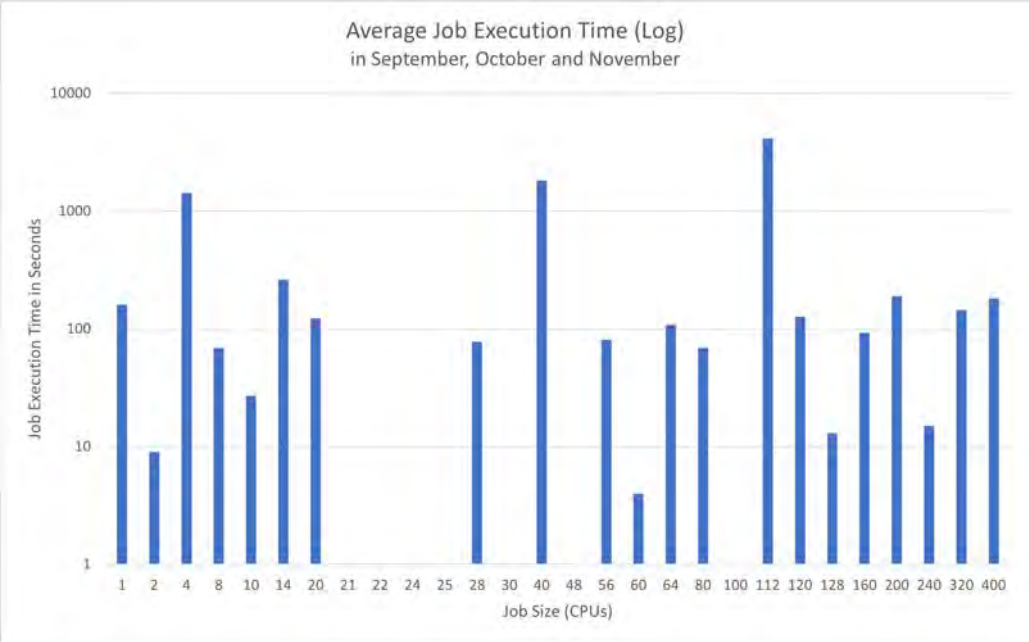


(b) Average Job Execution Time (Log) in October



(c) Average Job Execution Time (Log) in November

Figure 4.10: Charts showing Average Job Execution Time (Log)

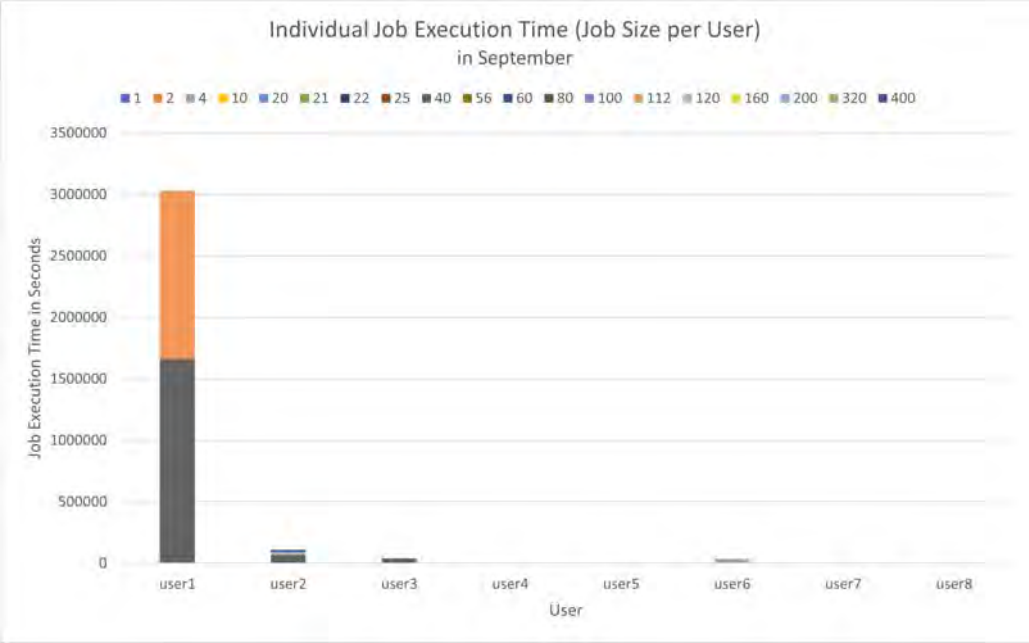


(d) Average Job Execution Time (Log) during all 3 months

Figure 4.10: Charts showing Average Job Execution Time (Log)

4.4.5 Individual Job Execution Time

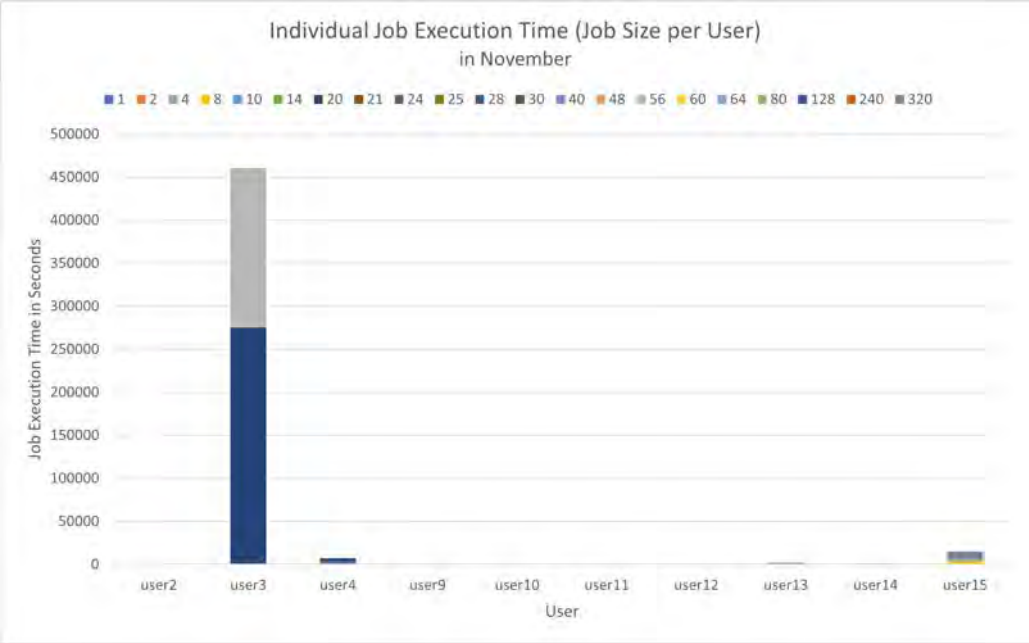
Here, we examine the individual job execution times. How many seconds did a user spend executing jobs of a particular size? This question was answered using the overall job execution time, not the average.
Figure 4.11 (a) clearly demonstrates that user1 ran the longest jobs. The two most notable job sizes are size 40 and 112, respectively. The other users' overall job execution times were significantly lower.
As shown in (b), user3 and user7 executed the most time-intensive jobs in October. While user3 made use of four different job sizes, user7 utilized only three different sizes. It may appear that user3 used three sizes and user7 only two, but this is due to the extremely low job execution time of jobs of size 1 for user3 and jobs of size 20 for user7.
In November, the highest individual job execution time was by far achieved by user3. They used five different sizes, with three of them having such small job execution times that they were imperceptible.
Overall, user1 ran the most lengthy jobs while also using only a small number of different job sizes. Following user1 is user3, with a job execution time a lot lower than the one of user1. But they only use a small number of different job sizes as well.



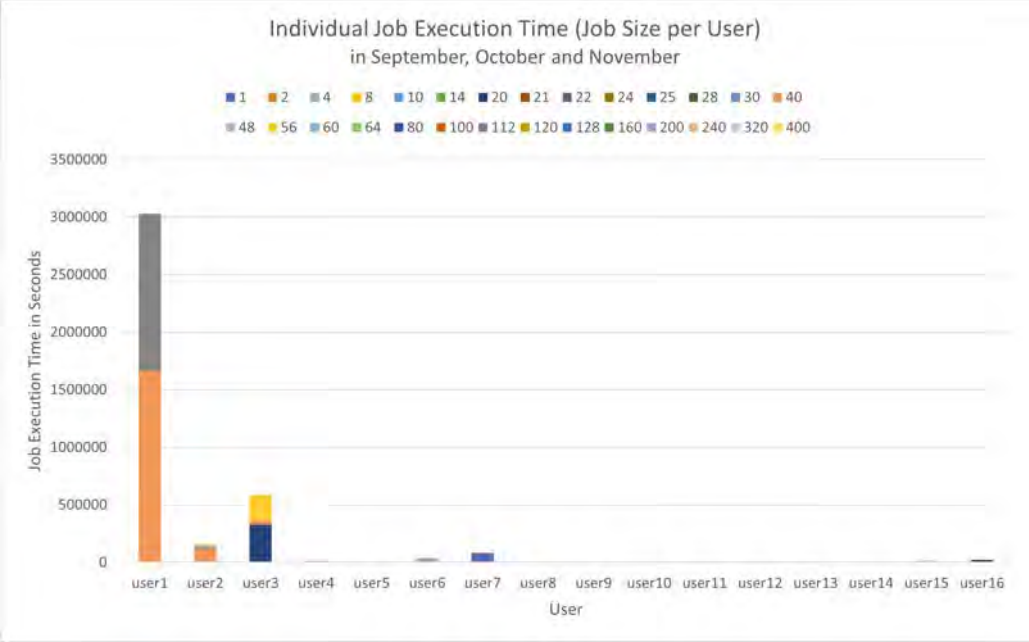
(a) Individual Job Execution Time in September

(b) Individual Job Execution Time in October

Figure 4.11: Charts showing Individual Job Execution Time



(c) Individual Job Execution Time in November



(d) Individual Job Execution Time during all 3 months

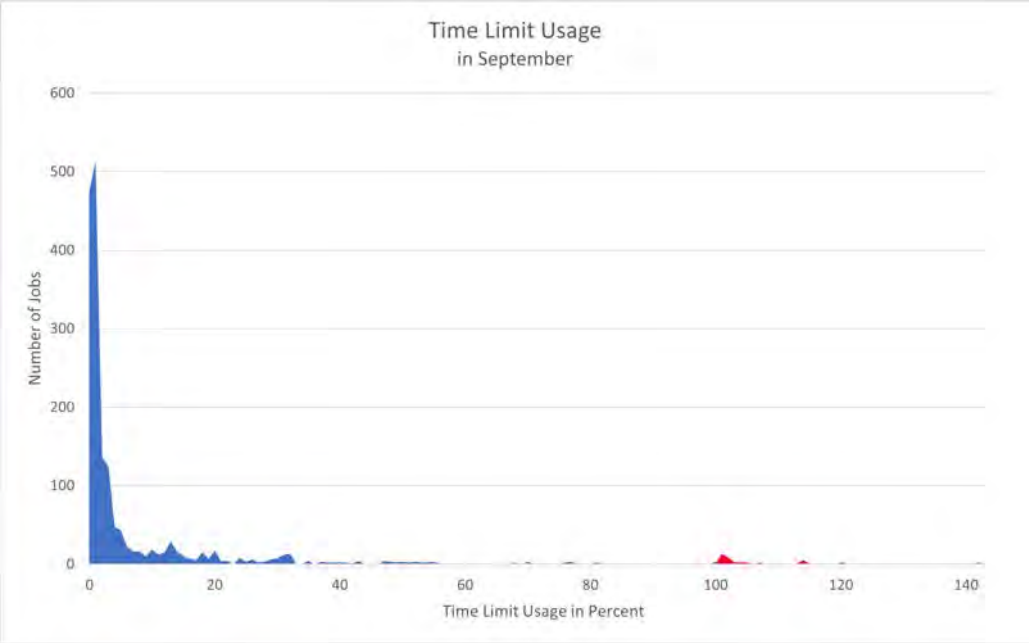
Figure 4.11: Charts showing Individual Job Execution Time

4.5 Time Limit Usage

Whenever a user runs a job, they can put a time limit. This is the maximum time a job may take to be executed and terminated. If a job exceeds the time limit, it gets timed-out, meaning it will not terminate.

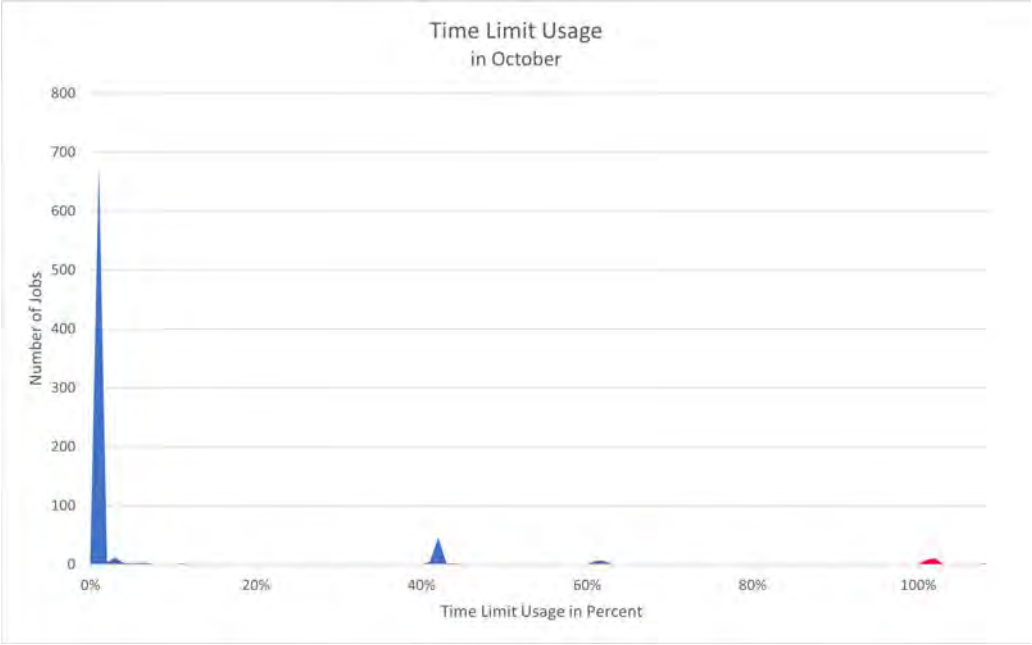
4.5.1 Time Limit Usage

In this section, we will look at the time limit usage. We want to know how much percent of the time limit the running jobs usually use. (a) shows us the time limit usage in September. The horizontal axis shows the time limit usage in percent, and the vertical axis depicts the number of jobs. Whenever a job gets timed out, more than 100 percent of the time limit is used. This is the red area in this chart. As can be seen here, while a few jobs get timed out, many jobs only use a tiny percentage of the time limit. This is what the peak at around one percent tells us.
The chart from October, chart (b), looks similar, with only a small number of jobs being timed out and a peak at one percent. Interestingly, not a single job got timed out in November, but more than 1'000 jobs only used one percent of the set time limit. The overview over the three months shows no differences from the other charts, as seen in (d).

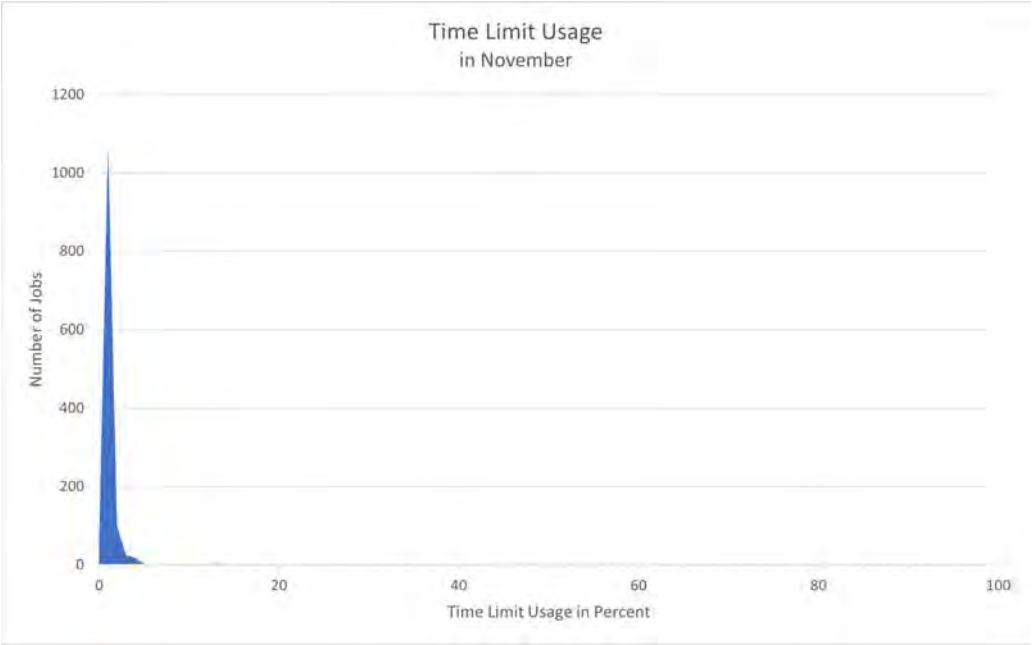


(a) Time Limit Usage in September

Figure 4.12: Charts showing Time Limit Usage

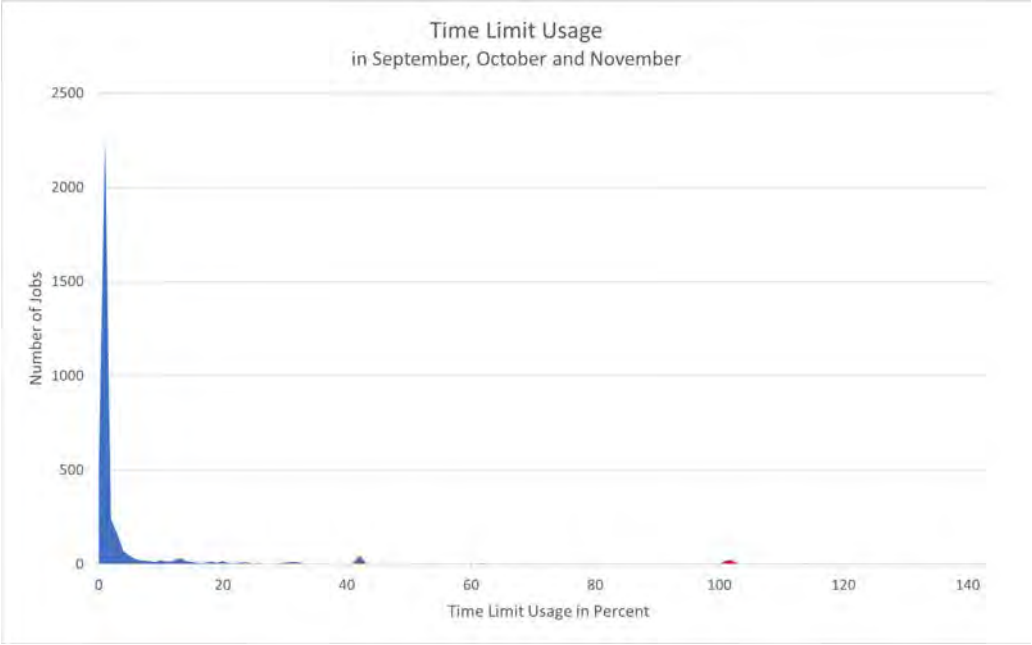


(b) Time Limit Usage in October



(c) Time Limit Usage in November

Figure 4.12: Charts showing Time Limit Usage

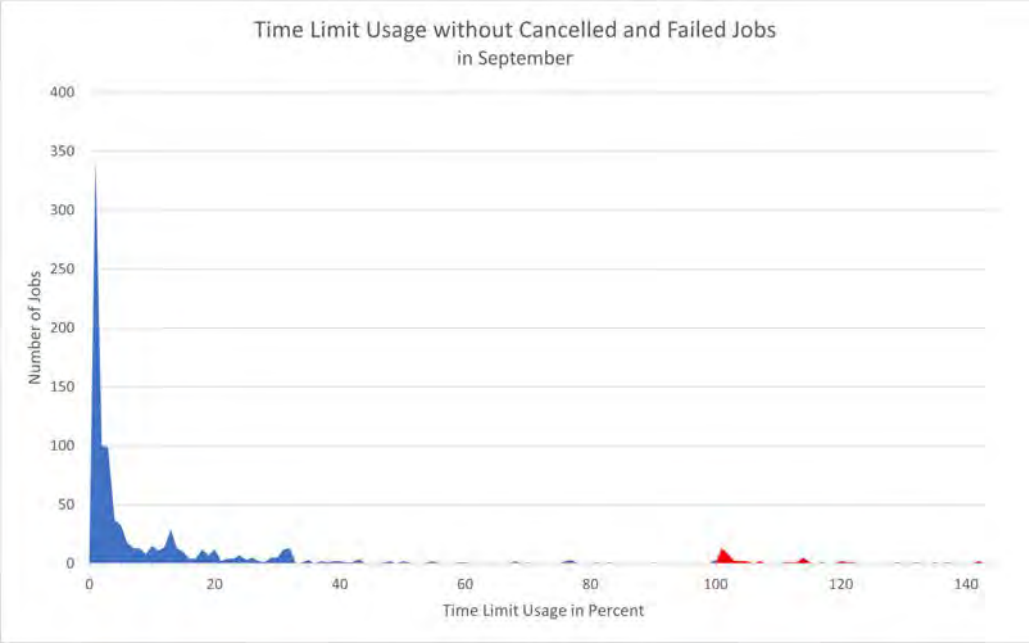


(d) Time Limit Usage during all 3 months

Figure 4.12: Charts showing Time Limit Usage

4.5.2 Time Limit Usage without Cancelled or Failed Jobs

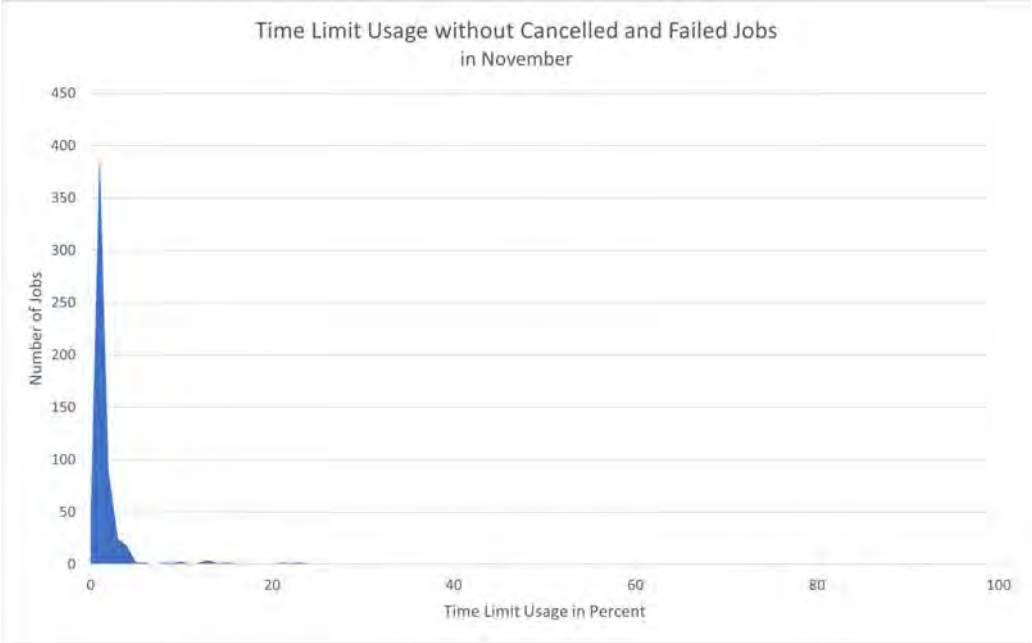
In this section, we are looking at the time limit usage again. The difference is that we only use the data of completed and timed-out jobs and not the data from cancelled or failed jobs. We decided to do this because if a job fails or gets cancelled, it certainly does not reach the time limit. Furthermore, we do not know how much of the time limit would have been used if the job was completed (or timed out).
In September and October, the only differences to the corresponding charts of Figure 4.12 are the low percentages. No completed job used zero percent of the time limit, and a smaller number used one percent. This can be seen in (a) and (b), respectively.
The peak at one percent is much lower in Figure 4.13 than in Figure 4.12 in November. Other than that, a lot stayed the same.
The same observations that were just stated can be seen in the chart containing the data for all three months, chart (d).



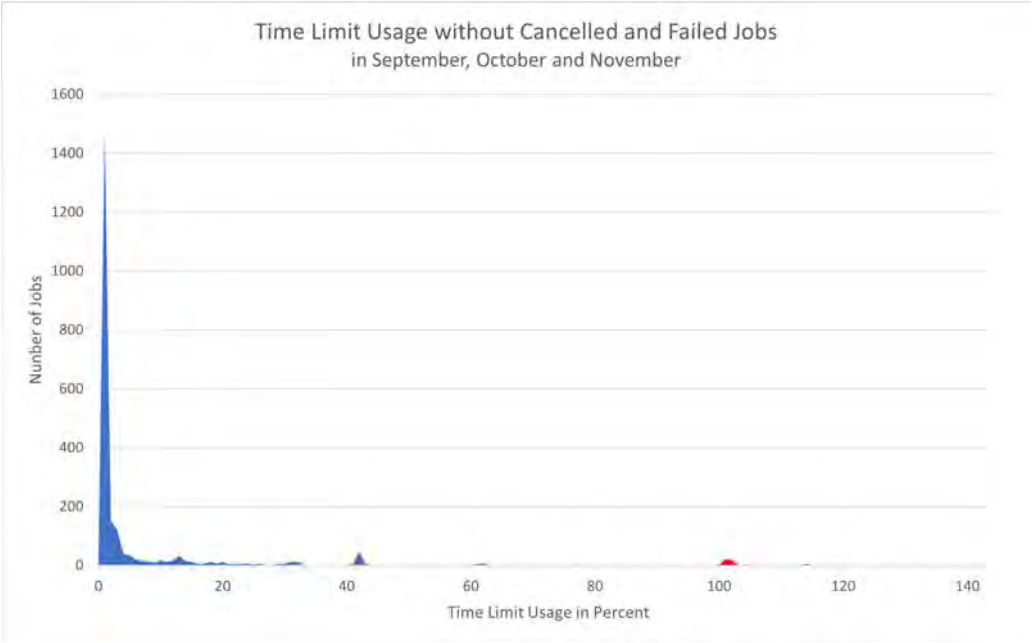
(a) Time Limit Usage without Cancelled or Failed Jobs in September

(b) Time Limit Usage without Cancelled or Failed Jobs in October

Figure 4.13: Charts showing Time Limit Usage without Cancelled or Failed Jobs



(c) Time Limit Usage without Cancelled or Failed Jobs in November



(d) Time Limit Usage without Cancelled or Failed Jobs during all 3 months

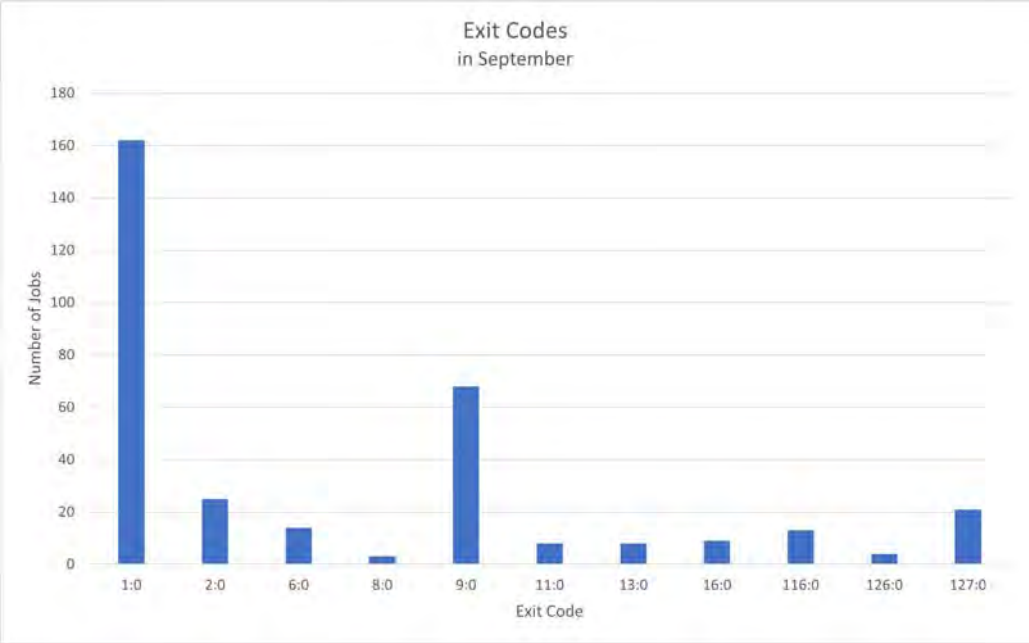
Figure 4.13: Charts showing Time Limit Usage without Cancelled or Failed Jobs

4.6 Exit Codes

This section displays the exit codes with which some of the executed jobs failed. Jobs terminated with the exit code 0:0 were executed successfully, meaning they were completed. Any non-zero values indicate that the job failed. The exit code tells us why a job failed.

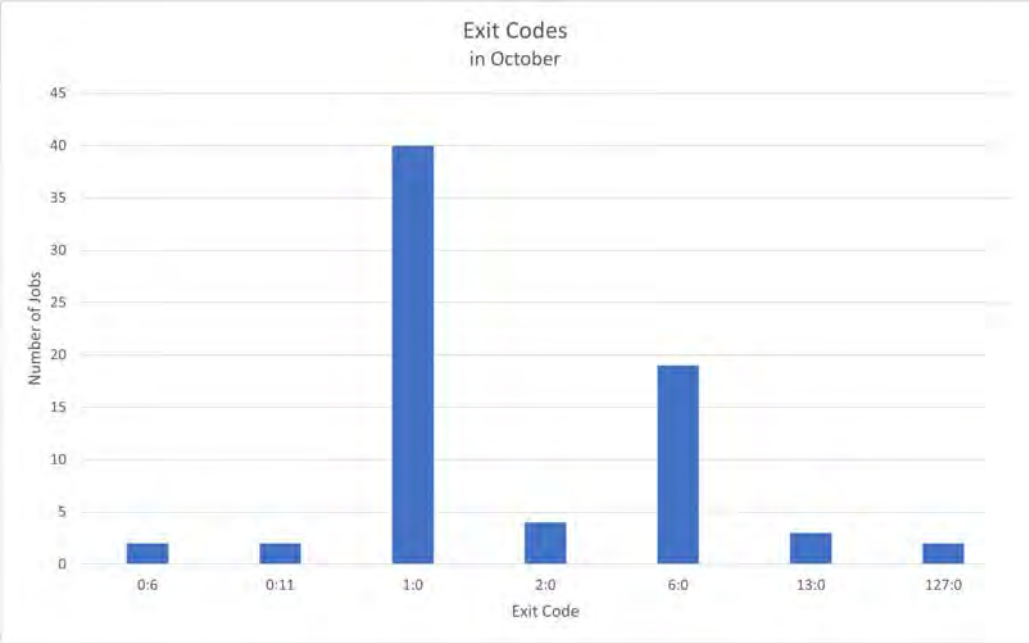
4.6.1 Exit Codes

As seen in (a), most jobs failed with the exit code 1:0 in September. The second most occurring exit code is 9:0. In total, there were eleven different exit codes in September, ranging from 1:0 to 127:0.
In October and November, fewer different exit codes occurred.
Overall, in the three months, thirteen exit codes have arisen. This can be seen in chart (d) of Figure 4.14.

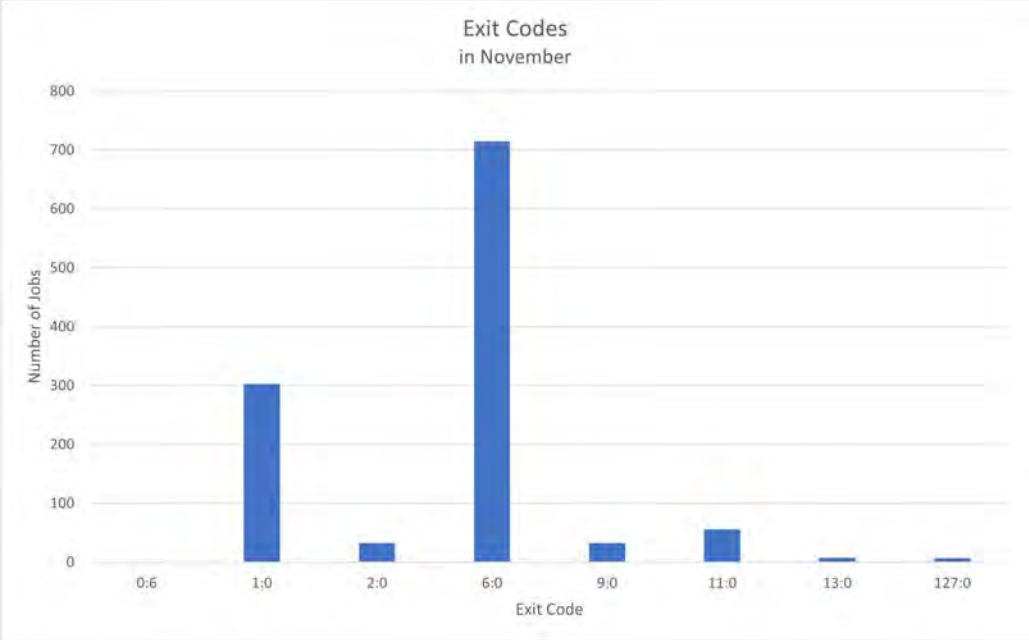


(a) Exit Codes in September

Figure 4.14: Charts showing Exit Codes

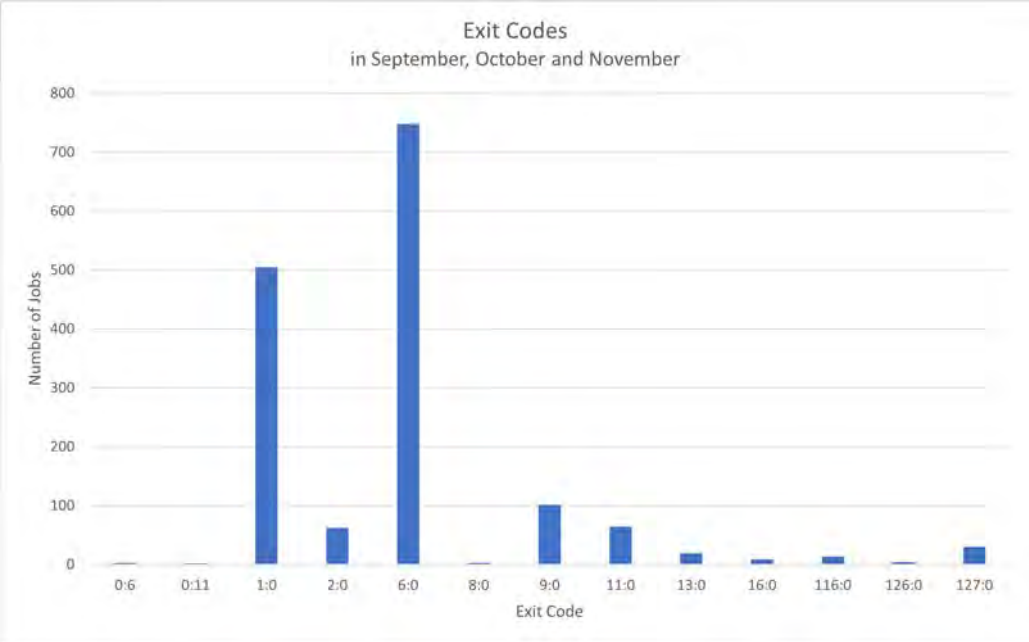


(b) Exit Codes in October



(c) Exit Codes in November

Figure 4.14: Charts showing Exit Codes

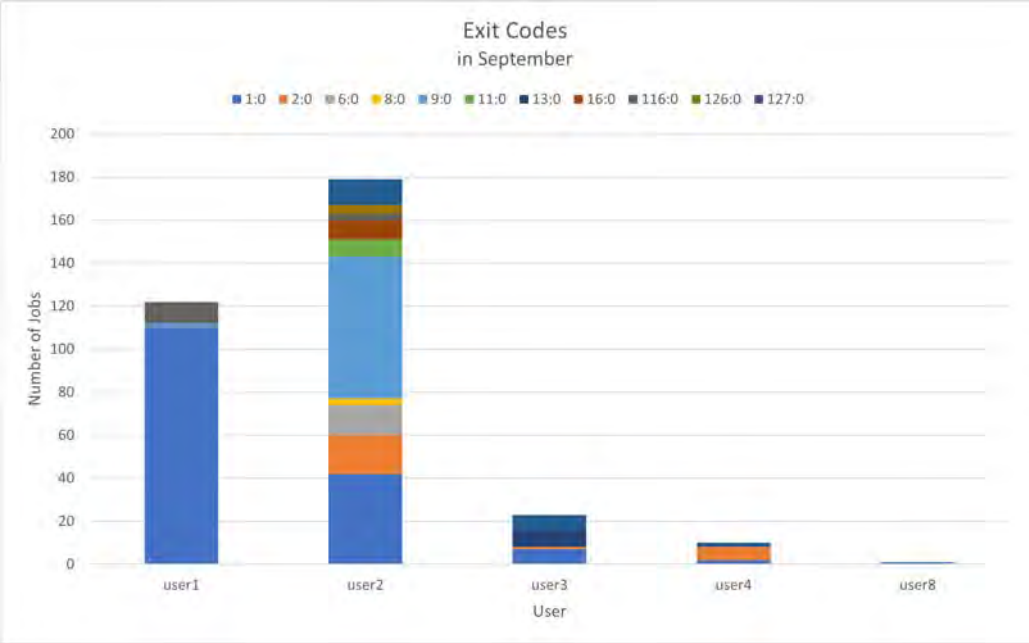


(d) Exit Codes during all 3 months

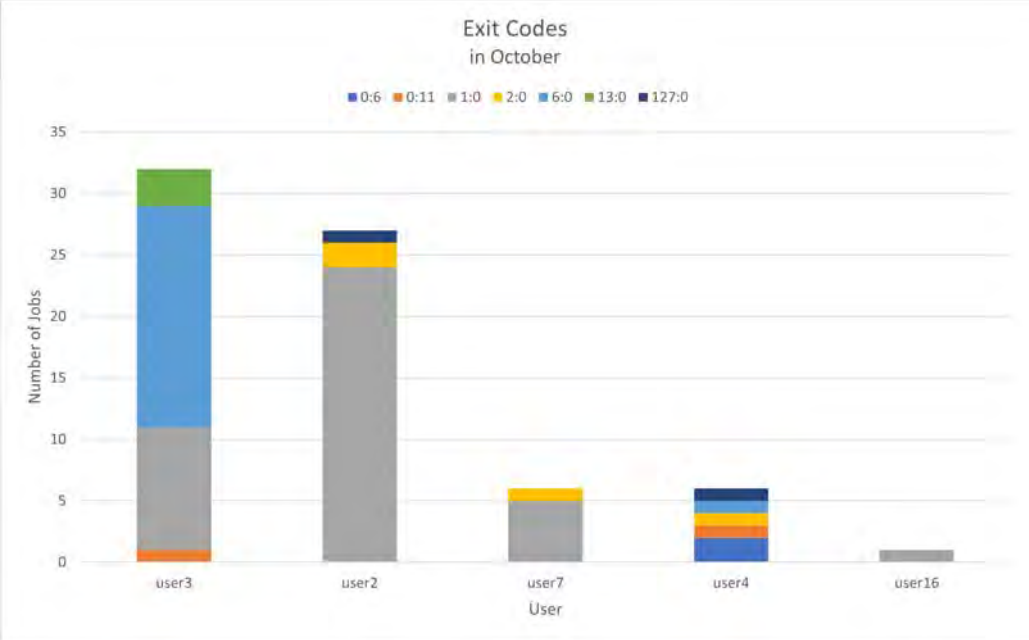
Figure 4.14: Charts showing Exit Codes

4.6.2 Exit Codes per User

Figure 4.15 shows us which exit codes occurred at which user's failed jobs.
With the different users at the horizontal axis and the number of jobs at the vertical axis, we can see in (a) that, in September, user1 had most jobs failing with exit code 1:0, while user2 had numerous different exit codes.
In October and November, the users often had a more significant number of jobs that failed with the same exit codes. For example, user3 had over 700 jobs failing with code 6:0. This can be seen in (c).
Overall, as (d) shows, most exit codes were distributed between two or more users.

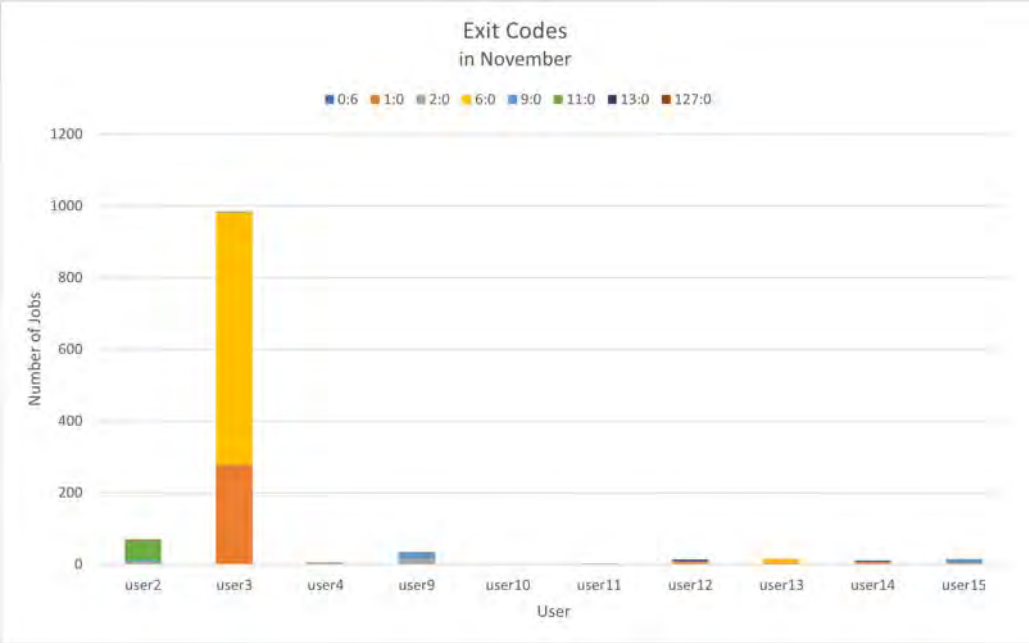


(a) Exit Codes per User in September

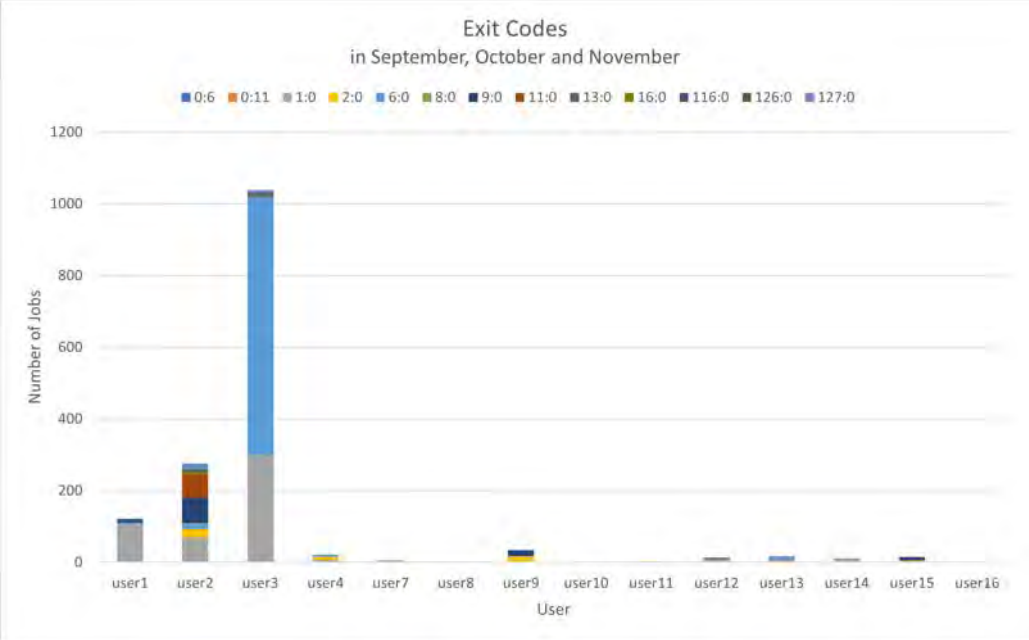


(b) Exit Codes per User in October

Figure 4.15: Charts showing Exit Codes per User



(c) Exit Codes per User in November



(d) Exit Codes per User during all 3 months

Figure 4.15: Charts showing Exit Codes per User

4.7 Consistent Job Characteristics

One objective of this thesis was to identify data outliers or anomalies. To do this, we searched for consistent job characteristics.

4.7.1 Job Name

The initial strategy involved examining a user's jobs with the same name. We wanted to find outliers in the job execution time. We hypothesized that jobs with the same name most likely are in some way related to each other. We assumed these jobs probably did similar things; otherwise, the user would not have chosen the same name. And if the jobs are alike, they may also require the same amount of execution time. Therefore, we compared the duration of these jobs in an attempt to identify a pattern. The discovery of these patterns then could have led us to the outliers. After conducting research, it became evident that the majority of users not only select the same job name for similar jobs but also tend to use the same name consistently. Therefore, this approach could not be used to find data outliers.

4.7.2 Submit Line

The next approach had us look at the executed jobs' submit lines. Here we did not investigate the job execution time but rather the state of the job. We asked ourselves whether, when using the same submit line, some pattern can be found when looking at the state.

sbatch job_mandel-timestep-xeon-nopin-noht-passive-nocache.sbatch	85
CANCELLED	5
COMPLETED	79
FAILED	1

(a) Submit Line with Majority of Jobs Completed

sbatch job_sph-gpu-nopin-noht-passive-nocache.sbatch	60
COMPLETED	1
FAILED	59

(b) Submit Line with Majority of Jobs Failed

sbatch jobscript	33
CANCELLED	11
COMPLETED	9
FAILED	1
TIMEOUT	12

(c) Submit Line with many Timed Out Jobs

Figure 4.16: Different Submit Lines and the Jobs' States

Figure 4.16 depicts different submit lines, how often they have been submitted, and the jobs' states. In (a), we can see that this particular submit line was used 85 times. Five times, either the user or the system administrator cancelled the job. We will disregard this number in this case. Out of the remaining 80 executions, the job failed only once and was completed 79 times. (b) shows us another case, namely one where out of 60 executions, the jobs failed a total of 59 times and were only completed once. In (c), we see a submit line that was run 33 times. Interestingly, while it only failed once, it timed out twelve times, which is more than a third of the total number of executions.

Chapter 5

Discussion

5.1 Error Minimization

One of the goals of this thesis is the minimization of errors. To do this, we are using two different approaches.

5.1.1 Exit Code

For the first approach, we looked at the different exit codes in chapter 4.6. With this data, the users can see with which exit codes their jobs failed the most. Knowing this and understanding the meaning of the exit codes can help the user change their way of using the system to prevent their jobs from failing.
We encountered the following problem while visualizing this data: We do not know the meaning of many of these exit codes. The following table shows some exit codes and their description.
From Figure 4.15, we know that the most occurring exit codes are 1:0 and 9:0. With table 5.1, we can conclude that these jobs failed because of a general failure or some error in the job. This knowledge does not really help the error minimization since we still do not know the exact reason for these job failures and, therefore, can not prevent them.

Exit Codes	
Exit Code	Description
1:0	general failure
2:0	incorrect use of shell builtins
3:0 - 124:0	some error in job
125:0	out of memory
126:0	command cannot execute
127:0	command not found

Table 5.1: Exit Codes and their Descriptions [9]

5.1.2 Submit Line

Another step towards error minimization can be made when looking at the submit lines. Let us have a look at Figure 4.16 in chapter 4.7.2. In (a), leaving the cancelled jobs out, out of the remaining 80 executions, the job failed only once and was completed 79 times. This leads us to believe that if we run this same submit line again, the probability of the job being completed is nearly 100 percent. So we can predict what most probably will happen when submitting the same submit line again.
(b) shows us the exact opposite, where out of 60 executions, 59 resulted in a failure. Here we have two options to try to minimize these failures. The first idea is to tell the user after another failure that their job failed again and that they should investigate the cause in order to minimize the occurrence of these failures. The other option to prevent the failure is to warn the user before submitting this submit line. A possibility would be to print out a warning. For instance: "The submit line you are using resulted in a failure 59 times out of the last 60 executions. Do you still want to run this submit line?" The user could then enter "Y" or "N" to indicate "Yes" or "No." This option would help prevent more jobs from failing.
Looking at (c), we see that more than a third of the executed jobs timed out. Once again, there is the possibility of warning the user either after a job timed out or before. The system could ask for confirmation to execute this exact submit line even when knowing that this submit line tends to time out a lot.

5.2 Job Scheduling

Another goal of this thesis is the simplification of job scheduling which leads to the improvement of system usage. There are various aspects that can be investigated while trying to achieve this.

5.2.1 Investigating System Usage

The first option is to look at how the users utilize the system.

Job Arrivals

First, we observe Figure 4.2, containing the job arrivals. From these charts, we see at what time during the day the most jobs arrived. So, for example, on weekdays in September, most jobs were executed in the afternoon. We also see that there are clear lows. Of course, not many jobs arrived during the night, as the users tend to be asleep then. But there is also a low at around 10 am, which indicates that most users take a (coffee) break. The same can be observed at 1 pm when the people usually are on their lunch breaks. On the weekends, most jobs were executed after 4pm.
If users know at what time most jobs arrive, they can plan around these "rush hours". Running a job at 10 am or 1 pm would make more sense than running it at 3 pm. On the weekend, submitting a job at noon would lead to it being executed way faster than in the evening.

Job Size per User and Individual Job Execution Times

The next idea is observing the number of jobs in relation to the individual job execution times. For this, we look at figures 4.5 and 4.11. For example, if we look at the chart containing the October data, we can see in Figure 4.5 that user4 runs a lot of jobs of size 1. But if we look at Figure 4.11, we see that these are very small jobs since the bar in the plot is very small. Then there is user7,

who only ran 20 jobs of size 1, but these Jobs had an execution time of more than 80'000 seconds. Now let's say a user wants to run a job of size 1. Maybe they know that user7 usually works in the afternoon and user4 in the morning. Now they can run their job in the morning since they know that even though a lot of jobs are executed then, the jobs will not run for a long time.

Job Size and Average Job Execution Time

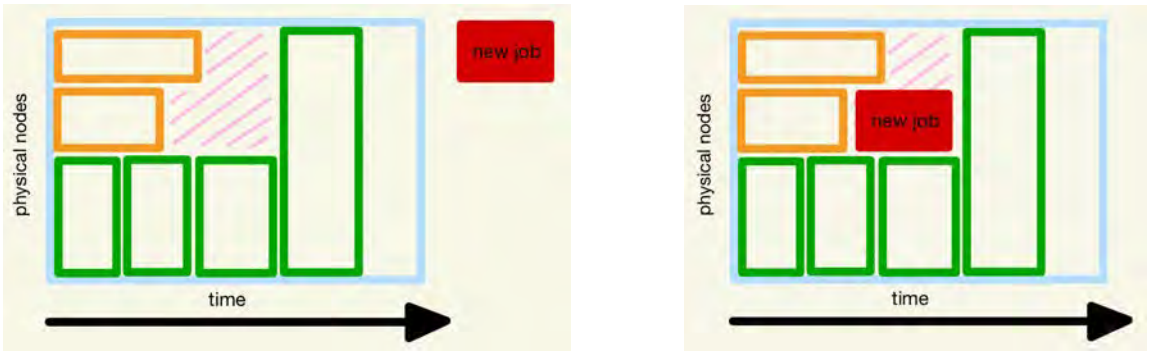
A somewhat similar idea uses the job size and the average job execution time. As can be seen in 4.3 (a), only a small number of jobs of size four get executed. But now looking at 4.10 (a), we see that these jobs take, on average, the longest to complete out of all the job sizes. Knowing this, a user might schedule this job differently from how they would plan it if they did not know the average job execution time.

5.2.2 Maximizing Time Limit Usage

Figure 4.12 displays the results of the time limit usage. We see that only a small number of jobs got timed out during the three months, with none of these timeouts happening in October. This is a remarkable statistic. The issue is the clear peak at around 1 percent. But why is this a problem?

Backfilling

What can be seen in the light blue rectangle in Figure 5.1 are already scheduled jobs in a queue. The jobs on the left side are prioritized to run first. As one can see, there are some empty spots marked as a light pink shaded area. Now, if we want to add a new job, displayed as a red rectangle, to this queue, the job would typically be added at the end of the queue, which here is the far right. But with backfilling, the system checks whether the job could be executed earlier without influencing other scheduled jobs. So if we, for example, have an empty spot on a certain number of nodes for 5 minutes, and we add a job that uses at most this amount of nodes, and we put a time limit of fewer than 5 minutes, this job can be added to this point in the queue. This is shown in (b). This means that we saved time and did not waste resources since we did not have to add the job at the end of our queue.



(a) New Job arrives (b) Backfilling Completed

Figure 5.1: Backfilling

But what happens if our time limit is not accurate? If the time limit for this job in this specific case had been more than 5 minutes, the job would not have met the requirements to fit into this empty spot. This is visualized in Figure 5.2. If this happens, the job has to be added at the end of the queue since it would influence the other already scheduled jobs when trying to fit into the empty spot.
Many users tended to pick a relatively large time limit to ensure that their job finished in time. One of the most used time limits was 16:30 hours, even though most of these jobs then only ran for a few seconds. As Figure 5.2 shows, such extensive time limits can be bad for job scheduling, and they waste a lot of resources.

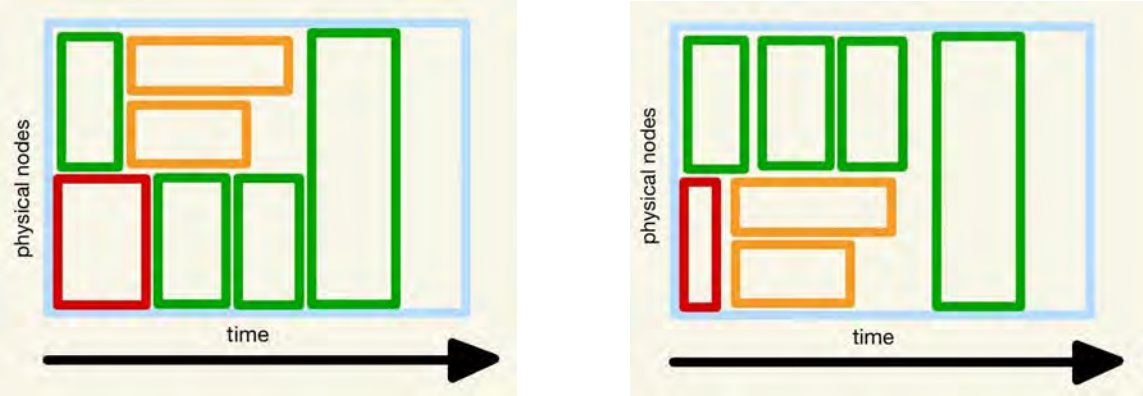
(a) New Job arrives (b) Backfilling Failed

Figure 5.2: Failed Backfilling

Compression

Another problem can be seen in Figure 5.3. On the left side, once again, the scheduled jobs are displayed. We see that the first two jobs to be executed are the red and the green one, followed by the two orange ones and then the other green ones. The orange ones start running as soon as there

is an open spot. We predicted that the green job would finish first, meaning the orange ones would run on the same nodes the green one did. If the red job's time limit estimate is inaccurate and the job finishes early, the orange ones will run on these nodes. Meaning not only will the predicted start times of the following jobs change but also the job-to-node allocation. One can see this on the right side. This can be a problem since now our predictions are wrong. Maybe we wanted the orange jobs to run on some particular nodes or start at a specific time. This change could have been prevented if more exact time limits had been chosen.



(a) Before Compression

(b) After Compression

Figure 5.3: Compression

5.3 Energy Efficiency

As already discussed in section 5.2.1, the job arrival times can be used to enhance job scheduling. Another usage of these charts displayed in Figure 4.2 is the improvement of energy efficiency. Thanks to these graphs, we know at what time during the week or on the weekend the least amount of jobs is running. For example, looking at (d), we can see that during the week, only a minimal number of jobs arrived between 0:30 am and 4 am. Knowing this, some nodes could be shut down during this time, leading to fewer resources being used. This would result in the system running more energy efficiently.

Chapter 6

Conclusion

The goal of this thesis was to visualize and analyze HPC job data of the miniHPC to find ways to improve the performance, resilience and energy efficiency of the HPC system. We wanted to find ways to enhance job scheduling, job anomaly detection and user feedback.
This was done using the sacct command to download the data from Slurm, which was then visualized using excel pivot tables and charts. By doing so, various ways of possible improvement regarding job scheduling have been found. We learned that an easy way to efficient job scheduling is looking at the data and observing the behaviour and usage of system users. Understanding how many jobs of what sizes are executed and how long the execution takes on average, knowing at what time of day most jobs arrive, and gaining an understanding of a user's job size and job state tendencies are, alongside others, important aspects when wanting to improve the job scheduling.

Additionally, we found a way to reduce the waste of resources by looking at the time limits and observing that most users utilize inaccurate time limits to make sure that their jobs terminate instead of trying to calculate appropriate time limits, which could then enable backfilling.

Furthermore, the job arrival time charts can be used to enhance the system's energy efficiency, as they show at what time the least amount of jobs arrive, which could result in nodes being shut down during this time.

6.1 Future Work

As HPC systems and their users change significantly over time, future work can always be done. Some examples are listed here:

- The charts displayed in this work could be created for other months, maybe over the span of a year or even more, to result in more meaningful charts.
- Like most HPC systems, the miniHPC system has many more job status fields that could be analyzed to gain more insights into the system and its users.
- The suggestions stated in chapter 5.1.2 could be applied to the system to enhance error minimization.

Bibliography

[1] High Performance Computing. https://www.techtarget.com/searchdatacenter/definition/high-performance-computing-HPC (accessed December 25, 2022)

[2] miniHPC. https://hpc.dmi.unibas.ch/en/research/minihpc/ (accessed December 14, 2022)

[3] Dror G. Feitelson: Workload Modeling for Computer Systems Performance Evaluation. 2015

[4] Pascal Kunz: HPC Job-Monitoring with SLURM, Prometheus and Grafana. Bachelor Thesis. 2022

[5] Aldi Dyla: Visual Analysis of Job Accounting Information of High Performance Computing Systems. Bachelor Thesis. 2020

[6] Slurm Workload Manager. https://en.wikipedia.org/wiki/Slurm_Workload_Manager (accessed December 14, 2022)

[7] sacct Command. https://manpages.ubuntu.com/manpages/bionic/man1/sacct.1.html (accessed December 14, 2022)

[8] Job Accounting Fields. https://slurm.schedmd.com/sacct.html (accessed December 14, 2022)

[9] Exit Codes and their Description. https://hpc-discourse.usc.edu/t/exit-codes-and-their-meanings/414 (accessed November 15, 2022)