Comparative Analysis of Historical Job Data Across High Performance Computing Systems

Bachelor Thesis

University of Basel
Faculty of Science
Department of Mathematics and Computer Science
High Performance Computing Group

Advisor and Examiner: Prof. Dr. Florina M. Ciorba
Supervisor: Dr. Osman Simsek

Author: Alexander Lutsch
Email: alexander.lutsch@stud.unibas.ch

June 16, 2025

Acknowledgements

I would like to express my sincere gratitude to the High Performance Computing Group at the University of Basel and to Prof. Dr. Florina M. Ciorba, under whose supervision this Bachelor thesis was conducted.
I am thankful to Thomas Jakobsche and Prof. Dr. Florina M. Ciorba for proposing the topic of Comparative Analysis of Historical Job Data Across High Performance Computing Systems.
Finally, I am especially grateful to Dr. Osman Simsek for his continuous support and hands-on guidance throughout the development of this thesis.

Abstract

This thesis analyses job data from three distinct High Performance Computing (HPC) systems to propose actionable efficiency improvements. We introduce the framework of Operational Data Analysis (ODA) in similar work and apply both descriptive and prescriptive ODA across the datasets. The analysis aims to identify patterns, anomalies, and actionable improvements.

Our approach involves a thorough examination of each dataset, followed by methodical both within and across the datasets. Key findings include the prevalence of short, single-node workloads that are not well-suited to current HPC hardware, as well as the identification of a small group of approximately 20 power users responsible for nearly half of all job submissions on all datasets. Despite their experience, these users frequently overestimate wallclock times.

Based on these findings, we propose targeted improvements: implement workshop and feedback mechanisms to help power users improve their wallclock time estimations; schedule short jobs more efficiently using aggressive backfilling strategies and a dedicated high-throughput queue; and procure hardware specifically optimized for short, single-node workloads.

We further explore job simulators to demonstrate the impact of the stated improvements. To this end, we set up the OpenDC job simulator with our target hardware and workloads. However, the simulation results were limited by constraints within the OpenDC framework As a way forward, we suggest switching to a more suitable simulator or combining multiple simulators, such as OpenDC and Batsim—to leverage their respective strengths and produce more meaningful results.

To our knowledge, this study is the first to derive general HPC system recommendations based on the cross-analysis of multiple real-world HPC job datasets, offering a general foundation for HPC efficiency improvements.

Contents

1 Introduction **3**
- 1.1 Operational Data Analytics . . . 3
- 1.2 Motivation . . . 3
- 1.3 Challenges . . . 4
- 1.4 Goals . . . 4

2 Related Work **5**

3 Methods **7**
- 3.1 Technical Analysis Setup . . . 7
  - 3.1.1 Parquet . . . 7
- 3.2 Analysis Methodology . . . 8
  - 3.2.1 Dataset Analysis Framework . . . 10
- 3.3 Job Simulator . . . 13

4 Analysis **14**
- 4.1 Hardware . . . 14
- 4.2 Marconi 100 Analysis . . . 14
- 4.3 Fugaku Analysis . . . 20
- 4.4 Eagle Analysis . . . 23
- 4.5 Similarities . . . 26
- 4.6 Anomalies . . . 26
- 4.7 Recommendations . . . 26

5 Simulation **28**
- 5.1 Available Simulators . . . 28
- 5.2 OpenDC Setup . . . 29
- 5.3 Simulation Results . . . 30

6 Discussion **34**
- 6.1 ODA of HPC Systems . . . 34
- 6.2 Simulation with OpenDC . . . 34

7 Conclusion **35**
7.1 Cross-System Operational Data Analysis 35
7.2 Outlook 35

A Figures **39**
A.0.1 Cross-System Analysis 39
A.0.2 Fugaku-Specific Analysis 62
A.0.3 Marconi-Specific Analysis 72
A.0.4 Eagle-Specific Analysis 82

B Topology Setup **86**

Chapter 1

Introduction

1.1 Operational Data Analytics

High-Performance Computing (HPC) systems form an important pillar in our modern society by enabling fields like earthquake prediction, climate analysis and artificial intelligence. In recent years both the application fields and demand for processing power are growing explosively [23], especially with the trend of deep learning AI. This results in the emergence of the first exascale systems that face significant operational challenges due to unprecedented complexity. To fulfill this operational demand, improving the speed, efficiency, and reliability of these systems is vital. A term named Operational Data Analytics (ODA) was coined by Bourassa et al. [3] in the context of optimizing cooling systems at the National Energy Research Scientific Computing Center (NERSC) and expanded on by Netti et al.[20] into a general conceptual Framework with the goal of "continuous monitoring, archiving, and analysis of near real-time performance data, providing immediately actionable information for multiple operational uses."[21] In this bachelor thesis we aim to produce descriptive and prescriptive ODA on the application layer across different HPC system datasets by analyzing their submitted job data. In our analysis, we will identify patterns and anomalies within each individual dataset, as well as across all datasets collectively. Based on these findings, we will provide recommendations for potential improvements. Additionally, we will implement an HPC simulator to analyse the results of our datasets and to evaluate the impact of proposed changes. To our knowledge, this is the first academic study that systematically compares HPC job data across multiple supercomputing systems.

1.2 Motivation

By analyzing historical job data multiple high-performance computing (HPC) systems, we aim to produce valuable insights for operational improvements. We will be able to visualize and examine both user and system behavior, allowing us to identify patterns and anomalies. With these insights we can form recommendations for improving user behavior, scheduling strategies and resource allocation. To evaluate the potential impact of these improvements, we will employ an HPC simulator. Given the scale of HPC environments, even small improvements scale up to significant gains in speed and efficiency.

1.3 Challenges

Publicly available HPC Job datasets are rare, The three datasets explored in this thesis have all been released recently: the Marconi100 dataset from 2023 [2], the NREL Eagle dataset from 2023 [7] and the Fugaku dataset from 2024 [1]. Dataset size can also be a constraint, for example the Marconi100 dataset has a total size of 400GB. However, it is a manageable constraint since the job data subset size, in this case, is at most 27GB. Moreover, available HPC simulators are relatively niche tools with limited documentation and unique technical constraints. We will explore these simulators as part of the thesis.

1.4 Goals

We methodically analyse metrics related to job submissions, scheduling, and anonymized user behavior from three HPC systems. Our goal is to identify patterns and anomalies across these systems to uncover insights and form improvement proposals. These proposed changes will then be tested using an HPC simulator to evaluate their potential effectiveness in realistic scenarios.

Chapter 2

Related Work

A methodological groundwork for what we are doing was laid out by Netti et al. called "Operational Data Analytics" [20], combining the popular HPC frameworks "4-Pillar Framework for Energy-Efficient HPC Data Centers" [24] and the "Four Types of Data Analytics" [13]. The 4 pillar framework "provides a classification [...] in the form of building infrastructure, system hardware, system software and applications." [20] The four types of data analytics "describe the typology of the underlying analytics techniques. The four types are descriptive, diagnostic, predictive and prescriptive analytics." [20] This thesis focuses on the Applications pillar, which concerns "individual workloads as well as the workload mix executed on a system. An application can be considered a unit of work, since the goal of an HPC system is to find new scientific insight using software applications" [24]. As our work involves the analysis of historical HPC job data, we are particularly concerned with identifying behavioral patterns and system dynamics. Our analytical approach falls under both descriptive and prescriptive analytics as we summarize historical data to uncover patterns and test actionable recommendations for improving workload management and system efficiency.

For ODA, the operational data of an HPC system needs to be collected in the first place. "In many cases, the data collection agents are proprietary or self-developed software that are tailor-made to the specific requirements of the site." [21] The data collection implementations are also the most labor-intensive parts of the ODA process. On the other hand, channeling the data over a message bus to databases, and analyzing the data is generally done over off the shelf tools. [21] Figure 2.1 shows an overview of different HPC facilities (rows) and the tools they use for the ODA process (columns).

The comprehensive survey by Ott et al. [21] shows that the primary use cases for ODA can be grouped into three categories: optimizing facility infrastructure, managing power and energy and improving strategic planning. A lot of analytical tools exist for the purpose of descriptive analysis, a topic we are also focusing on in our work. An example implementation can be seen by Chan [5] using Grafana for historical and real time resource analytics for the "Savio Supercluster". Descriptive research on historical HPC job data, similar to this thesis, has also been conducted by Chu et al. [6], who analysed node energy consumption, job failures, and node-job correlations on SURF Lisa—a Dutch HPC system comprising 338 nodes and primarily used by universities and researchers. Our approach is novel in that we analyse data across multiple HPC systems, enabling cross-system comparisons and broader insights.

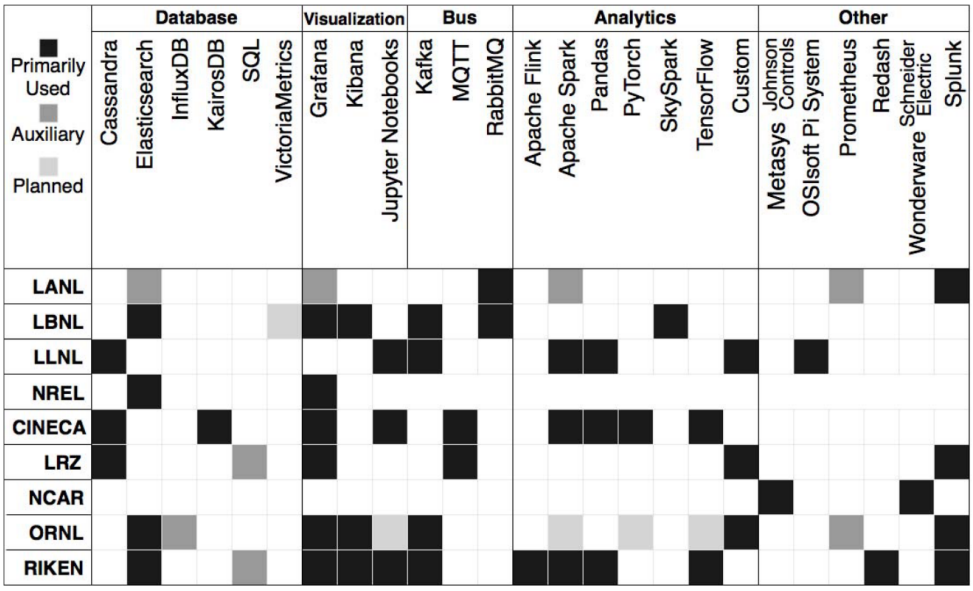


Figure 2.1: ODA Tools used by different HPC facilities. Source: [21]

In addition to descriptive analytics, predictive analytics also has been growing in recent research. One area involves predicting the runtime of submitted jobs using various machine learning techniques [17]. Other studies explore machine learning-based online performance prediction for parallelization and task scheduling [14], as well as classification and prediction of job resource consumption on HPC platforms [9].

Chapter 3

Methods

3.1 Technical Analysis Setup

We are interested in the publicly available job data of three HPC systems:

Fugaku: Provided by the RIKEN Center for Computational Science. Available at https://zenodo.org/records/11467483. Download size: 26 GB.

M100: From CINECA. Dataset accessible via https://data.openei.org/submissions/5860. Download size: 401 GB.

Eagle: From the U.S. Department of Energy's (DOE's) National Renewable Energy Laboratory (NREL). Available at https://gitlab.com/ecs-lab/exadata/-/tree/main. Download size: 240 MB.

While the *Fugaku* and *Eagle* datasets are already job-centric, the *Marconi 100* dataset contains additional metrics such as power consumption and temperature, which are outside the scope of our analysis. Therefore, we extract only the relevant job data of the Marconi100 dataset by filtering for the `job_table` plugin in the directory structure. The resulting subset of job data has a size of 217 MB.
The full analysis workflow is done using Python with a range of Python libraries. All datasets are provided in the `.parquet` format and are loaded into memory using the `pyarrow` library. The data is then converted into a `pandas` DataFrame, enabling flexible table manipulation and analysis. Visualizations are generated using the `Matplotlib` and `seaborn` libraries, while regression analysis is conducted with `SciPy`.
Reading the parquet files into the panda dataframe requires north of 256GB memory. We will be using a high-performance computing cluster funded by the University of Basel, "miniHPC" to proces large in memory computations.

3.1.1 Parquet

"Apache Parquet is an open source, column-oriented data file format designed for efficient data storage and retrieval." [22]. An advantage of the column oriented data structure is its positive

impact on data compression since values of the same type are grouped together. Before compression, Parquet applies encoding strategies to represent the data more compactly. After encoding, a compression algorithm is applied to the data. Parquet supports a range of compression codecs, allowing for optimization based on the data's characteristics [22]. The format has become a popular tool for storing and processing data in analytics pipelines.

3.2 Analysis Methodology

Each dataset offers a unique set of features (columns). Our objective is to explore these features both individually and across datasets. We construct a table with the datasets as columns and the available features as rows. Next, we group related features into broader "dimensions" to help formulate methodical analysis questions based on both dimensions and datasets.

Table 3.1: Grouping relevant features of Marconi100, Fugaku and Eagle datasets.

Dimension	Feature	Marconi100	Fugaku	Eagle
Scheduling	Wall-clock run time	`run_time` (seconds)	`duration` (seconds)	`run_time` (seconds)
	Job submission time	`submit_time` (datetime)	`adt` (datetime)	`submit_time` (datetime)
	Job start time	`start_time` (datetime)	`sdt` (datetime)	`start_time` (datetime)
	Job end time	`end_time` (datetime)	`edt` (datetime)	`end_time` (datetime)
	Job completion state	`job_state` (`CANCELLED`, `FAILED`, `COMPLETED`, `TIMEOUT`, `OUT_OF_MEMORY`, `NODE_FAIL`, `PREEMPTED`)	`exit state` (`completed`, `failed`)	`state` (`CANCELLED`, `COMPLETED`, `FAILED`, `TIMEOUT`, `NODE_FAIL`, `OUT_OF_MEMORY`, `PENDING`, `RUNNING`)
	Reason for pending/failed state	`state_reason` (string)		
	Average job idle time		`idle_time_ave` (seconds)	
	Threads per core required by job	`threads_per_core`		
	Number of tasks requested by job	`num_tasks`		

Continued on next page

Table 3.1 – continued from previous page

Dimension	Feature	Marconi100	Fugaku	Eagle
	CPU statistics		`perf1` (exec cycles) `perf6` (sleep cycles) `sctmut` (sys. CPU time, ms) `usctmut` (user CPU time, ms)	
	Tasks per node/socket	`ntasks_per_node` `ntasks_per_socket`		
Resources	Requested wall-clock time	`time_limit` (minutes)	`elpl` (seconds)	`wallclock_req` (seconds)
	Requested processors	`tres_req_str`	`cnumr`	`processors_req`
	Required cores per socket	`cores_per_socket`		
	Requested nodes	`tres_req_str`	`nnumr`	`nodes_req`
	Allocated nodes	`tres_alloc_str`	`nnuma`	
	Allocated CPUs	`tres_alloc_str`	`cnumat`	
	Requested memory	`tres_req_str`		`mem_req`
	Allocated memory	`tres_alloc_str`	`msza` (only 3 unique values)	
	Requested GPUs	`tres_req_str`		`gpus_req`
	Allocated GPUs	`tres_alloc_str`		
	Used amount of processors		`cnumut`	
	Used amount of nodes		`nnumu`	
	Job can share nodes	`shared` (string: '0' or 'OK')		
	Contiguous nodes requested	`contiguous`		
	Used memory		`mmszu`	

Continued on next page

Table 3.1 – continued from previous page

Dimension	Feature	Marconi100	Fugaku	Eagle
	Node frequency requested		`freq_req` (megahertz)	
	Node frequency allocated		`freq_alloc` (megahertz)	
	Floating point ops/sec		`flops`	
	Memory bandwidth		`mbwidth` (bytes)	
	Memory limit exceeded		`mem_per_cpu` (per CPU) `mem_per_node` (per Node) (boolean)	
	Performance class		`pclass` (e.g. memory- or compute-bound)	
User	Associated user name	`user_id`	`usr`	`user`

3.2.1 Dataset Analysis Framework

Based on the available feature and dimension information, we are now able to formulate targeted analysis questions for each dataset.

1. **Cross-System Analysis (Marconi, Fugaku, Eagle)**

   (1.1) **Metadata**

   1.1.1. Total number of submitted jobs.

   1.1.2. Timeframe of job submissions.

   1.1.3. Overview of the hardware setup.

   (1.2) **Scheduling Analysis**

   1.2.1. Histogram of wall-clock run time, including the median, to analyse job duration (A.1).

   1.2.2. Histogram of job wait time (submission time - start time), including the median, to analyse job scheduling duration (A.2).

   1.2.3. Histogram of job exit states (A.3).

   1.2.4. Analysis of job arrivals throughout day, week and year to highlight workload submission load balancing (A.4, A.5, A.6).

   1.2.5. Analysis of wait times throughout day, week and year to highlight periods with the lowest average wait time (A.7, A.8, A.9).

(1.3) **Resource Analysis**

1.3.1. Histogram of requested nodes per job (A.10, A.11, A.12).

1.3.2. Histogram of requested processors per job (A.13).

(1.4) **Scheduling vs. Resources**

1.4.1. Histogram of job node seconds (wall-clock run time × requested nodes) (A.14).

1.4.2. Histogram of job processor seconds (wall-clock run time × requested processors) (A.15).

1.4.3. Analysis of time limit usage for completed jobs to determine how much users overestimate wallclock runtime (A.16).

1.4.4. Comparison of wait time and requested processor wall-clock to analyse if larger resource requests increase wait times (A.17).

1.4.5. Comparison of unsuccessful job rate and processor wall-clock runtime to analyse if long high resource jobs are more likely to fail (A.18).

(1.5) **Scheduling vs. User Behavior**

1.5.1. Top 20 users by job submission count to identify power users queuing a significant amount of jobs (A.19).

1.5.2. Comparison of wallclock usage efficiency and number of submitted jobs to analyse if power users are better at estimating wallclock time (A.20).

1.5.3. Top 20 users by total overestimated processor time to identify the least efficient users in wall-clock estimation (A.21).

1.5.4. Top 20 users by total wall-clock processor-seconds to identify top users by allocation area (A.22).

1.5.5. Highlighting top 20 users by submitted jobs, total processor time and overestimated requested time to see overlap in the usage metrics (A.23).

2. **Fugaku-Specific Analysis**

(2.1) **Scheduling**

2.1.1. Comparison of execution cycles and wallclock runtime to analyse scaling behavior (A.24).

2.1.2. Comparison of sleep cycles and wallclock runtime to analyse scaling behavior (A.25).

2.1.3. Analysis of the execution-to-sleep cycle ratio compared to wallclock runtime (A.26).

(2.2) **Resources**

2.2.1. Comparison of requested and used processors to determine actual usage (A.27).

2.2.2. Comparison of requested and used processors grouped by requested processors (A.28).

2.2.3. Comparison of requested and allocated nodes to determine allocation (A.29).

2.2.4. Comparison of requested and used nodes to determine actual usage (A.30).

2.2.5. Comparison of requested and used nodes grouped by requested nodes (A.31).

2.2.6. Comparison of allocated and used memory to determine over allocation of memory (A.32).

2.2.7. Analysis of used memory and floating-point operations (flops) to see if flops scale with memory usage (A.33).

2.2.8. Analysis of flops and memory bandwidth to determine if higher bandwidth results in higher flops (A.34).

2.2.9. Analysis of used memory vs. memory bandwidth to determine if memory usage influences bandwidth (A.35).

2.2.10. Comparison of requested and allocated nodes grouped by requested nodes (A.36).

(2.3) **Scheduling vs. Resources**

2.3.1. Relationship between user/system CPU time and memory bandwidth (A.37).

2.3.2. Analysis of average job idle time vs. used nodes to determine if using more nodes leads to longer idle times (A.38).

2.3.3. Analysis of execution-to-sleep cycles ratio compared to used processors to analyse if using more processors leads to a higher ratio of sleep cycles (A.39).

(2.4) **Resources vs. User Behavior**

2.4.1. Analysis of user ID vs. underused resources to identify users who consistently underutilize their requests (A.40).

3. **Marconi-Specific Analysis**

(3.1) **Scheduling**

3.1.1. Histogram of SLURM tasks requested by jobs (A.41).

(3.2) **Resources**

3.2.1. Histogram of requested memory amount (A.42).

3.2.2. Histogram of requested GPU amount (A.43).

3.2.3. Relationship between requested CPUs and GPUs for jobs demanding both (A.44).

3.2.4. Comparison of requested vs. allocated CPUs to determine over-allocation (A.45).

3.2.5. CPU Allocation Percentage grouped by requested processors (A.46).

3.2.6. Comparison of requested and allocated memory to determine over-allocation (A.47).

3.2.7. Memory Allocation Percentage grouped by requested memory (A.48).

3.2.8. Comparison of requested and allocated nodes to determine over-allocation (A.49).

3.2.9. Comparison of requested and allocated GPUs to determine over-allocation (A.50).

(3.3) **Scheduling vs. Resources**

3.3.1. Comparison of timeout percentage and requested wall-clock time to determine if high or low runtime estimates are more likely to time out (A.51).

3.3.2. Comparison of job cancellation percentage and wait time to determine if longer wait times lead to more cancellations (A.52).

3.3.3. Comparison of unsuccessful job rate and requested memory to analyse if memory requirements impact completion probability (A.53).

4. **Eagle-Specific Analysis**

(4.1) **Resources**

4.1.1. Histogram of requested memory amount (A.54).

4.1.2. Histogram of requested GPU amount (A.55).

4.1.3. Scatter plot analyzing the relationship between requested CPUs and GPUs for jobs demanding both (A.56).

(4.2) **Scheduling vs. Resources**

4.2.1. Comparison of timeout rate and requested wall-clock time to determine if high or low estimates are more likely to time out (A.57).

4.2.2. Comparison of job cancellation percentage and wait time to determine if longer wait times lead to more cancellations (A.58).

4.2.3. Comparison of unsuccessful job rate and requested memory to analyse whether memory requirements impact completion probability (A.59).

3.3 Job Simulator

We start by comparing a range of available job simulators to evaluate their suitability for our thesis. After formulating improvements, we aim to set up virtual HPC systems that replicate the configurations observed in our datasets.

These virtual systems will be used to simulate job execution using our dataset workloads. By comparing the simulated output with the actual workload data, we can compare the accuracy of the simulation and establish a baseline behavior of the simulator.

Next, we introduce specific changes such as queue adjustments or scheduling policy modifications into the simulator. By modifying both simulator parameters and workload characteristics, we can explore how these changes affect system performance and job behavior. The results will be visualized to evaluate the impact of our recommendations.

Chapter 4

Analysis

The complete set of analysis results is provided in Appendix A and is also organized and referenced in Section 3.2.1. This chapter provides a summary of the main findings and highlights the most significant results.

4.1 Hardware

Specification	Marconi	Fugaku	Eagle
Total number of nodes	980	158,976	2,114
CPU cores per node	32	48	36
Available memory per node	256 GB	32 GiB	1728 nodes: 96GB 288 nodes: 192GB
Available GPUs per node	4	0	50 nodes with 2 GPUs

Table 4.1: Hardware Overview of the Marconi, Fugaku, and Eagle Systems [10][15][18]

4.2 Marconi 100 Analysis

The Marconi 100 dataset comprises 6,236,347 job entries submitted between May 5, 2020, and September 28, 2022. An analysis of the wallclock runtime histogram 4.1 reveals a median job duration of only two seconds, indicating a strong prevalence of short-running tasks. Furthermore, Figure 4.2 shows that a vast majority of these jobs request only a single node.
These characteristics display that the Marconi100 system is primarily utilized for short, single-node computations rather than large-scale, high-performance computing tasks.

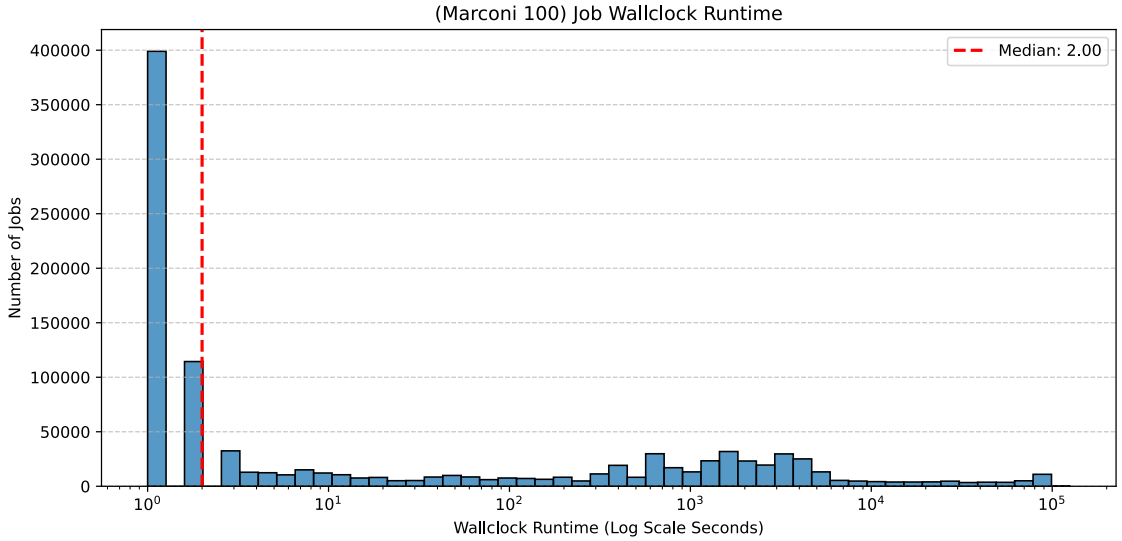


Figure 4.1: Marconi 100 job wallclock runtime distribution.

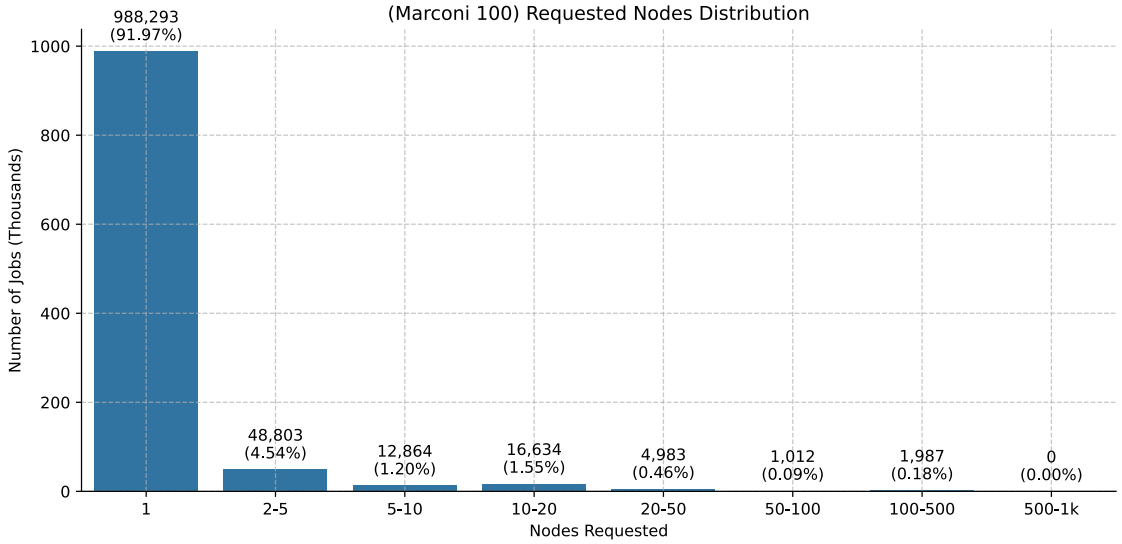


Figure 4.2: Marconi 100 requested nodes distribution.

Even though the majority of jobs are short and single node, the time delay between submission and scheduling (wait time) remains relatively high as seen in the wait time histogram 4.3. Paired with a very inaccurate wallclock time estimation across all users as seen in Figure 4.4 **we suspect that the wait times are relatively high due to inefficient wallclock time estimations.** We see a correlation between the requested wallclock processor seconds (wallclock $\times$ requested processors) and the requested wallclock runtime in Figure 4.5, however the high variation and the low R^2 value leave room for uncertainty.

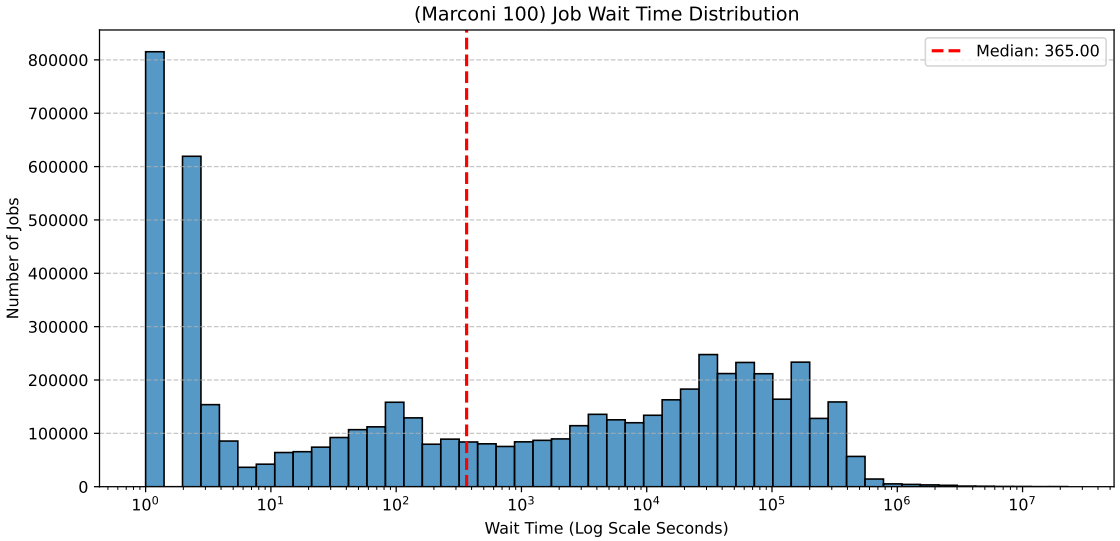


Figure 4.3: Marconi 100 job wait time distribution.

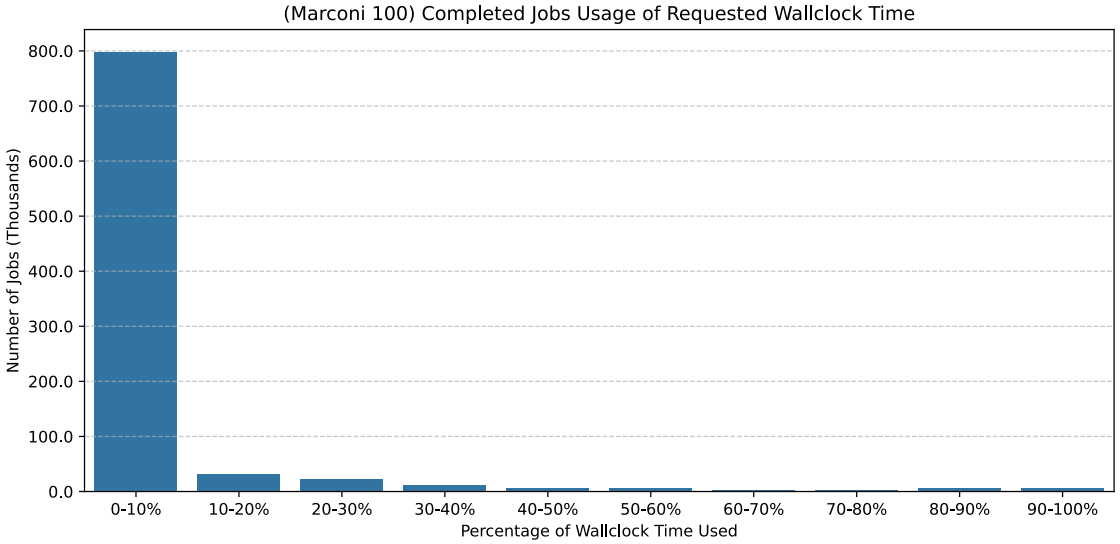


Figure 4.4: Marconi 100 usage of requested wallclock time.

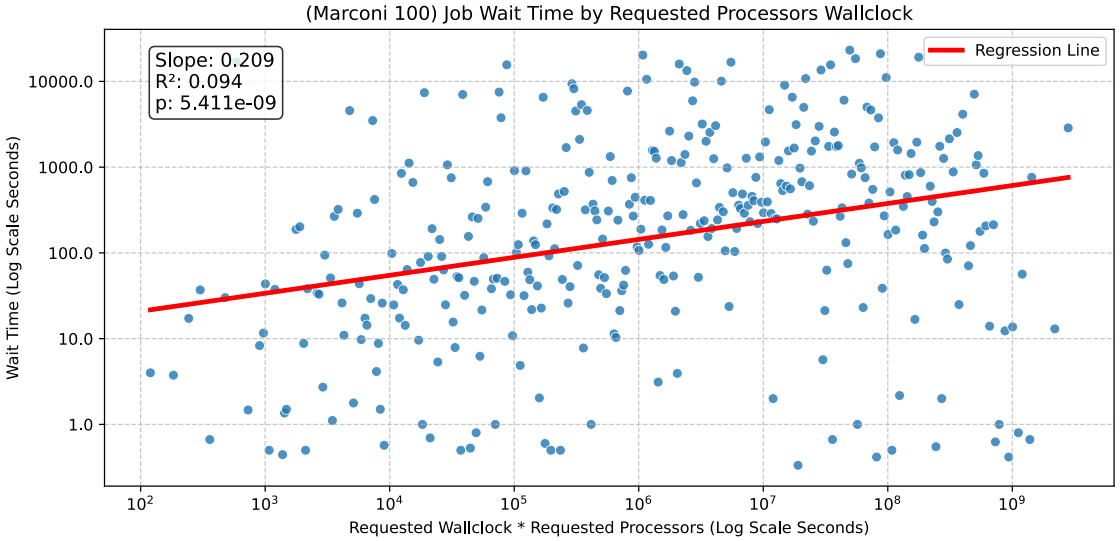


Figure 4.5: Marconi 100 linear regression of wait time and requested processor wallclock seconds.

When looking at user behavior we observe a correlation in job cancellation and waiting times (4.6). Up to 10^4 seconds (∼166 minutes) the job cancellation rate scales linearly to 40% and with waiting times longer than 10^6 seconds (∼11 days) we see a cancellation rate of up to 100%.

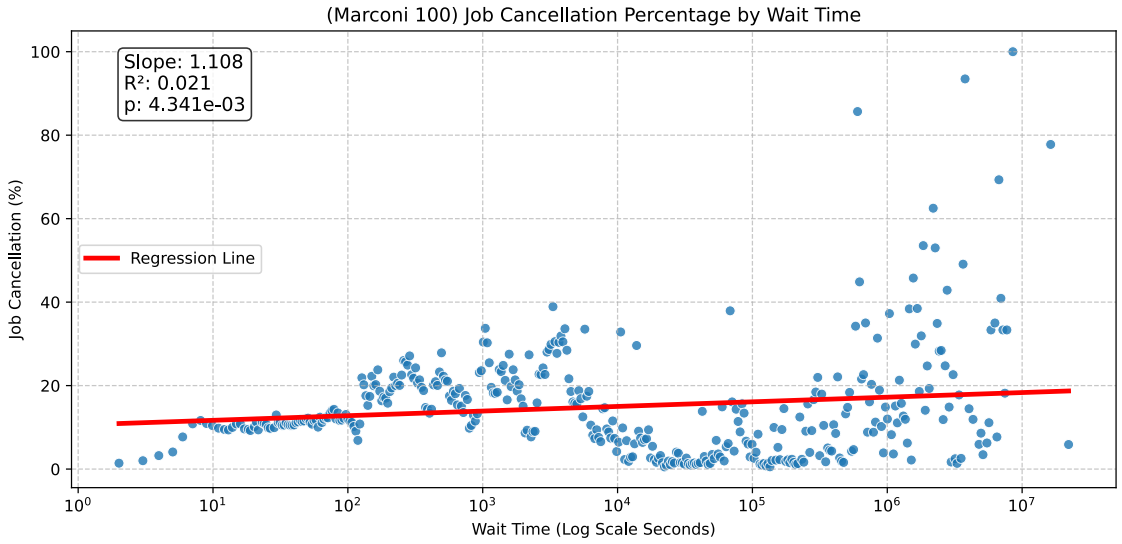


Figure 4.6: Marconi 100 linear regression of wait time and job cancellations.

When looking at total job submissions we see a small group of 20 users accounting for around 60% of total job submissions (4.7). **The Marconi100 system has a small set of power users accounting for over half of job submissions.** Analysis of the relationship between wallclock estimation efficiency and the number of jobs submitted per user reveals no discernible correlation (Figure 4.8). **Users with extensive job submission experience do not demonstrate improved accuracy in estimating wallclock usage.**

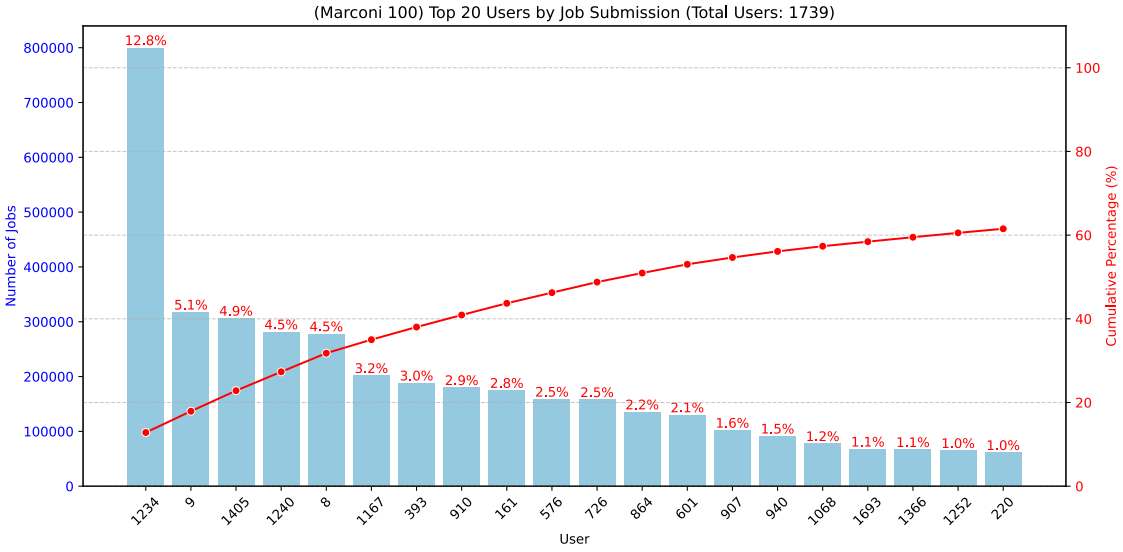


Figure 4.7: Marconi 100 linear regression of wait time and job cancellations.

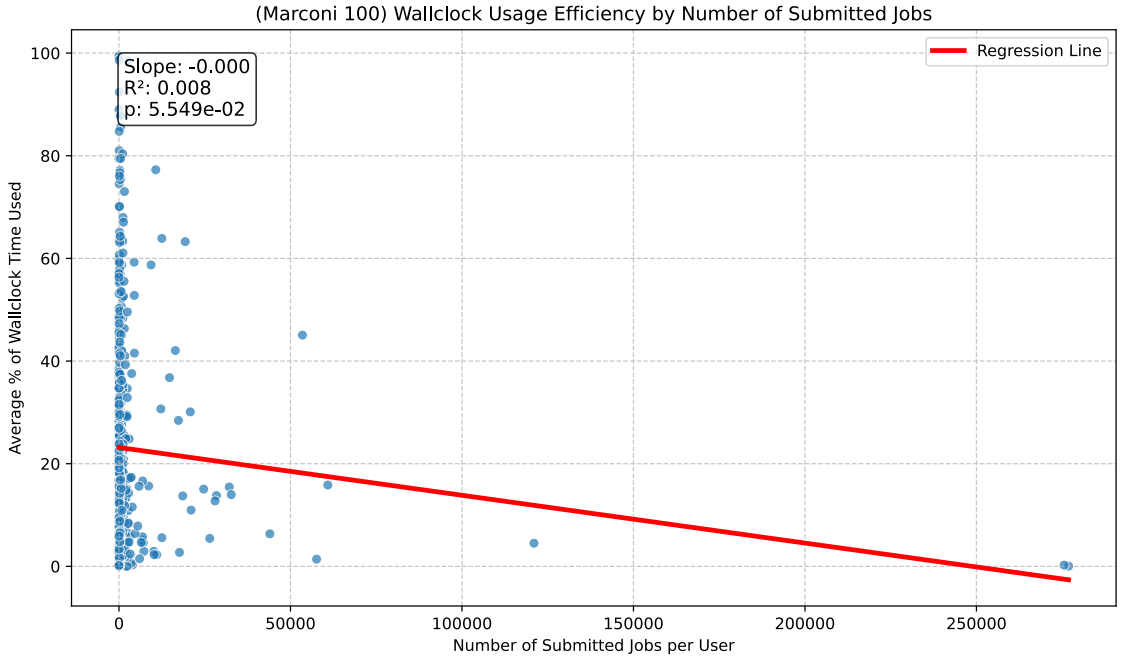


Figure 4.8: Marconi 100 linear regression of wallclock usage efficiency and number of submitted jobs.

4.3 Fugaku Analysis

The Fugaku dataset comprises 25,866,900 job entries submitted between March 1, 2021, and April 30, 2024. The median job wallclock runtime is 1253 seconds (∼21 minutes) 4.9 with 60% of jobs being single node (4.10). **The Fugaku system is utilized for long single to multi node workloads.** The majority of jobs use 10% or less of the estimated runtime (4.11). **The Fugaku**

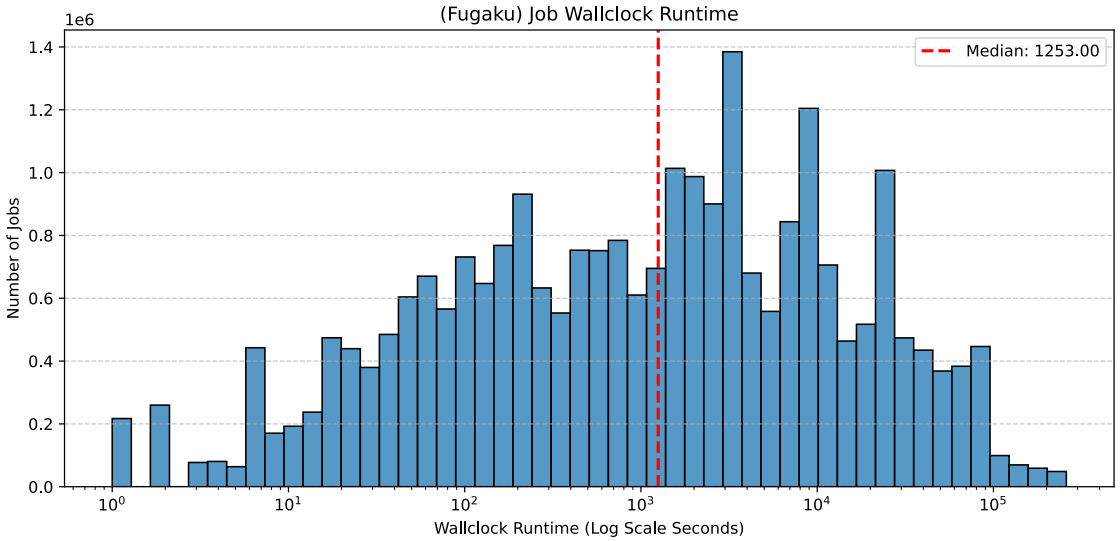


Figure 4.9: Fugaku job wallclock runtime distribution.

workloads show relatively poor runtime estimations. When looking at total job submissions we see a small group of 20 users accounting for around 40% of total job submissions (4.12). **The Fugaku system has a small set of power users accounting for 40% of job submissions.** Analysis of the relationship between wallclock estimation efficiency and the number of jobs submitted per user reveals no discernible correlation (Figure 4.11). **Users with extensive job submission experience do not demonstrate improved accuracy in estimating wallclock usage.**

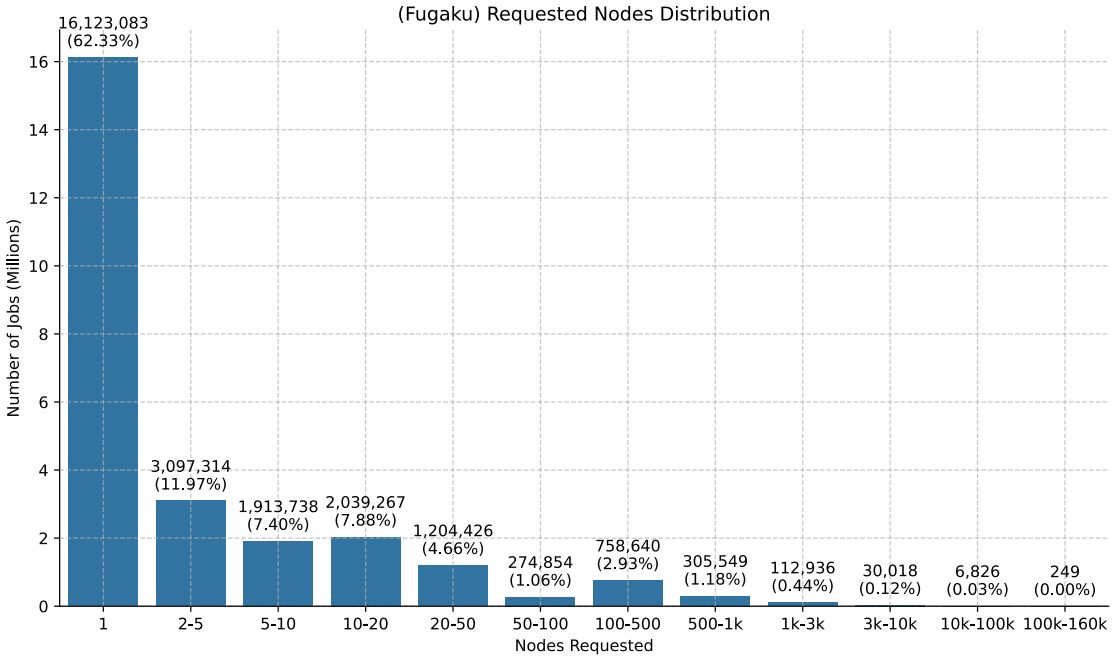


Figure 4.10: Fugaku requested nodes distribution.

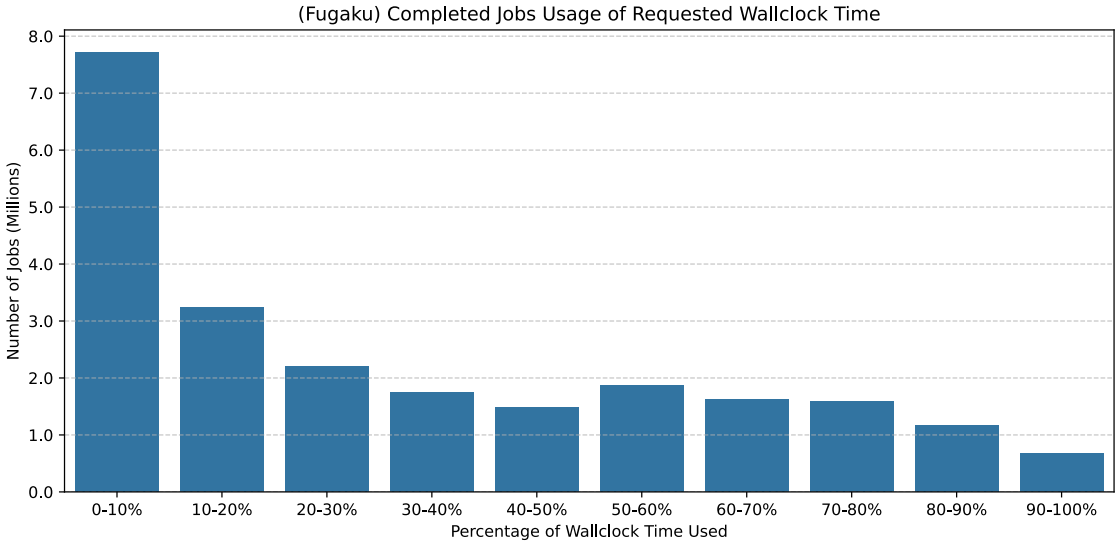


Figure 4.11: Fugaku usage of requested wallclock time.

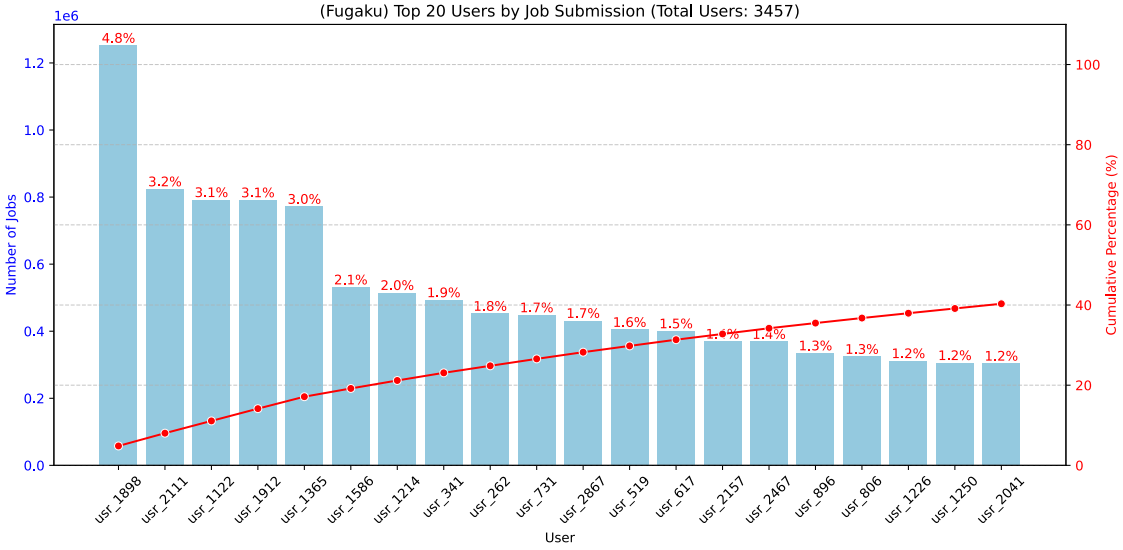


Figure 4.12: Fugaku linear regression of wait time and job cancellations.

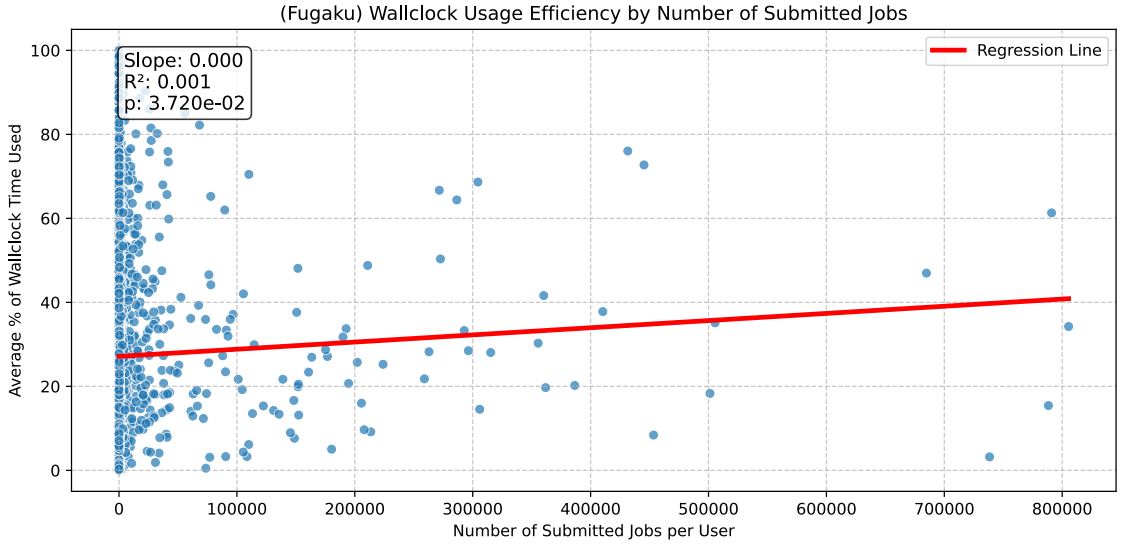


Figure 4.13: Fugaku linear regression of wallclock usage efficiency and number of submitted jobs.

4.4 Eagle Analysis

The Eagle dataset comprises 11,030,377 job entries submitted between November 14, 2018 and February 02, 2023. The job wallclock histogram 4.14 reveals an even distribution of short and long jobs, with a median job wallclock runtime of 336 seconds. The node distribution 4.15 shows a very strong preference of single node workloads of 90%. **The Eagle system is primarily utilized for short to long single-node computations.** The majority of jobs use 10% or less of the

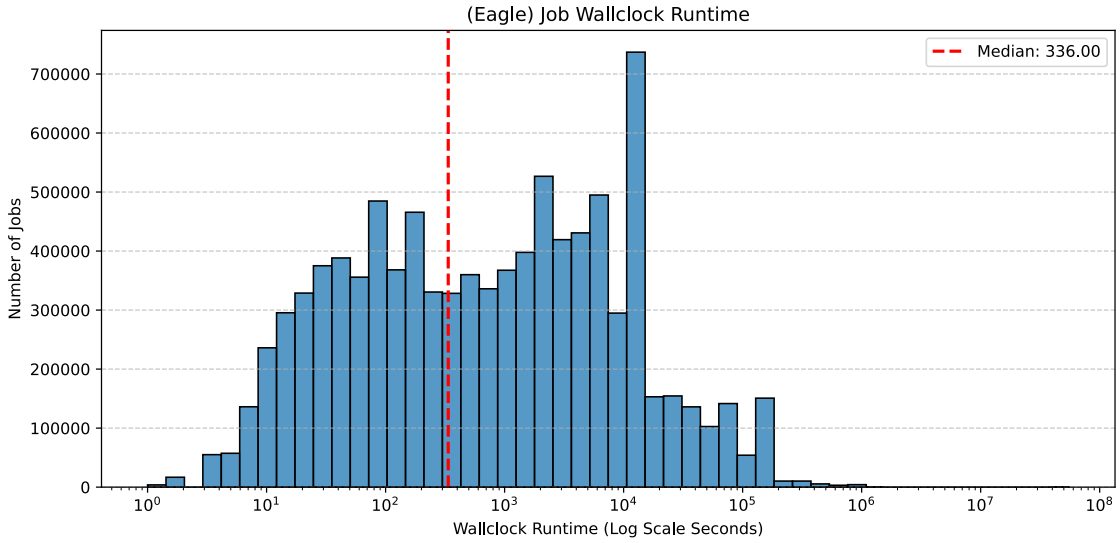


Figure 4.14: Eagle job wallclock runtime distribution.

estimated runtime (4.16). **The Eagle workloads show relatively poor runtime estimations.** When looking at total job submissions we see a small group of 20 users accounting for around 50% of total job submissions 4.17. **The Eagle system has a small set of power users accounting for 50% of job submissions** Analysis of the relationship between wallclock estimation efficiency and the number of jobs submitted per user reveals no discernible correlation (Figure 4.16). **Users with extensive job submission experience do not demonstrate improved accuracy in estimating wallclock usage.**

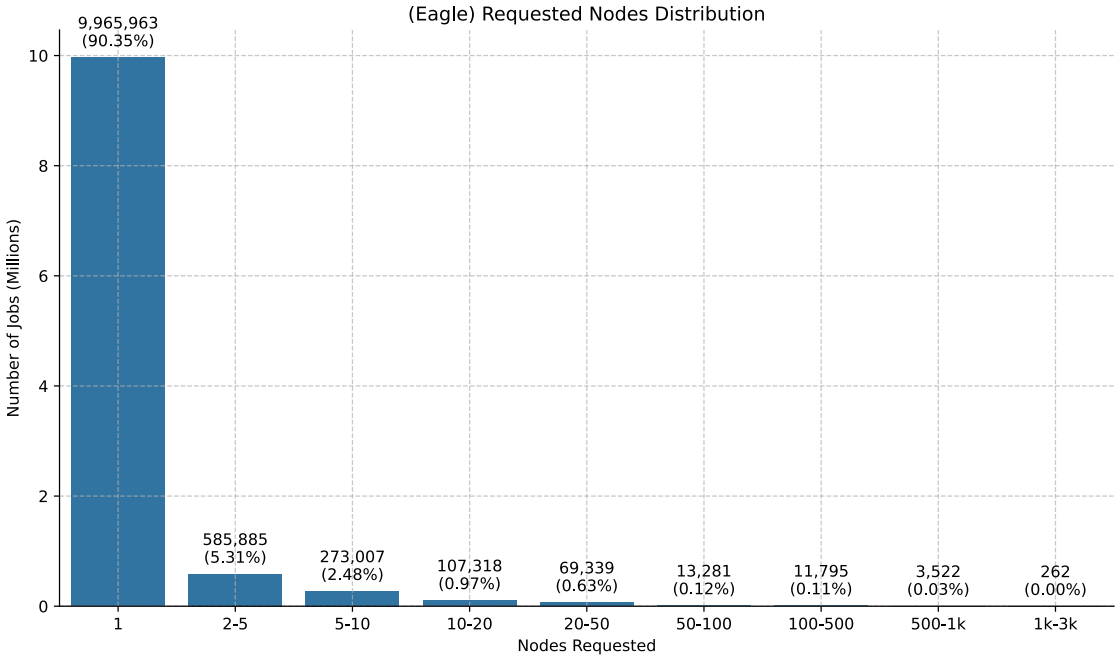


Figure 4.15: Eagle requested nodes distribution

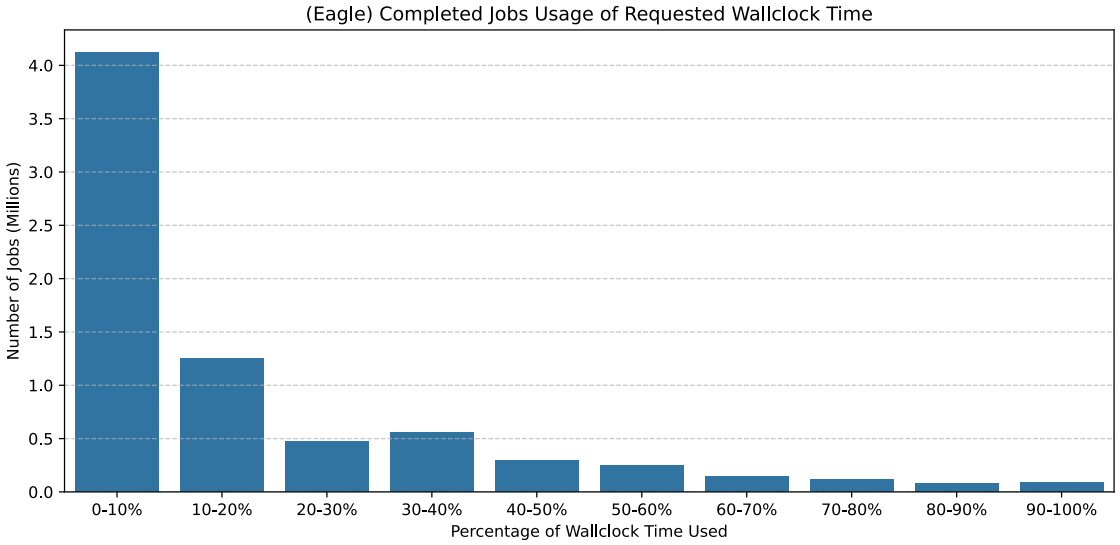


Figure 4.16: Eagle usage of requested wallclock time.

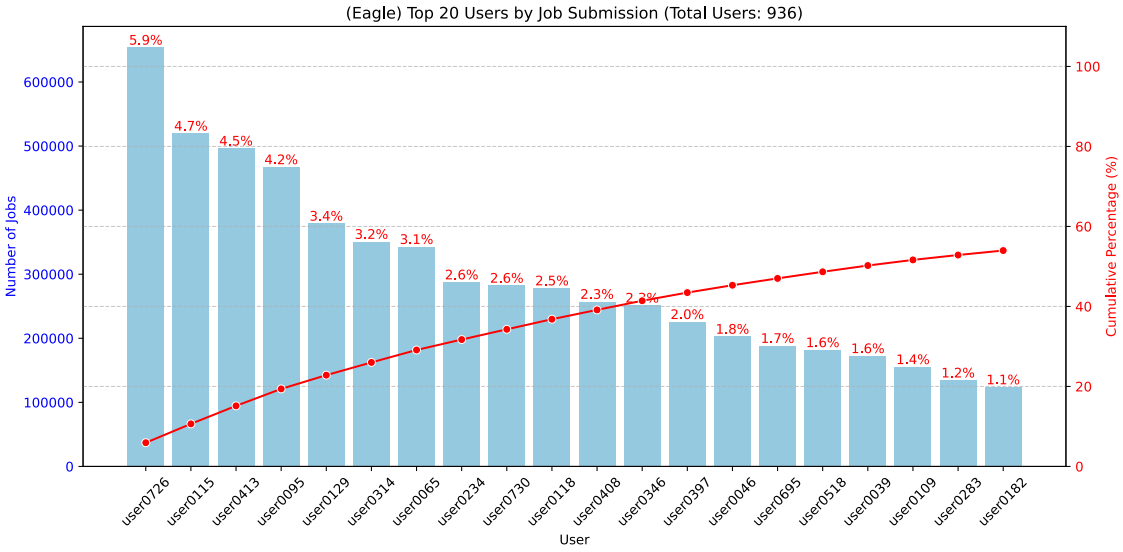


Figure 4.17: Eagle linear regression of wait time and job cancellations.

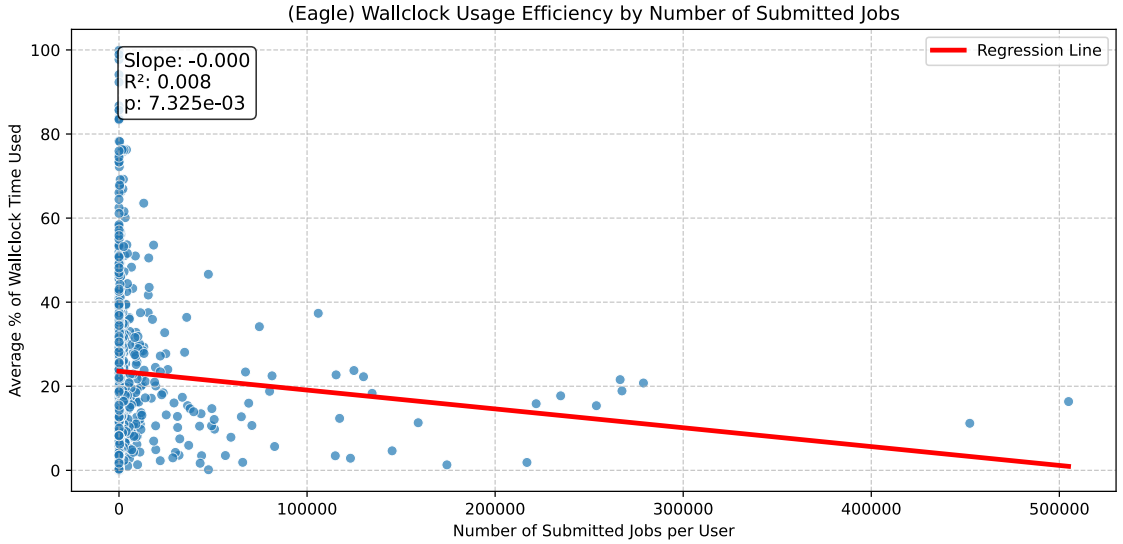


Figure 4.18: Eagle linear regression of wallclock usage efficiency and number of submitted jobs.

4.5 Similarities

- All systems exhibit poor wallclock time estimation accuracy (A.16).
- A small subset of users contributes the majority of submitted jobs on all systems (A.19).
- Users with high submission frequency do not demonstrate improved wallclock estimation accuracy (A.20).
- The majority of jobs on all systems are single-node workloads (A.10, A.11, A.12).
- System utilization peaks between 9 AM and 12 PM, and is lowest outside these hours (A.4, A.5, A.6), resulting in reduced queue wait times between midnight and 9 AM (A.7, A.8, A.9).
- On Marconi and Eagle, GPU job requests show a partial correlation between the number of requested processors and GPUs. However, a significant portion of the data also shows fixed configurations, resulting in mixed patterns (A.44, A.56).
- Wait time correlates with requested wallclock processor runtime (wallclock $\times$ processors) across all systems (A.2).
- On Marconi and Eagle, we see a correlation between wait time and job cancellations. Wait times exceeding 10^5 seconds ($\sim$28 hours) have up to 100% cancellation rate (A.52, A.58).

4.6 Anomalies

- Marconi shows a disproportionately high number of short-duration, single-node jobs (A.1, A.10).
- Wallclock estimation errors on Marconi are worse compared to the other systems (A.16).
- In contrast to Marconi, Fugaku overallocates nodes by an average of 300%, with single-node workloads contributing the most by being overallocated by approximately 500% on average (A.36, A.49).

4.7 Recommendations

Our analysis reveals several areas for possible improvement. A significant finding is that a small subgroup of "power users" is responsible for a majority of the workload on all systems, however these experienced users show no corresponding improvement in the accuracy of their wallclock time estimations. This suggests that influencing a small group of users would result in a high degree of change. Simple workshops or an automated post-job feedback system (e.g., "1 hour out of an estimated 30 hours wallclock time used"), would have an influence on these users and lead to more efficient scheduling. Predictive runtime estimation tools could also contribute to more accurate wallclock time predictions.
Furthermore, we have a prevalence of single-node workloads on all systems. From a scheduling perspective, policies like aggressive backfilling or creating dedicated high-throughput queues can be implemented to optimize for this dominant job type. From a hardware perspective, single nodes

with high flops per core and high memory for high-throughput serial processing could be added instead of focusing solely on parallel computing.
All systems show underutilization during night hours, indicating potential for improved load balancing. Incentivizing job submissions during off-peak times could lead to more efficient resource utilization.
Finally, we see that long wait times lead to high rates of user cancellations. Providing users with an estimated start time could help them make better decisions, reducing the number of jobs left in the queue only to be cancelled later.

The following concrete improvements are proposed:

- Offer workshops for power users to improve wallclock runtime estimation practices.
- Deploy an automated post-job feedback system to help users refine their future runtime estimates.
- Apply aggressive backfilling strategies to increase scheduling efficiency for short jobs.
- Establish a dedicated high-throughput queue and procure hardware for short, single-node workloads.
- Offer incentives to encourage job submissions during off-peak hours.
- Improve queue transparency by providing users with estimated job start times.

Chapter 5

Simulation

5.1 Available Simulators

We evaluated several available job simulators relevant to HPC analysis.

SimGrid is a long-established, high-performance simulation framework written in C++. It provides scalable models for computation and networking and serves as the simulation backend for other job simulators, for example Batsim. [4]

GridSim is an older Java-based simulator designed for Grid environments. Though no longer maintained, it influenced the design of ALEA.[19]

Batsim builds on SimGrid and provides a decoupled architecture where scheduling logic is separated from simulation execution.[8] It is convenient for research focused on custom scheduler development and has plenty of user documentation.

AccaSim is a Python-based HPC workload simulator, however it has not seen active development since 2018 and has very sparse user documentation. [11]

ALEA is an extension of GridSim with a library of standard scheduling algorithms. Like AccaSim, it is no longer actively maintained and doesn't have a lot of user documentation. [12]

OpenDC is a modern, cloud-focused simulator built in Kotlin with a visual, web-based interface. It is actively developed and supports modeling of serverless and ML workloads. [16] Its focus on ease of use and support for quick topology building make it an attractive choice, though it offers less flexibility in low-level scheduling.

Our final decision was between **Batsim** and **OpenDC**, both of which are actively maintained and provide sufficient user documentation. For this thesis, we chose to proceed with **OpenDC**. This decision was primarily based on the fact that OpenDC's input requirements are well-aligned with our available hardware and workload data, and its simulation output is comparably informative to that of Batsim.

While Batsim offers greater flexibility for scheduler customization, it relies on SimGrid to define the hardware topology, which requires detailed and complex specification of nodes and interconnects. In contrast, OpenDC supports a much simpler JSON-based topology configuration. This significant advantage in ease of use, combined with easy integration of our datasets, made OpenDC the choice for this work.

5.2 OpenDC Setup

OpenDC requires three fundamental input files to run a simulation: the **topology** of the system, the **workload** and the **experiment** configuration. The experiment file defines the scheduling policy, output targets, and links to both the topology and workload files.

We begin by defining a topology file for each of the three systems used in this study: Marconi100, Fugaku, and Eagle (B.1) (B.2) (B.3). A limitation of OpenDC is its current inability to natively schedule jobs across multiple nodes. To accommodate this, we represent each system as a single large node containing all hardware resources.

Next, we define the workload for OpenDC. The simulator expects the workload to be provided in two `.parquet` files: `tasks.parquet` and `fragments.parquet`. These files must follow specific Apache Arrow schemas (5.1) (5.2). The purpose of the `tasks` file is to define the workload arrival time, length and hardware requirements. The `fragments` file serves to dynamically change the workload requirements of the `tasks` workload. We will use the `fragments` file to model the actual wallclock runtime while putting the wallclock runtime estimation into the `tasks` file. Another caveat is the presence of `cpu_capacity` and `cpu_usage` fields that require MHZ values. These are irrelevant in our analysis as we do not model power requirements or model the slowdown of tasks when supplied with insufficient `cpu_capacity`. We generate both of these files from our own workload files using python.

Listing 5.1: Schema for tasks.parquet defined in Python.

```
job_schema = pa.schema([
    pa.field("id", pa.string(), nullable=False),
    pa.field("submission_time", pa.int64(), nullable=False),
    pa.field("nature", pa.string(), nullable=True),
    pa.field("deadline", pa.int64(), nullable=True),
    pa.field("duration", pa.int64(), nullable=False),
    pa.field("cpu_count", pa.int32(), nullable=False),
    pa.field("cpu_capacity", pa.float64(), nullable=False),
    pa.field("mem_capacity", pa.int64(), nullable=False),
])
```

Listing 5.2: Schema for fragments.parquet defined in Python.

```
fragment_schema = pa.schema([
    pa.field("id", pa.string(), nullable=False),
    pa.field("duration", pa.int64(), nullable=False),
    pa.field("cpu_count", pa.int32(), nullable=False),
    pa.field("cpu_usage", pa.float64(), nullable=False),
])
```

Finally we define an `experiment` json file (5.3) that links to the topology and the workloads (workload folder containing tasks and fragments parquet files). The allocation policy is "provisionedCores", in this case it does not make a difference however, since we defined big single node systems and OpenDC doesn't have workload allocation logic to split a workload among different nodes. For the export model we are interested in task, host and service metrics as these contain the relevant

job information. The export interval prints the state of the virtual machine, as we are only interested in the finished state of the machine and not intermediate states, we set this to a very high value to have one finished output. The scaling is set to "NoDelay", this disables any task slowdown simulation with the MHZ logic of `cpu_capacity`.

Listing 5.3: Experiment JSON for OpenDC

```
{
    "name": ";Simulation",
    "topologies": [
        {
            "pathToFile": "topology/toplogy.json"
        }
    ],
    "workloads": [
        {
            "type": "ComputeWorkload",
            "pathToFile": "workload"
                        "scalingPolicy": "NoDelay"
        }
    ],
    "allocationPolicies": [
        {
            "type": "prefab",
            "policyName": "ProvisionedCores"
        }
    ],
        "exportModels": [
        {
            "exportInterval": 999999999,
            "printFrequency": 360,
            "filesToExport": [
                "task",
                "host",
                "service"
            ]
        }
    ]
}
```

5.3 Simulation Results

The simulated results of Marconi Fugaku and Eagle show that the runtime distribution (5.1) is an exact replica of the original runtime. This occurs due to the discrete-event simulation scheduling of OpenDC. When comparing the simulated wait time distributions, we see much fewer jobs appearing

in the distribution in the first place. This is due to the scheduling field being often empty in the simulator output, we assume an empty scheduling field of the simulator output signalizes no wait time. This shows the simulator being much more efficient at scheduling, probably a result of our single node topology setup. Furthermore, node sharing is currently not simulated and leads to high processor overallocation as seen in Figure A.45.

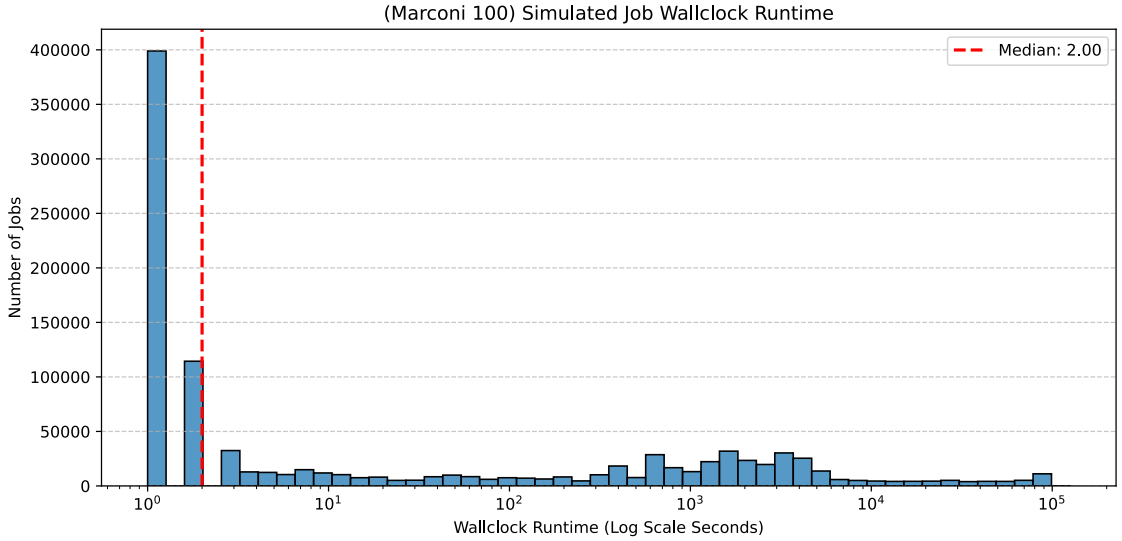


Figure 5.1: Marconi 100 simulated job wallclock runtime distribution.

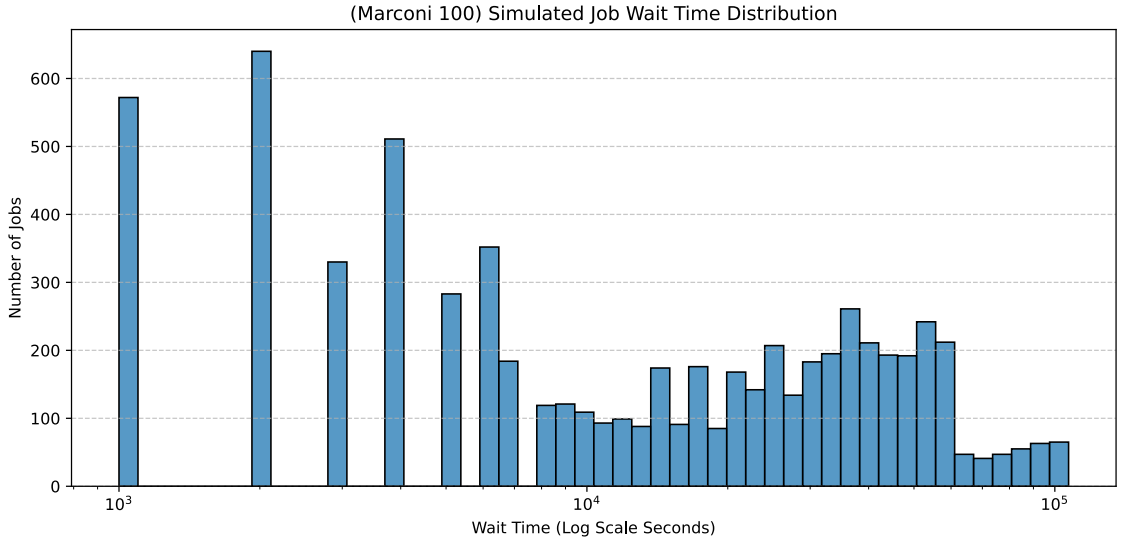


Figure 5.2: Marconi 100 simulated job wait time distribution.

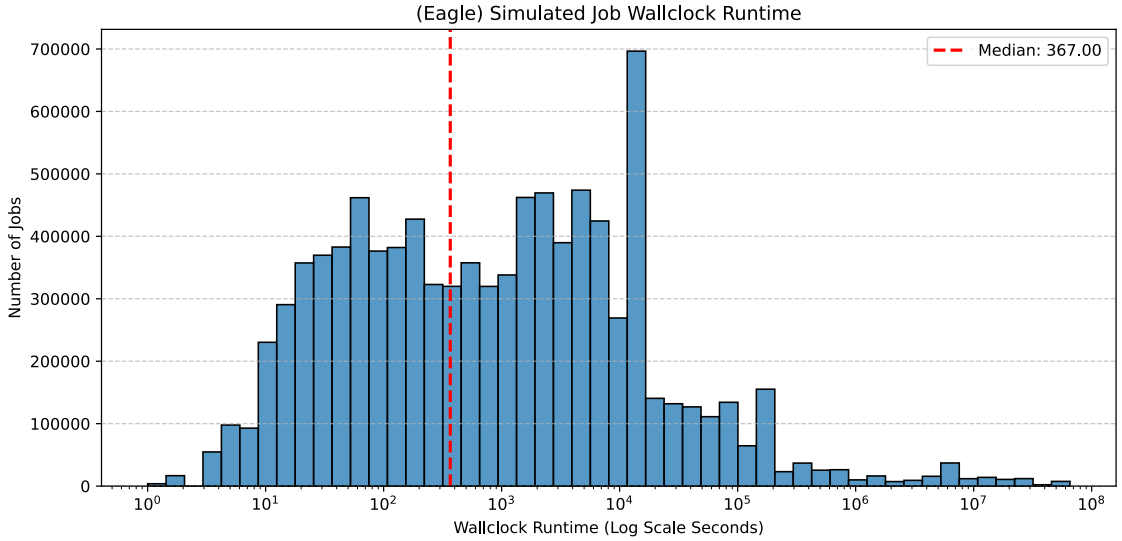


Figure 5.3: Eagle simulated job wallclock runtime distribution.

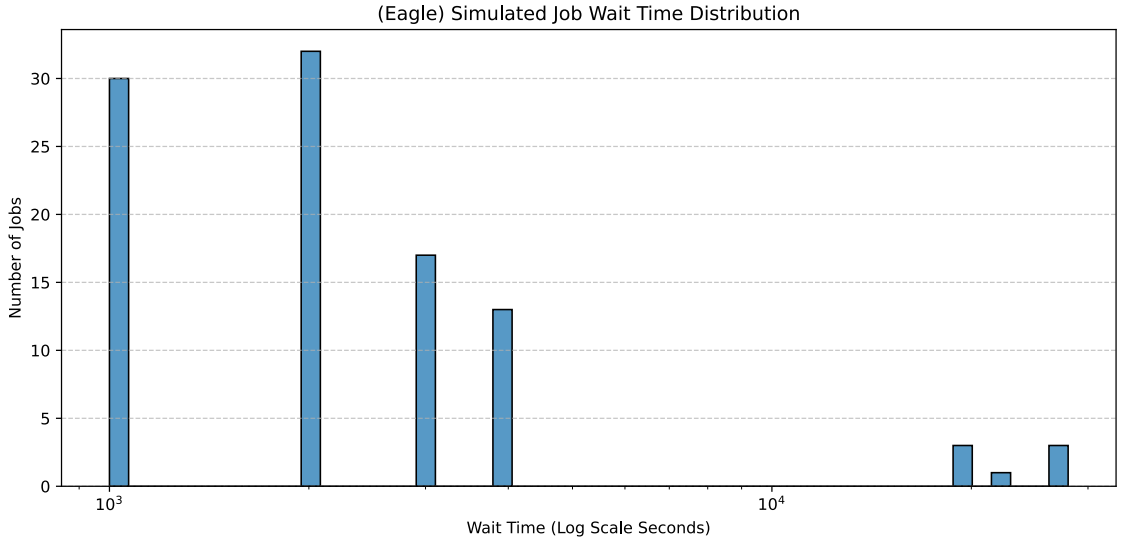


Figure 5.4: Eagle simulated job wait time distribution.

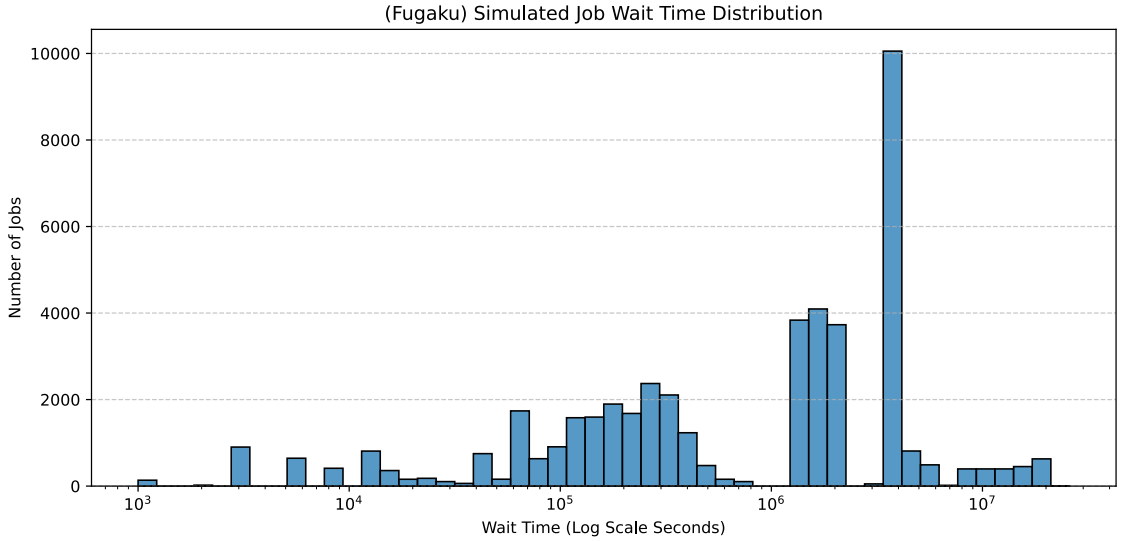


Figure 5.5: Fugaku simulated job wait time distribution.

Chapter 6

Discussion

6.1 ODA of HPC Systems

In our operational data analysis, we systematically examined the available dataset features, organized them along dimensions and developed a structured set of relevant analytical questions, which we then explored in detail. The overall approach is methodical and comprehensive, leaving little room for improvement. Our descriptive findings are thorough, and the resulting prescriptive recommendations are both relevant and well-supported by the analysis.

6.2 Simulation with OpenDC

Simulating our proposed improvements using OpenDC fell short of expectations. The two main limitations encountered were OpenDC's inability to automatically distribute workloads across multiple nodes, and its rigid discrete-event scheduler, which cannot be easily replaced or extended with alternative strategies such as backfilling.

Although we made efforts to extend OpenDC's functionality through direct code modifications, its fragmented codebase made progress slow, ultimately rendering these attempts unsuccessful. As a result, we were unable to simulate changes to the hardware topology or the scheduling logic. In conclusion, our simulation objectives could not be realized and OpenDC proved to be an unsuitable tool for our goals.

Chapter 7

Conclusion

7.1 Cross-System Operational Data Analysis

While many existing studies and tools focus on optimizing individual HPC systems through descriptive analysis, the growing availability of public HPC datasets now enables comparative analyses across multiple, independent systems. Such cross-system insights offer more generalizable conclusions that are not tied to a single infrastructure. This thesis represents a first step in that direction, providing both descriptive and prescriptive operational data analysis (ODA) across three distinct, job-centric HPC datasets.

We performed a thorough examination of each dataset and conducted single and comparative descriptive analysis across them. This enabled us to identify recurring patterns and anomalies. Based on these findings, we formulated a set of actionable recommendations for improving HPC system performance in general.

One particularly promising improvement is placing greater emphasis on efficiently handling short, sequential jobs within HPC environments traditionally optimized for large-scale parallel workloads. This approach not only enables faster processing of smaller jobs but also reduces scheduling interference with larger jobs. We predict an increased overall throughput and reduced power consumption.

7.2 Outlook

The promising improvements identified in this thesis now warrant validation through simulation. While our attempts using OpenDC were limited by its lack of support for multi-node scheduling and flexible scheduling strategies, future work could either extend OpenDC's capabilities or transition to an alternative such as Batsim. As the author, I propose a dual approach: leveraging OpenDC's great capabilities of modeling power consumption and leveraging Batsim for exploring scheduling strategies due to its flexibility in scheduler design. Using both simulators could provide a comprehensive simulation framework, enabling evaluation of both energy efficiency and scheduling performance.

Bibliography

[1] Francesco Antici, Andrea Bartolini, Jens Domke, Zeynep Kiziltan, Keiji Yamamoto, et al. F-data: A fugaku workload dataset for job-centric predictive modelling in hpc systems. 2024.

[2] Francesco Antici, Mohsen Seyedkazemi Ardebili, Andrea Bartolini, and Zeynep Kiziltan. Pm100: A job power consumption dataset of a large-scale production hpc system. In *Proceedings of the SC'23 Workshops of the International Conference on High Performance Computing, Network, Storage, and Analysis*, pages 1812–1819, 2023.

[3] Norman Bourassa, Walker Johnson, Jeff Broughton, Deirdre McShane Carter, Sadie Joy, Raphael Vitti, and Peter Seto. Operational data analytics: Optimizing the national energy research scientific computing center cooling systems. In *Workshop Proceedings of the 48th International Conference on Parallel Processing*, pages 1–7, 2019.

[4] Henri Casanova, Arnaud Giersch, Arnaud Legrand, Martin Quinson, and Frédéric Suter. Lowering entry barriers to developing custom simulators of distributed applications and platforms with simgrid. *Parallel Computing*, page 103125, 2025.

[5] Nicolas Chan. A resource utilization analytics platform using grafana and telegraf for the savio supercluster. In *Practice and Experience in Advanced Research Computing 2019: Rise of the Machines (learning)*, pages 1–6. 2019.

[6] Xiaoyu Chu, Daniel Hofstätter, Shashikant Ilager, Sacheendra Talluri, Duncan Kampert, Damian Podareanu, Dmitry Duplyakin, Ivona Brandic, and Alexandru Iosup. Generic and ml workloads in an hpc datacenter: Node energy, job failures, and node-job analysis. In *2024 IEEE 30th International Conference on Parallel and Distributed Systems (ICPADS)*, pages 710–719. IEEE, 2024.

[7] Dmitry Duplyakin and Kevin Menear. Nrel eagle supercomputer jobs. Open Energy Data Initiative (OEDI), National Renewable Energy Laboratory, https://data.openei.org/submissions/5860, 2023. Accessed: 2025-06-11.

[8] Pierre-François Dutot, Michael Mercier, Millian Poquet, and Olivier Richard. Batsim: a realistic language-independent resources and jobs management systems simulator. In *Workshop on Job Scheduling Strategies for Parallel Processing*, pages 178–197. Springer, 2015.

[9] Joseph Emeras, Sébastien Varrette, Mateusz Guzek, and Pascal Bouvry. Evalix: classification and prediction of job resource consumption on hpc platforms. In *Workshop on Job Scheduling Strategies for Parallel Processing*, pages 102–122. Springer, 2015.

[10] Fujitsu. Fugaku specifications : Fujitsu global. https://www.fujitsu.com/global/about/innovation/fugaku/specifications/, 2025. Accessed on 2025-06-11.

[11] Cristian Galleguillos, Zeynep Kiziltan, and Alessio Netti. Accasim: an hpc simulator for workload management. In *Latin American High Performance Computing Conference*, pages 169–184. Springer, 2017.

[12] Dalibor Klusáček, Mehmet Soysal, and Frédéric Suter. Alea–complex job scheduling simulator. In *International Conference on Parallel Processing and Applied Mathematics*, pages 217–229. Springer, 2019.

[13] Katerina Lepenioti, Alexandros Bousdekis, Dimitris Apostolou, and Gregoris Mentzas. Prescriptive analytics: Literature review and research challenges. *International Journal of Information Management*, 50:57–70, 2020.

[14] Jiangtian Li, Xiaosong Ma, Karan Singh, Martin Schulz, Bronis R de Supinski, and Sally A McKee. Machine learning based online performance prediction for runtime parallelization and task scheduling. In *2009 IEEE international symposium on performance analysis of systems and software*, pages 89–100. IEEE, 2009.

[15] Mervi Mantsinen. Welcome to marconi100: a new tool for fusion research. https://fusion.bsc.es/index.php/2020/06/11/welcome-to-marconi100-a-new-tool-for-fusion-research/, June 2020. Accessed on 2025-06-12.

[16] Fabian Mastenbroek, Georgios Andreadis, Soufiane Jounaid, Wenchen Lai, Jacob Burley, Jaro Bosch, Erwin Van Eyk, Laurens Versluis, Vincent Van Beek, and Alexandru Iosup. Opendc 2.0: Convenient modeling and simulation of emerging technologies in cloud datacenters. In *2021 IEEE/ACM 21st International Symposium on Cluster, Cloud and Internet Computing (CCGrid)*, pages 455–464. IEEE, 2021.

[17] Kevin Menear, Ambarish Nag, Jordan Perr-Sauer, Monte Lunacek, Kristi Potter, and Dmitry Duplyakin. Mastering hpc runtime prediction: From observing patterns to a methodological approach. In *Practice and Experience in Advanced Research Computing 2023: Computing for the Common Good*, pages 75–85. 2023.

[18] Kris Munch and Aaron Andersen. The eagle supercomputer: System and user environment. Technical Report NREL/TP-2C00-86370, National Renewable Energy Laboratory (NREL), September 2023.

[19] Manzur Murshed, Rajkumar Buyya, and David Abramson. Gridsim: A toolkit for the modeling and simulation of global grids. *Monash University Journal*, 1, 2001.

[20] Alessio Netti, Woong Shin, Michael Ott, Torsten Wilde, and Natalie Bates. A conceptual framework for hpc operational data analytics. In *2021 IEEE International Conference on Cluster Computing (CLUSTER)*, pages 596–603. IEEE, 2021.

[21] Michael Ott, Woong Shin, Norman Bourassa, Torsten Wilde, Stefan Ceballos, Melissa Romanus, and Natalie Bates. Global experiences with hpc operational data measurement, collection and analysis. In *2020 IEEE International Conference on Cluster Computing (CLUSTER)*, pages 499–508. IEEE, 2020.

[22] The Apache Software Foundation. Apache parquet documentation: Overview. https://parquet.apache.org/docs/. Accessed on 2024-06-10.

[23] Jia Wei, Mo Chen, Longxiang Wang, Pei Ren, Yujia Lei, Yuqi Qu, Qiyu Jiang, Xiaoshe Dong, Weiguo Wu, Qiang Wang, et al. Status, challenges and trends of data-intensive supercomputing. *CCF Transactions on High Performance Computing*, 4(2):211–230, 2022.

[24] Torsten Wilde, Axel Auweter, and Hayk Shoukourian. The 4 pillar framework for energy efficient hpc data centers. *Computer Science-Research and Development*, 29:241–251, 2014.

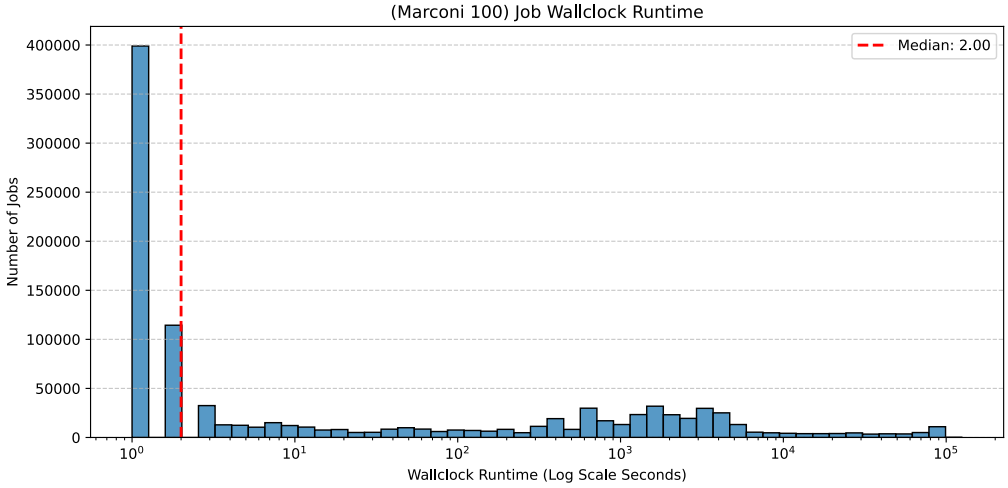
Appendix A

Figures

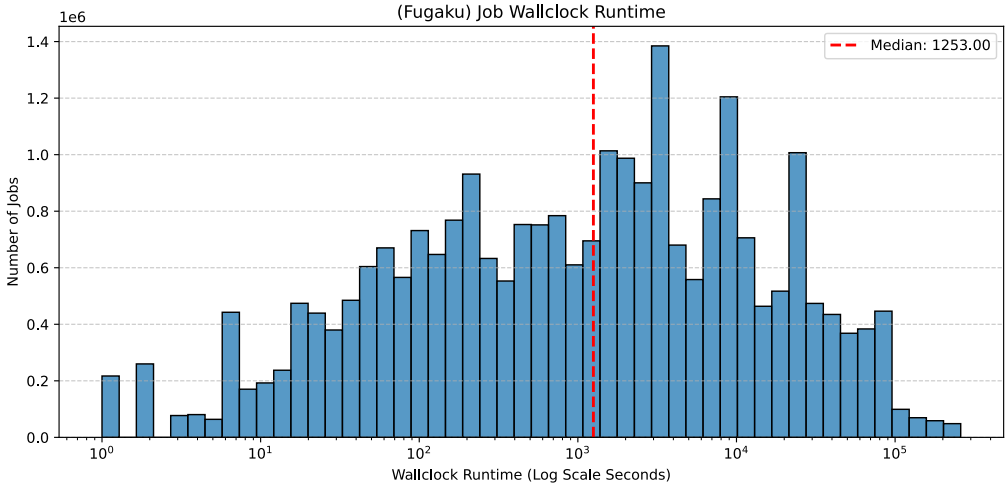
A.0.1 Cross-System Analysis

Figure A.1: Histograms of wall-clock run time across systems.

(a) Marconi



(b) Fugaku



(c) Eagle

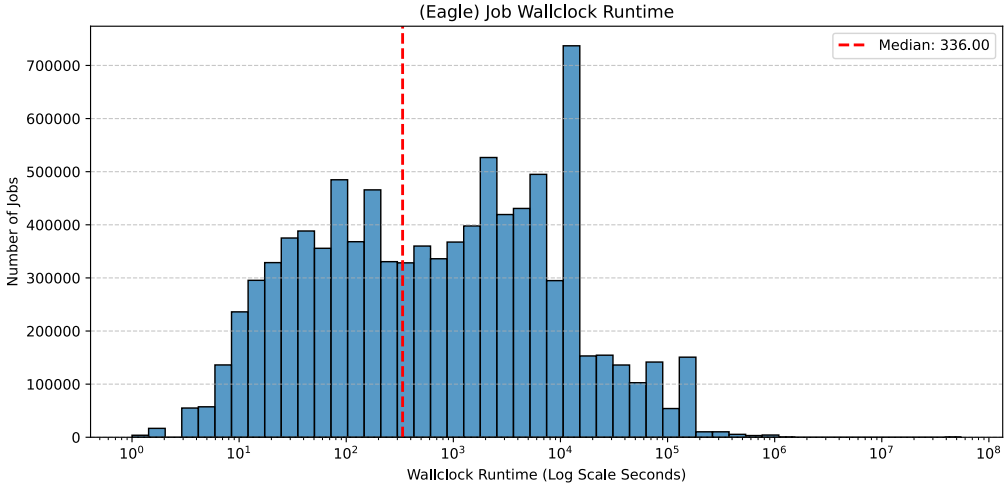
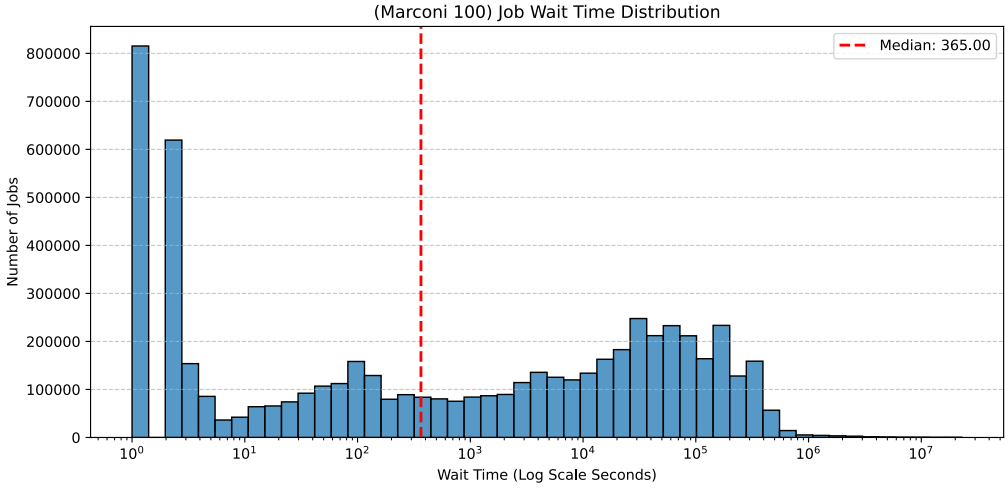
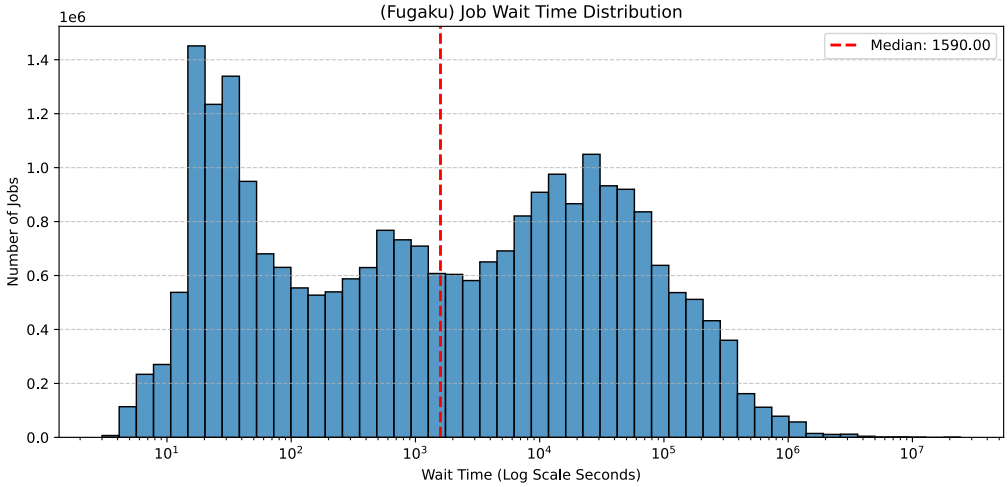


Figure A.2: Histograms of job wait time across systems.

(a) Marconi



(b) Fugaku



(c) Eagle

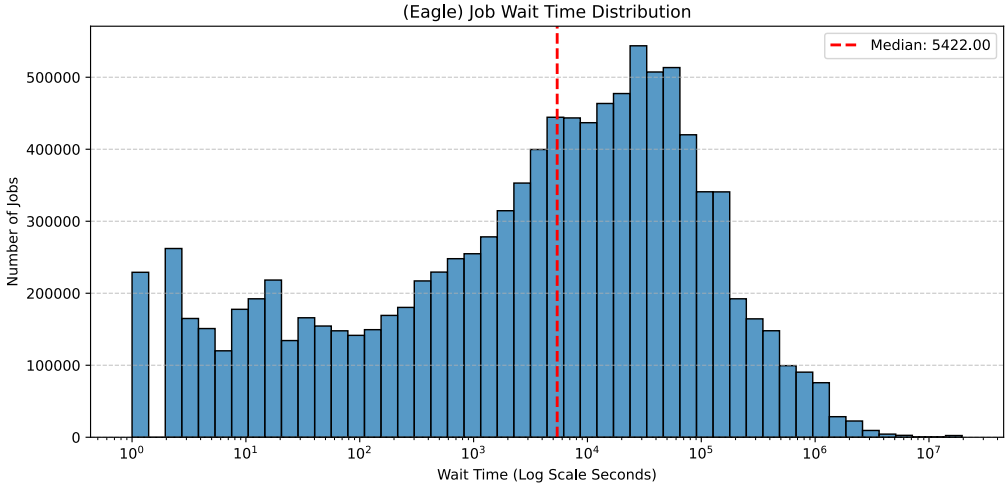
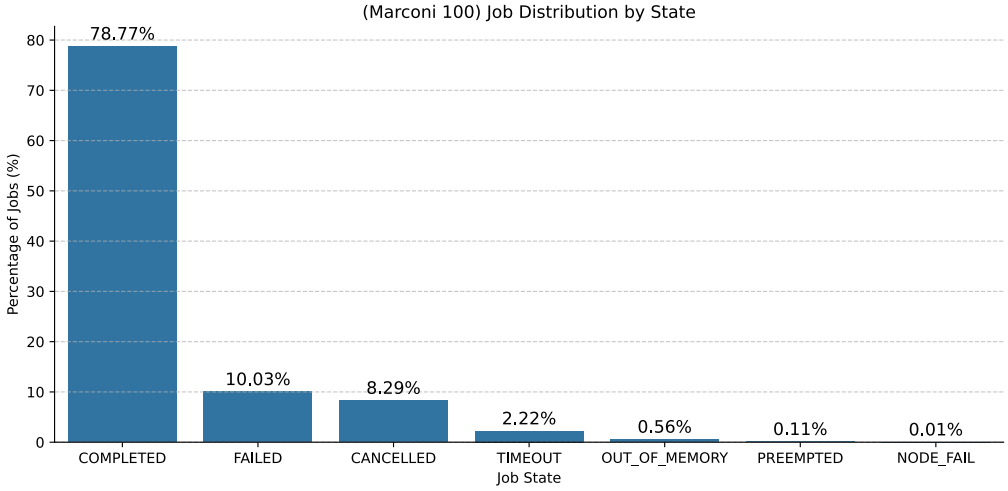
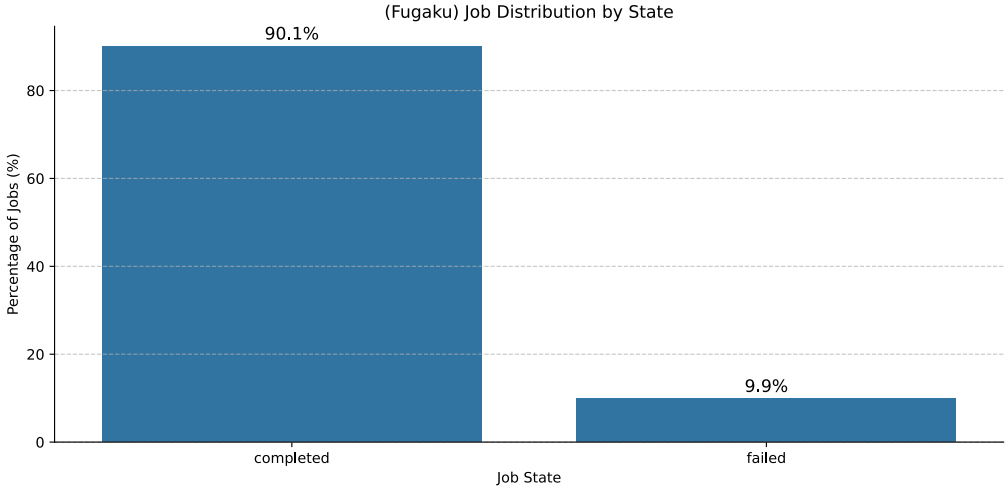


Figure A.3: Histograms of job exit states across systems.

(a) Marconi



(b) Fugaku



(c) Eagle

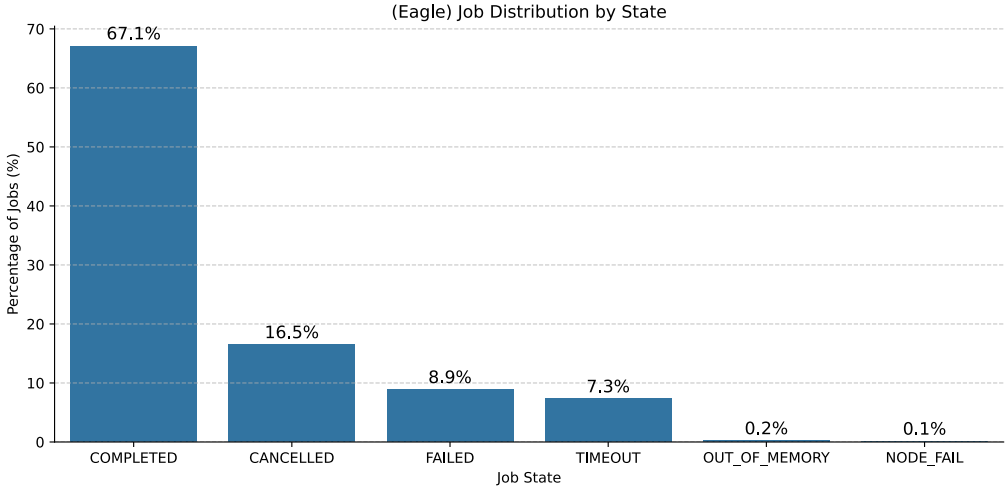


Figure A.4: Analysis of job arrivals (Marconi).

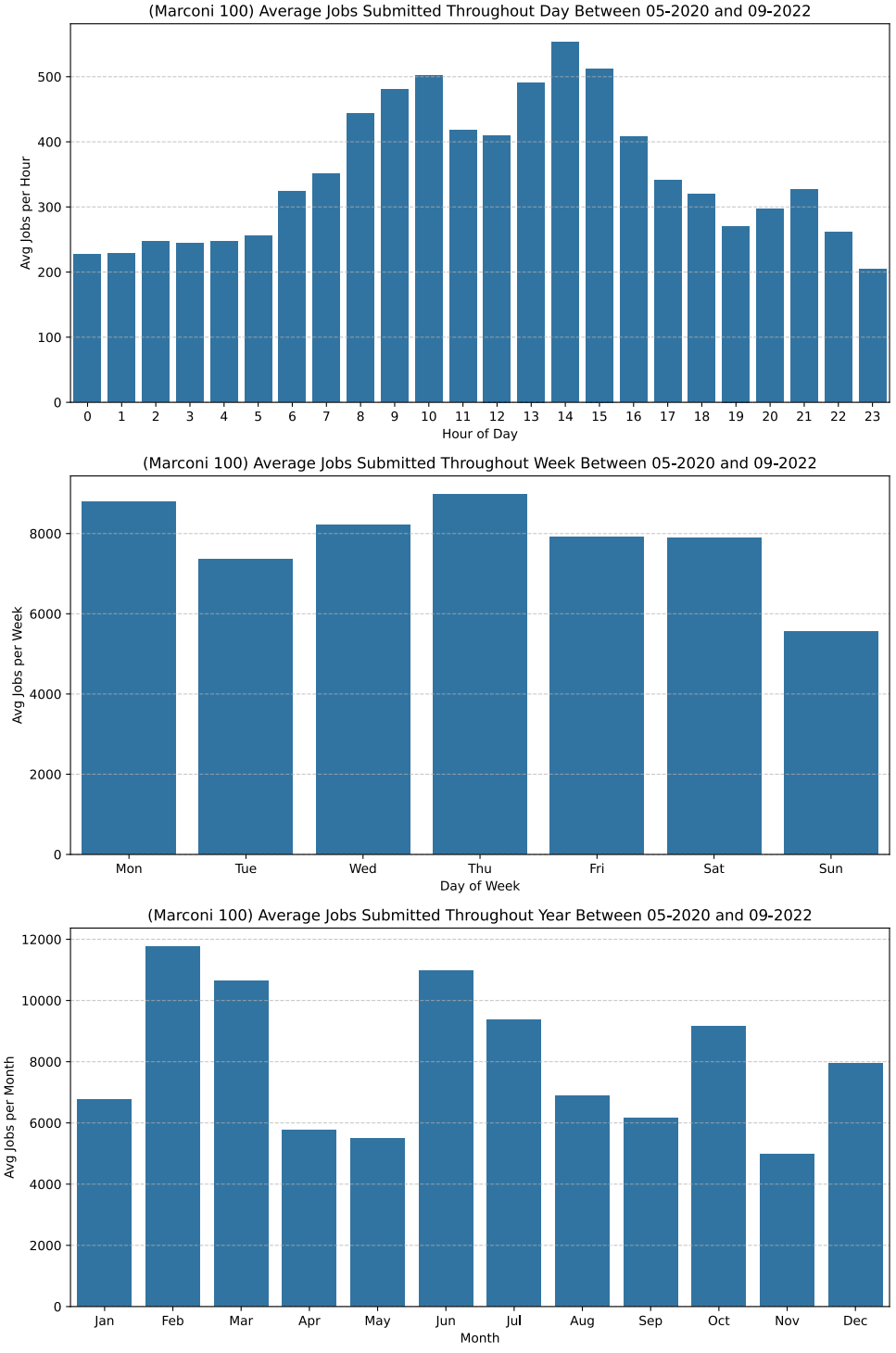
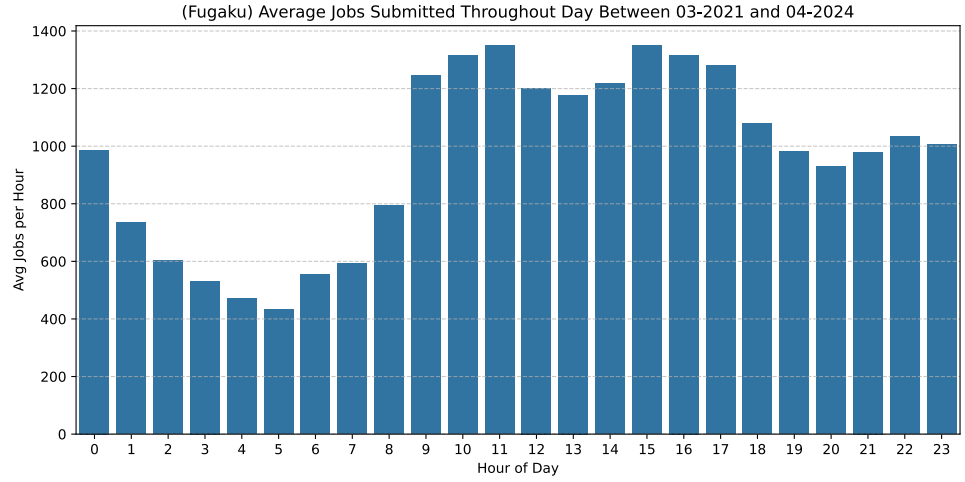
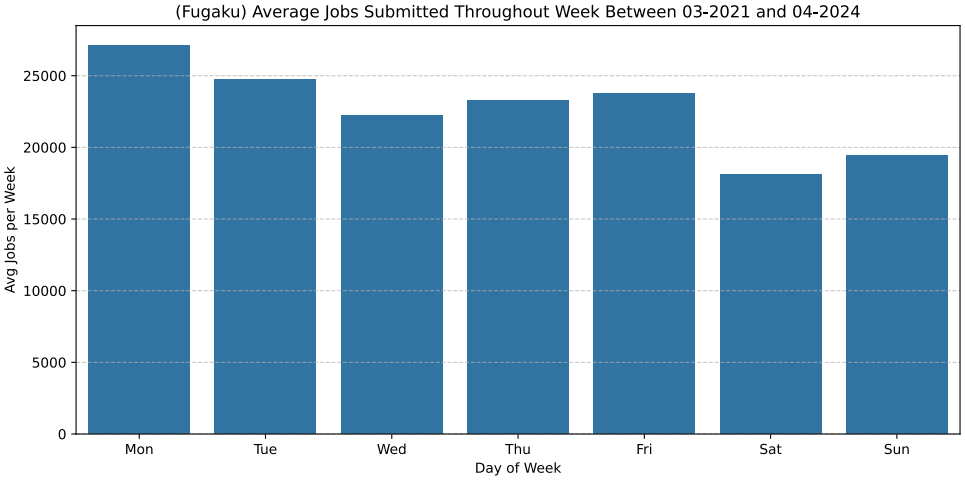


Figure A.5: Analysis of job arrivals (Fugaku).





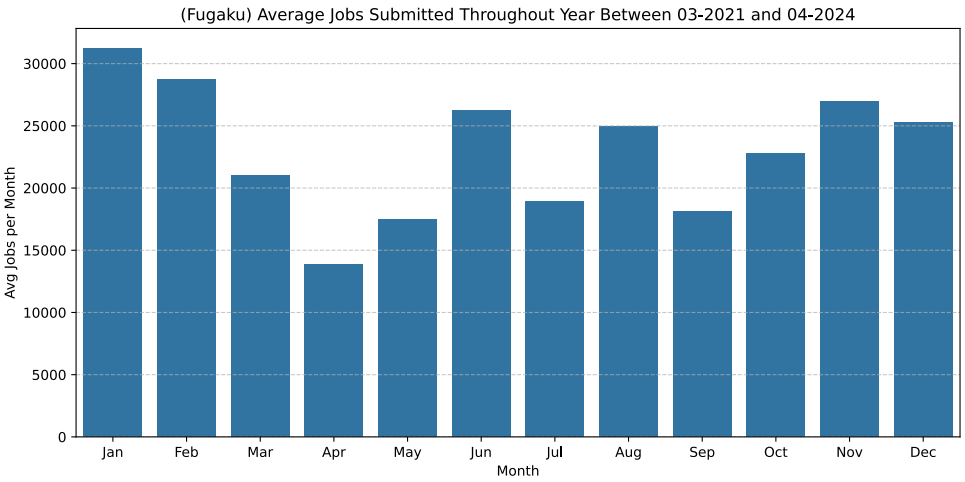
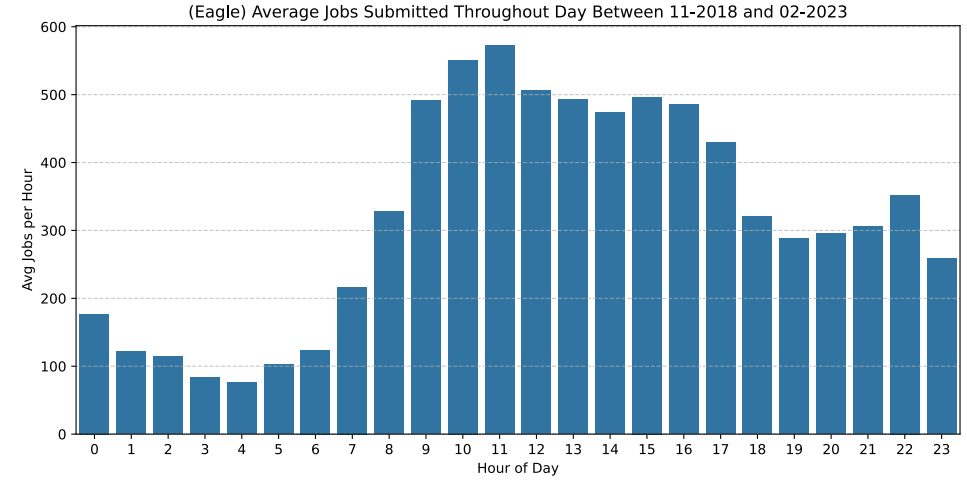
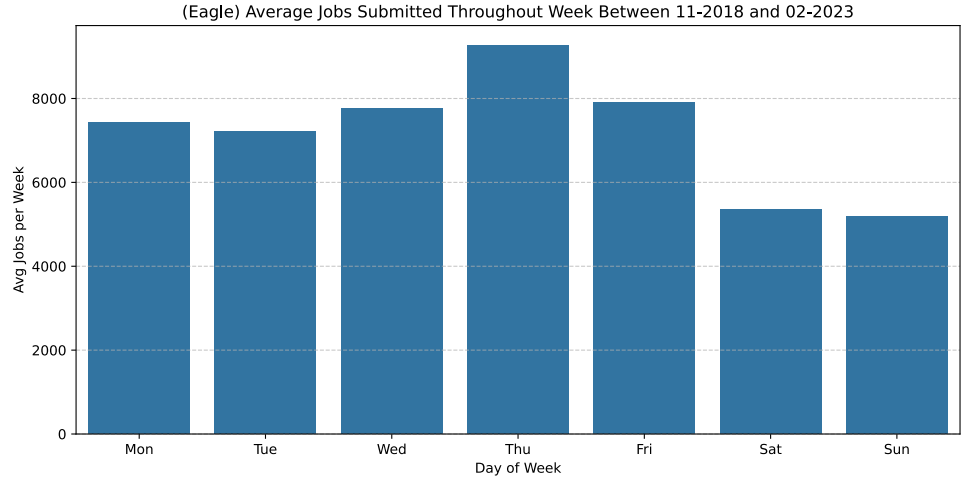


Figure A.6: Analysis of job arrivals (Eagle).





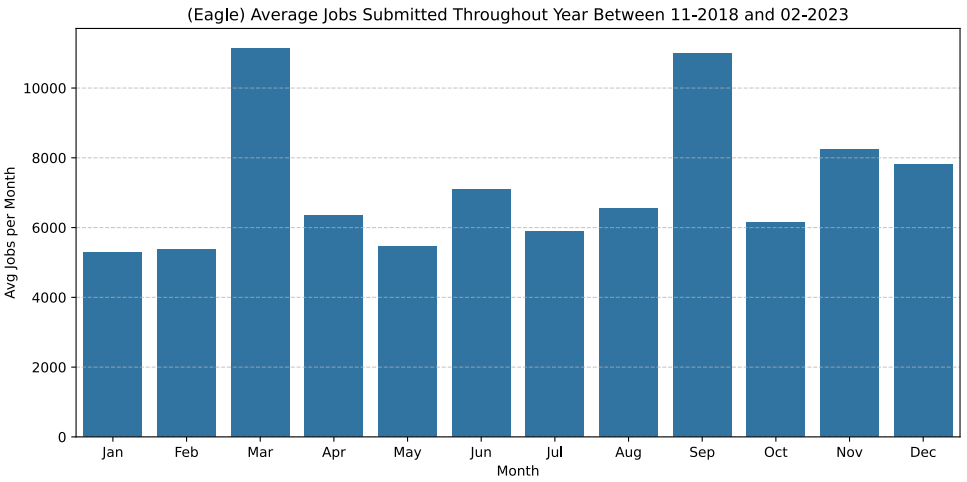


Figure A.7: Analysis of wait times (Marconi).

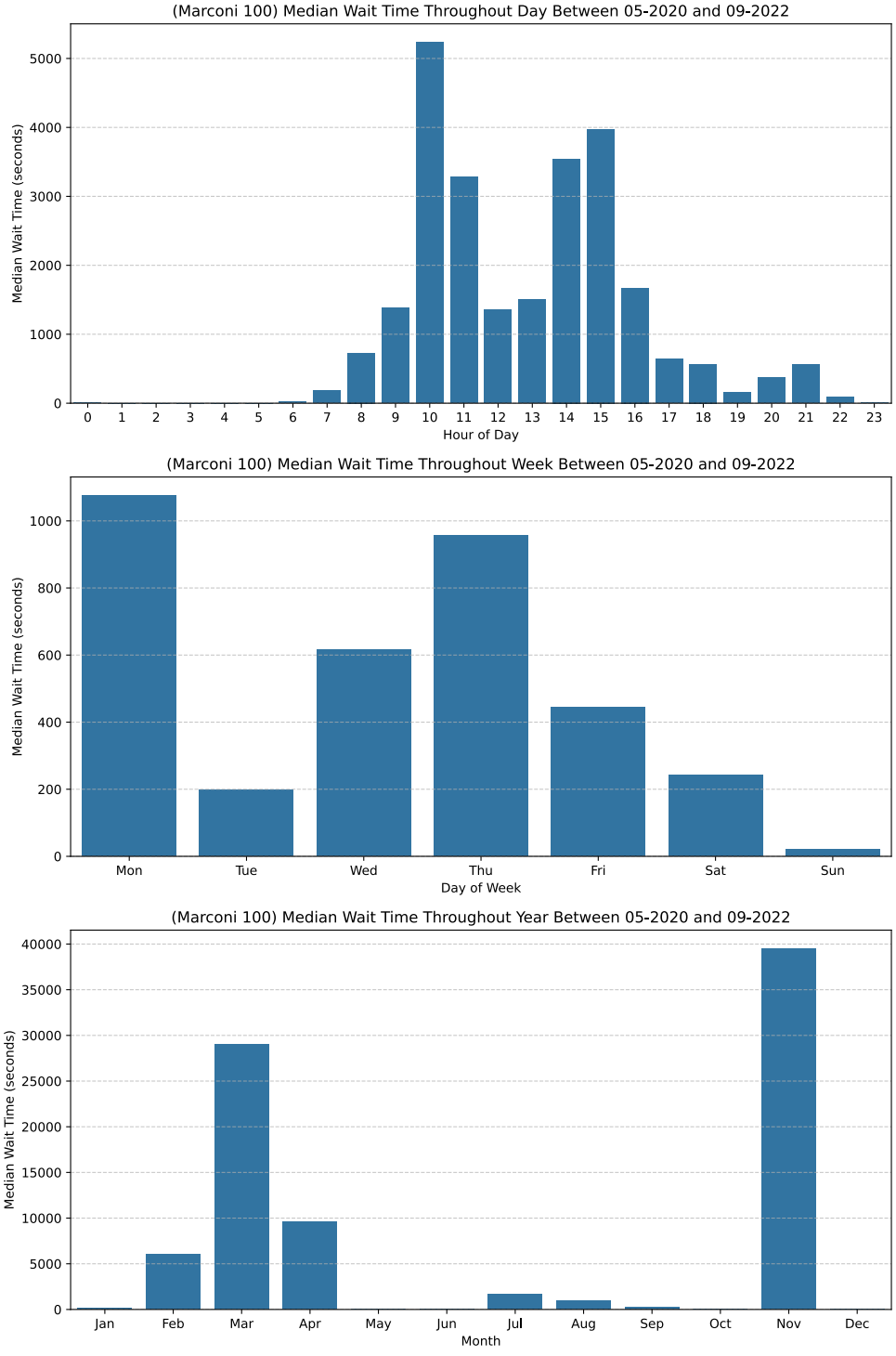


Figure A.8: Analysis of wait times (Fugaku).

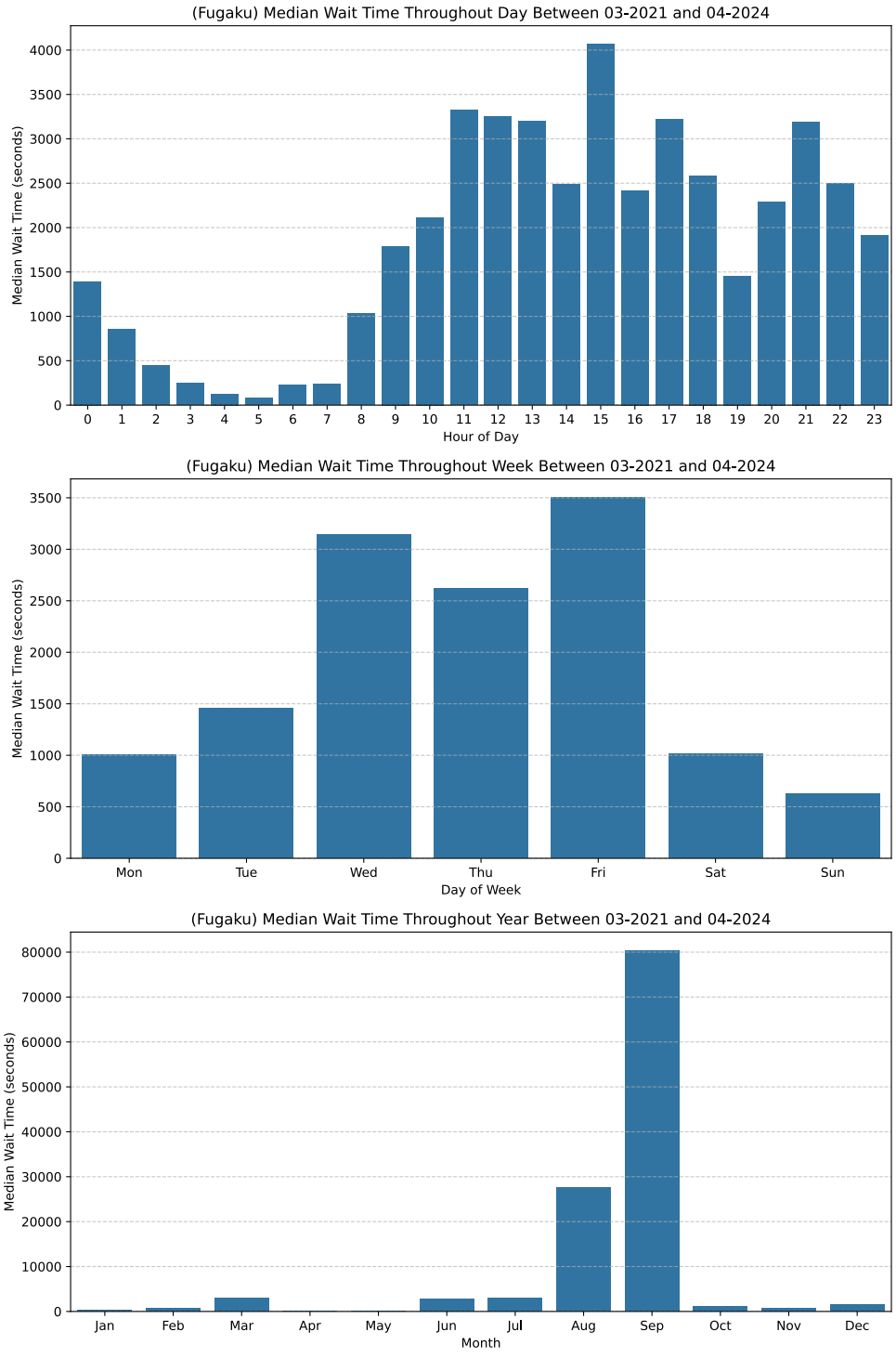


Figure A.9: Analysis of wait times (Eagle).

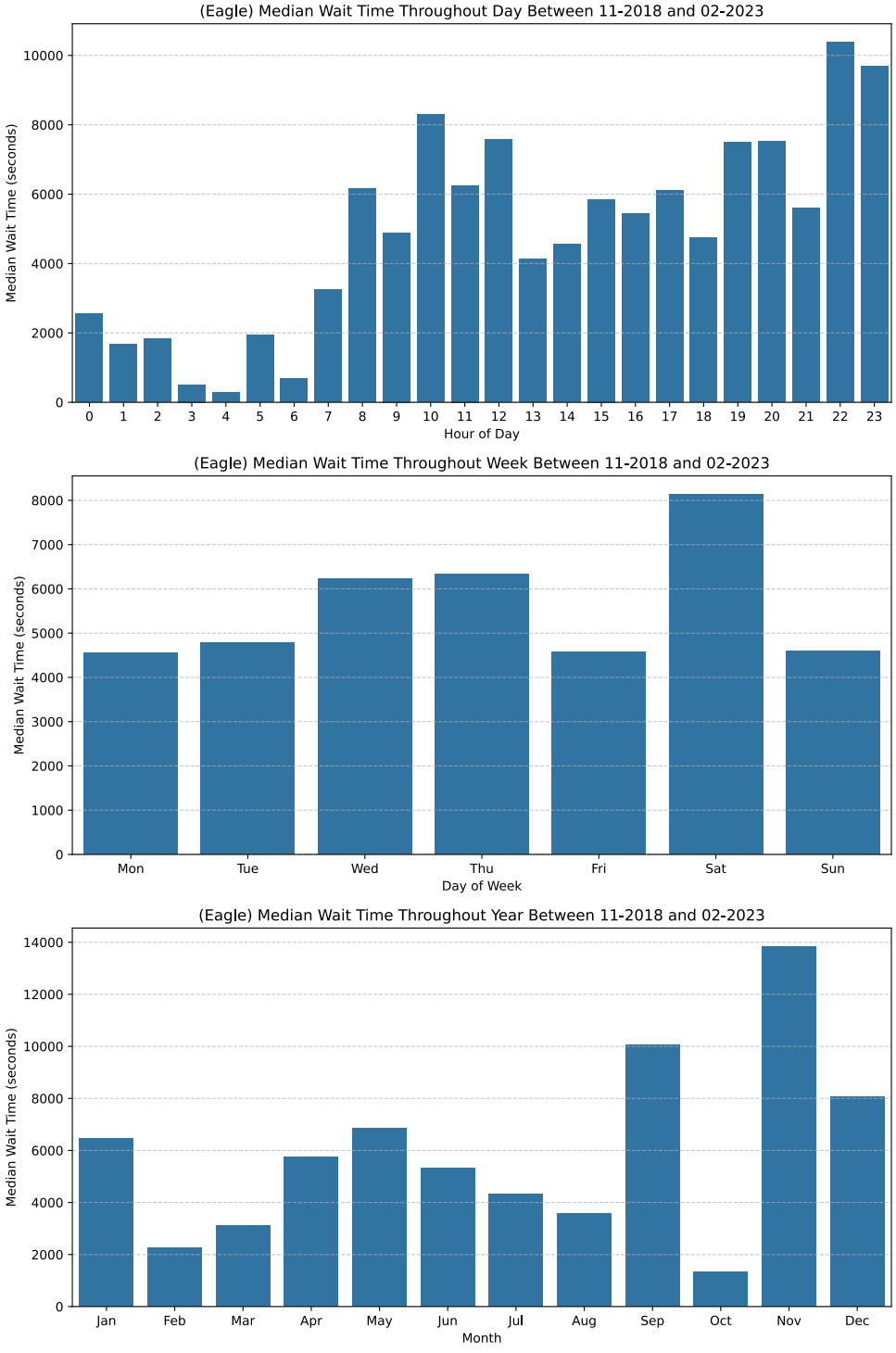


Figure A.10: Histograms of requested nodes per job (Marconi).

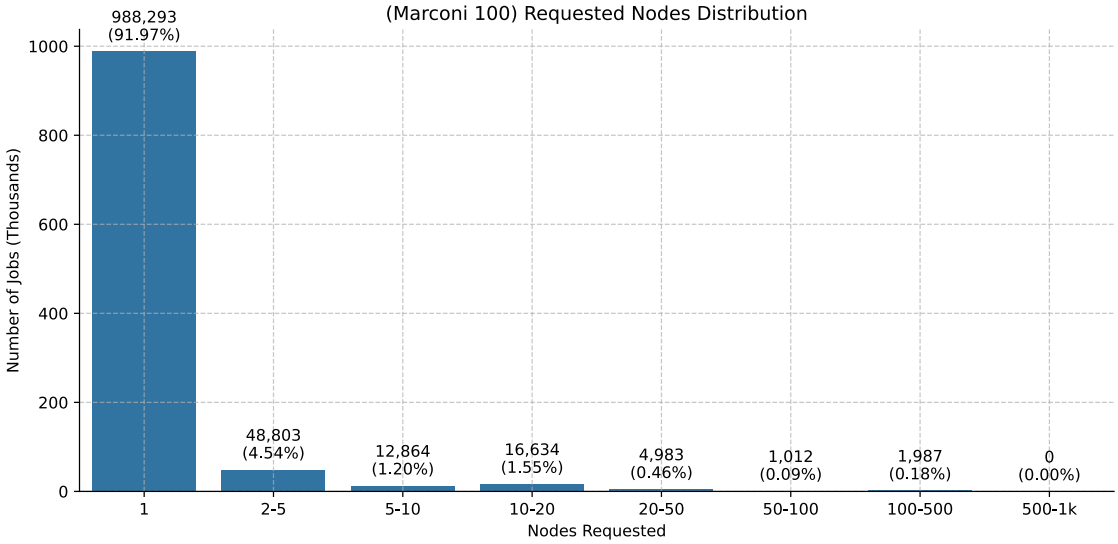


Figure A.11: Histograms of requested nodes per job (Fugaku).

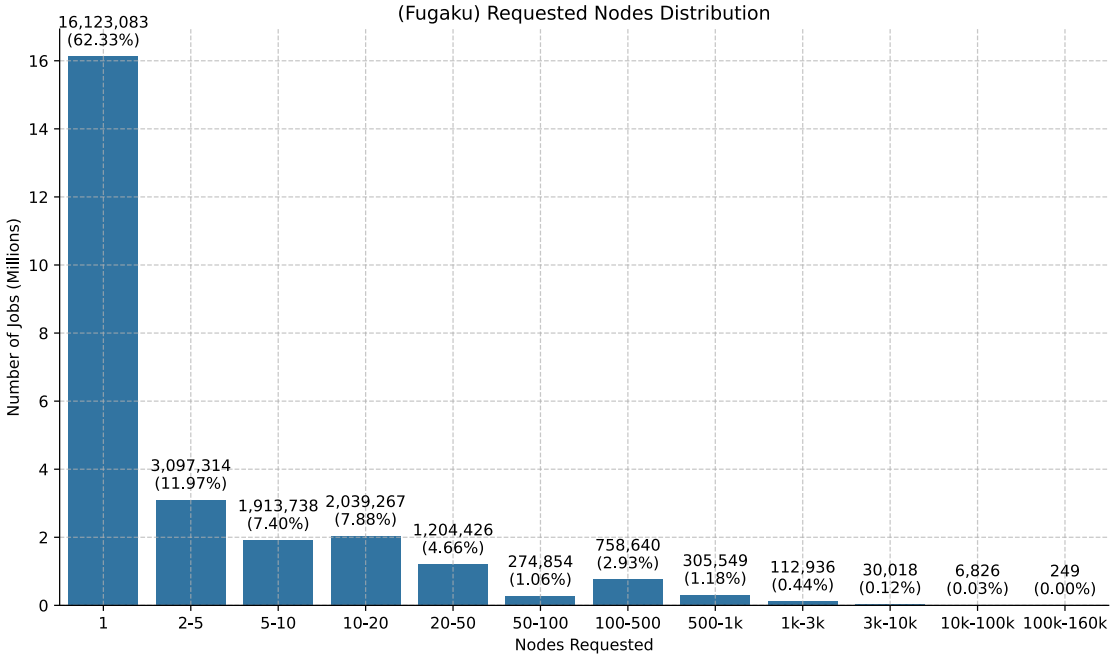


Figure A.12: Histograms of requested nodes per job (Eagle).

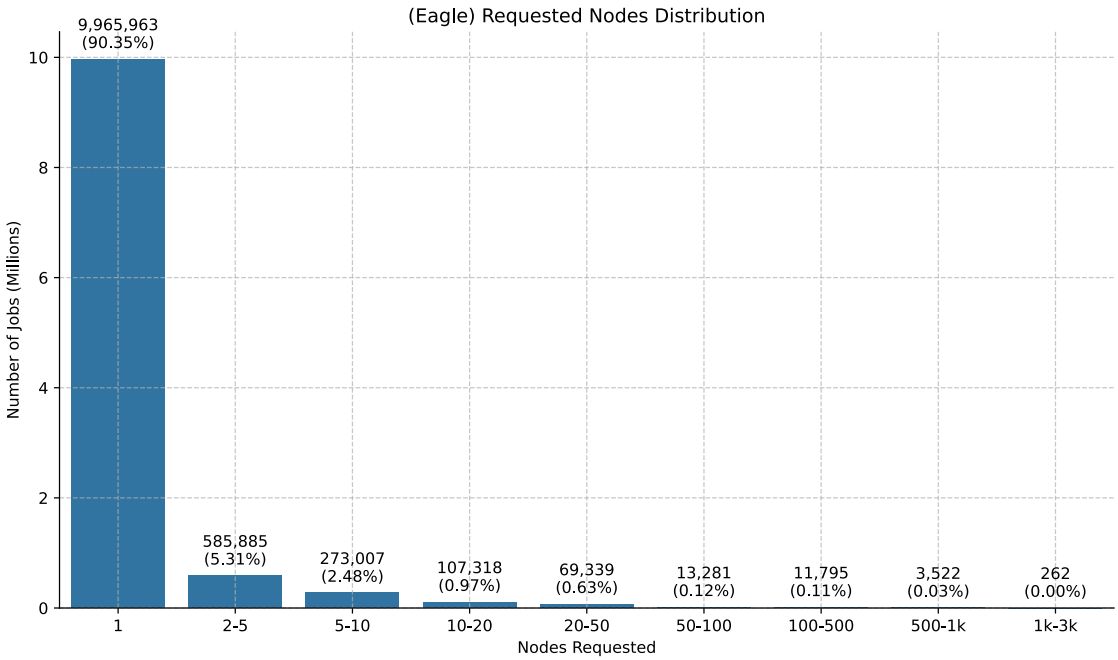
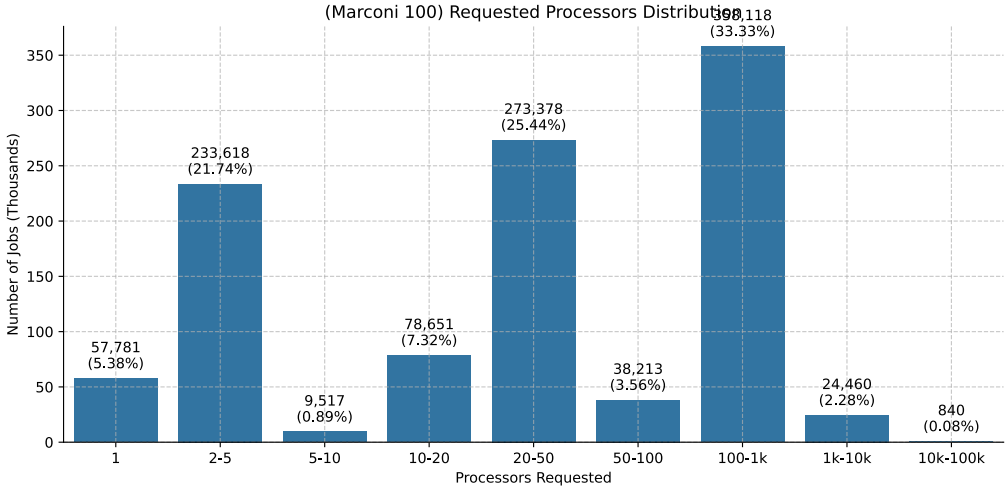
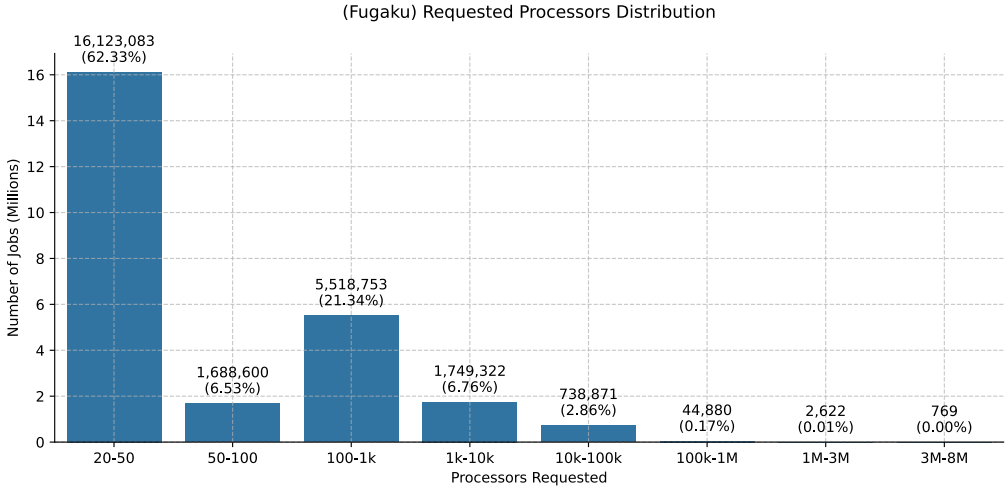


Figure A.13: Histograms of requested processors per job across systems.

(a) Marconi



(b) Fugaku



(c) Eagle

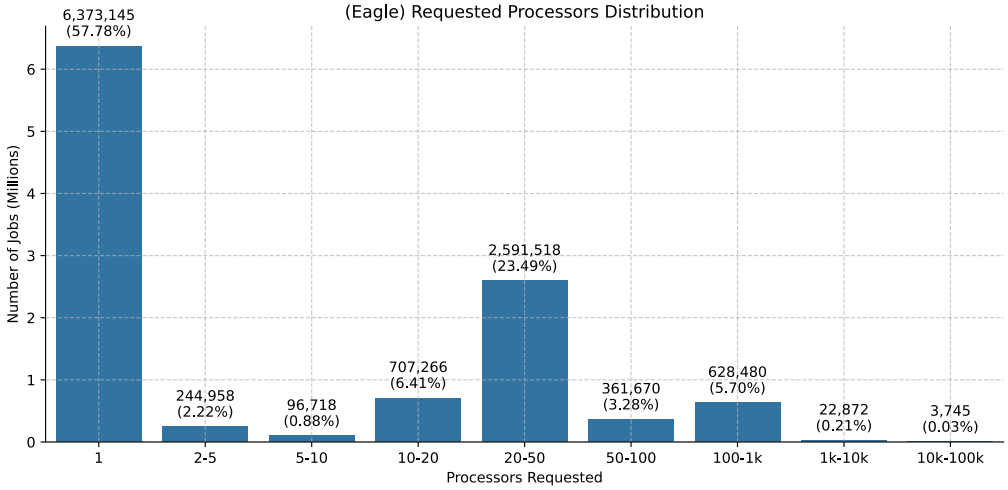
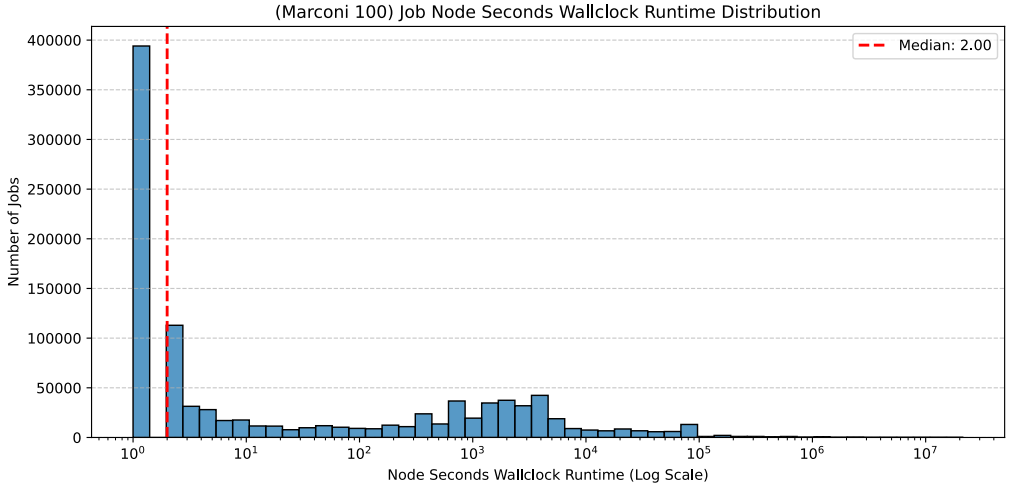
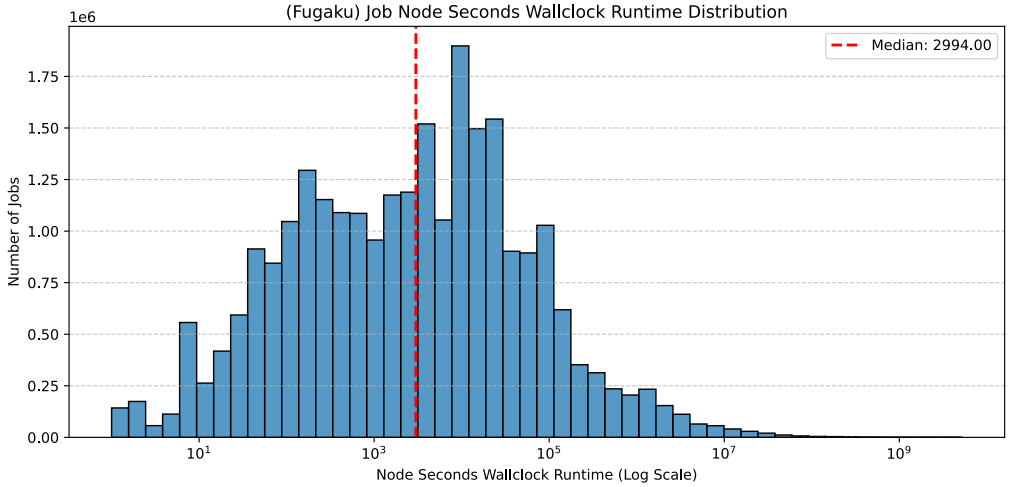


Figure A.14: Histograms of job node seconds across systems.

(a) Marconi



(b) Fugaku



(c) Eagle

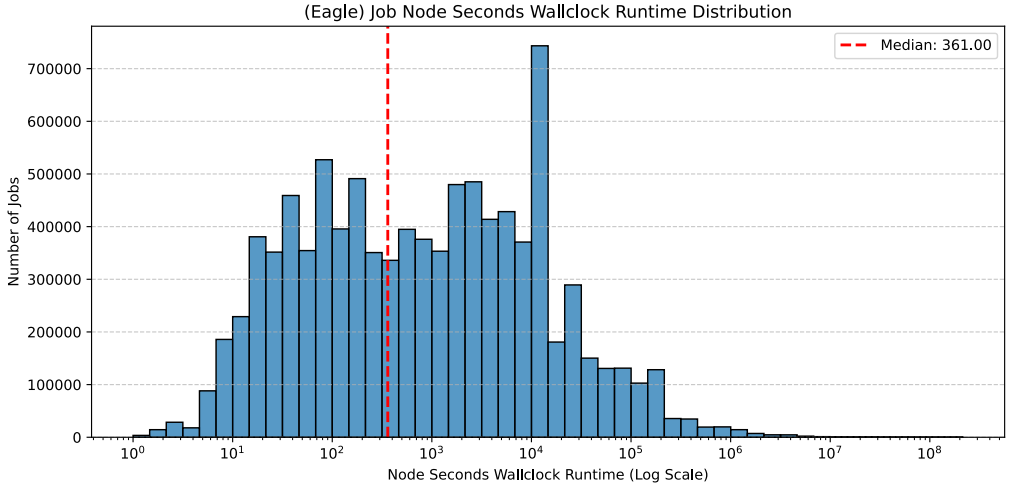
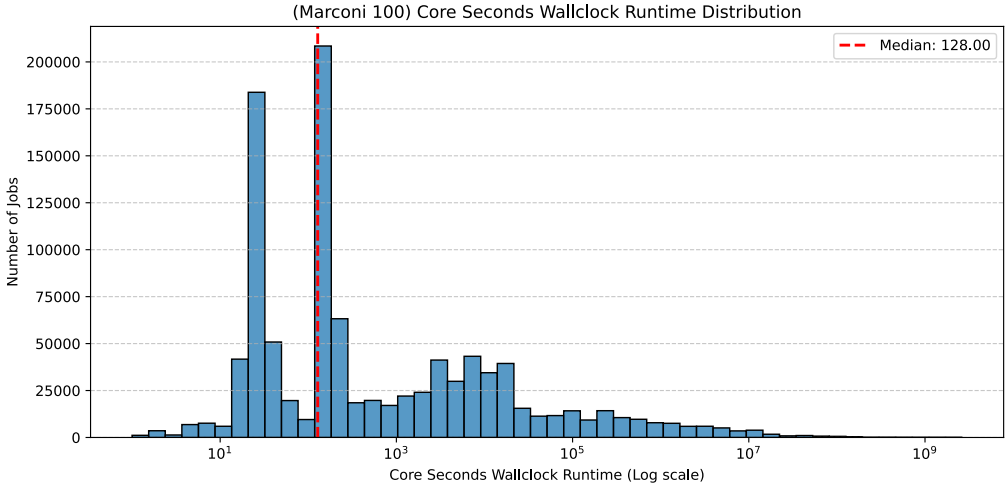
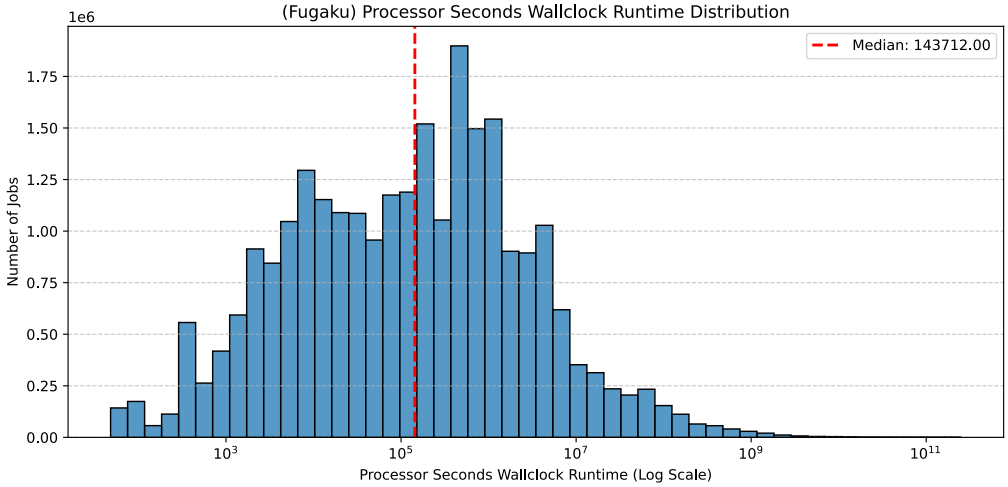


Figure A.15: Histograms of job processor seconds across systems.

(a) Marconi



(b) Fugaku



(c) Eagle

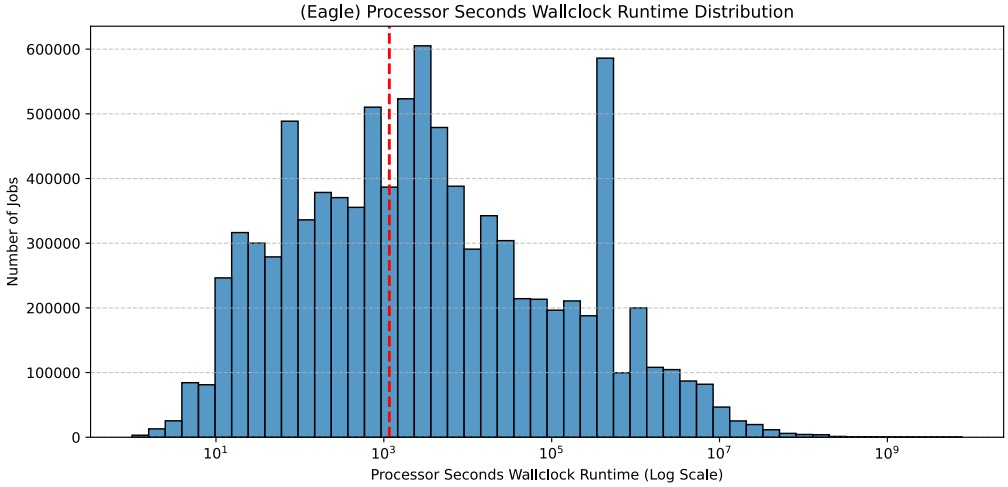
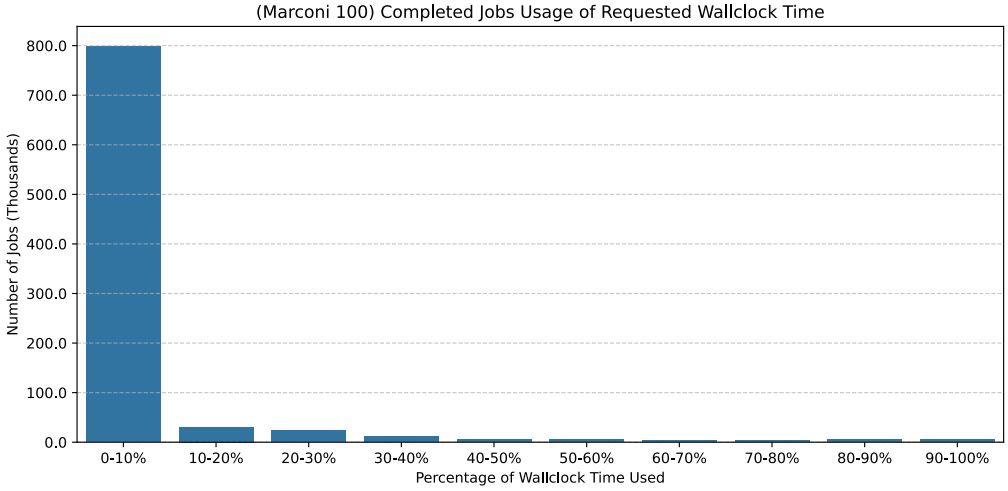
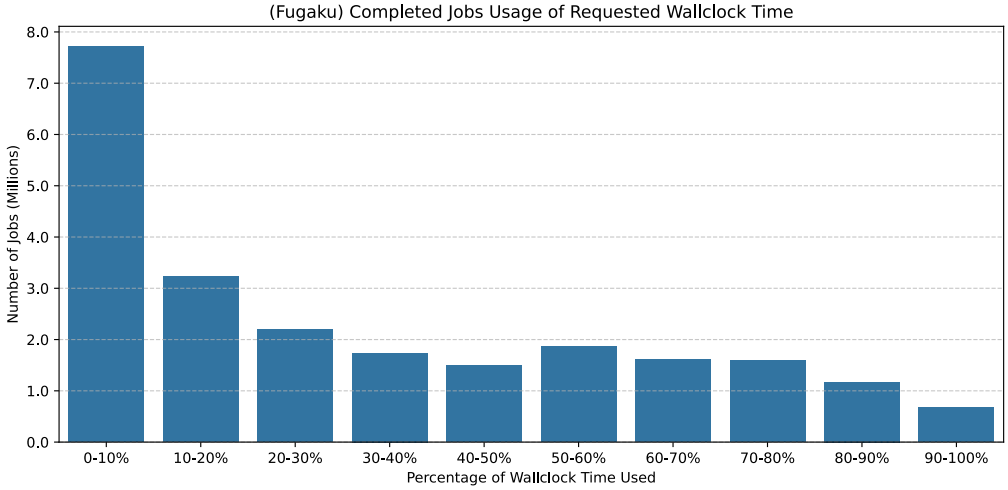


Figure A.16: Analysis of time limit usage for completed jobs across systems.

(a) Marconi



(b) Fugaku



(c) Eagle

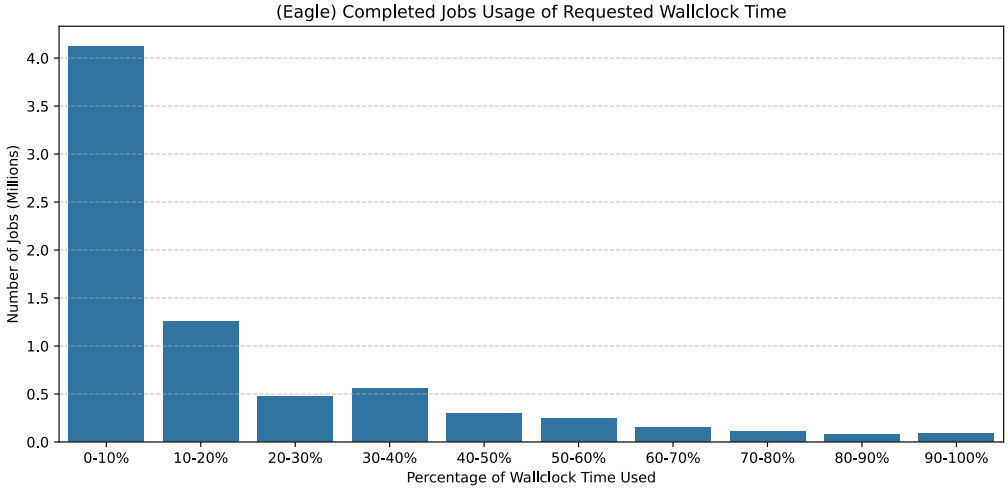
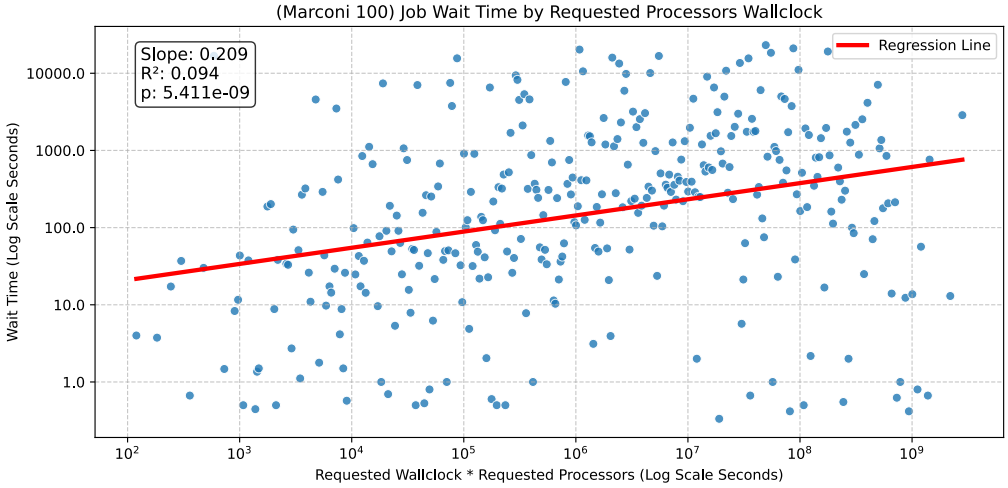
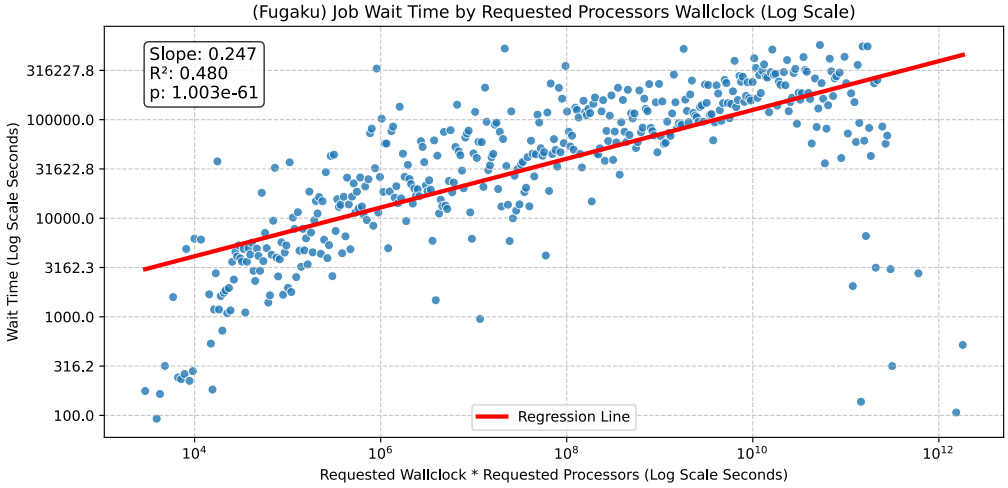


Figure A.17: Comparison of wait time and requested processor wall-clock across systems.

(a) Marconi



(b) Fugaku



(c) Eagle

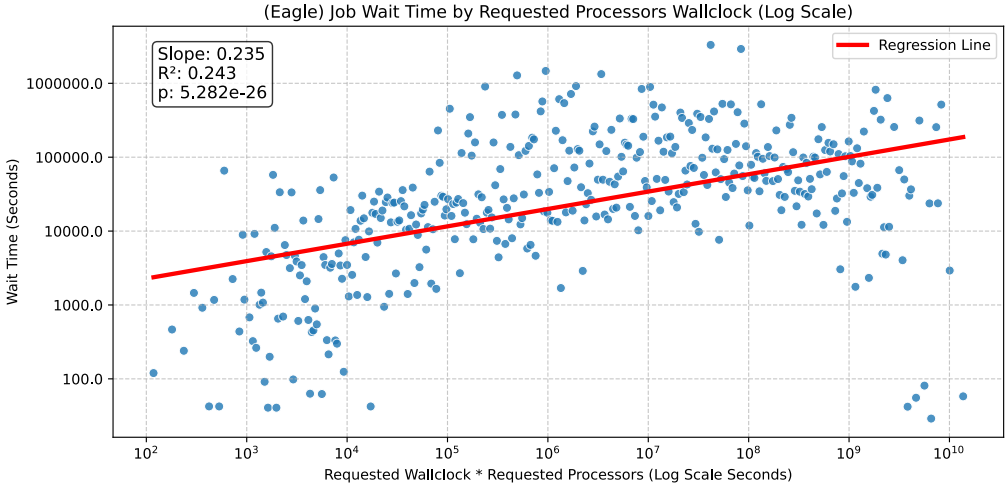
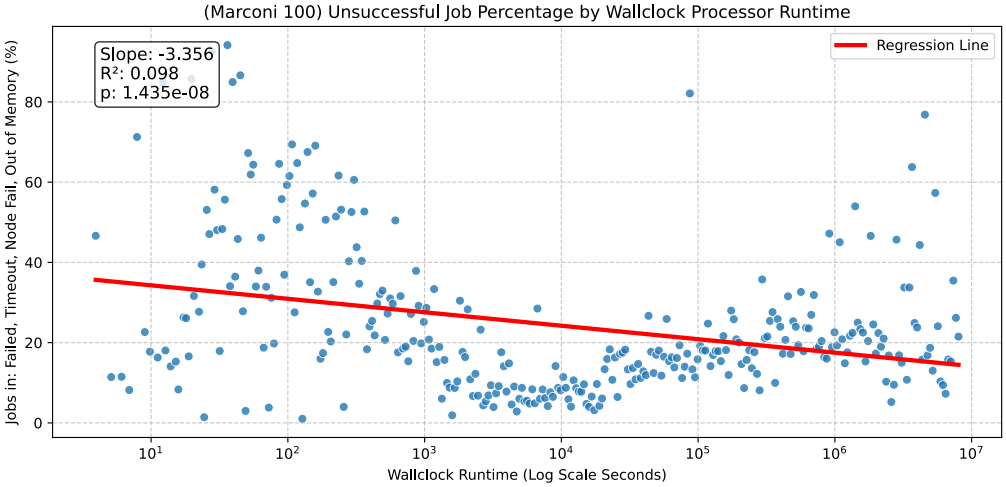
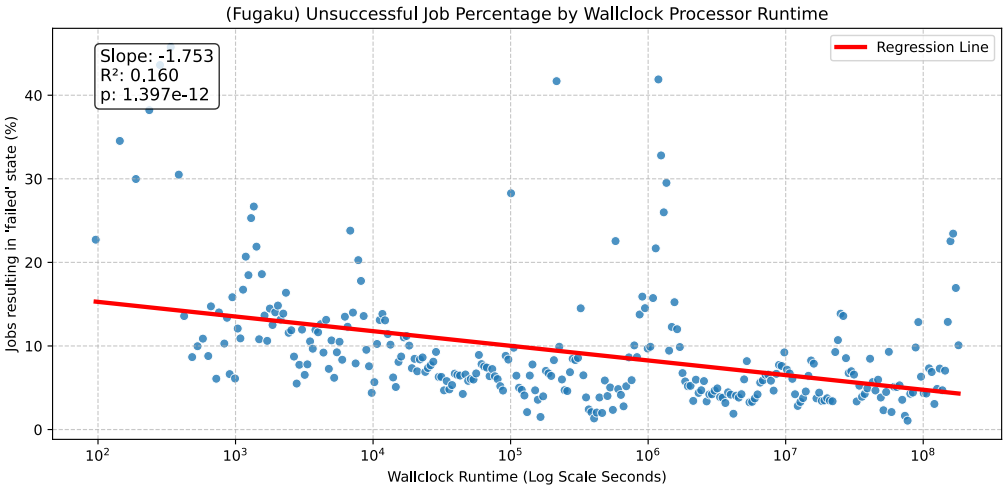


Figure A.18: Comparison of unsuccessful job rate and processor wall-clock runtime across systems.

(a) Marconi



(b) Fugaku



(c) Eagle

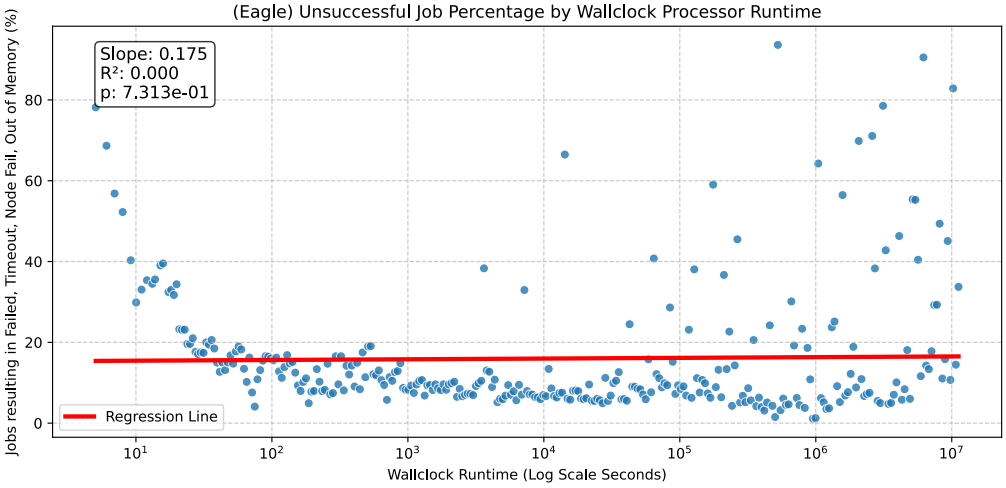
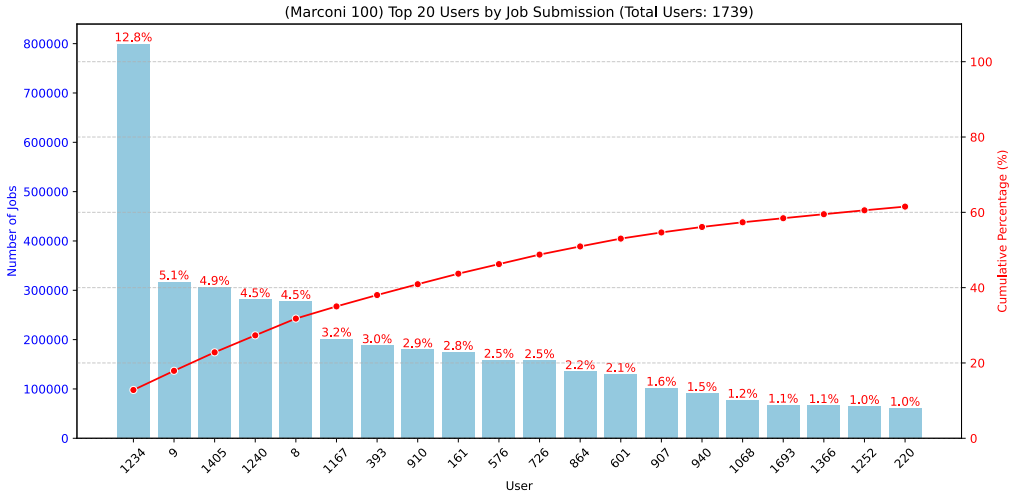
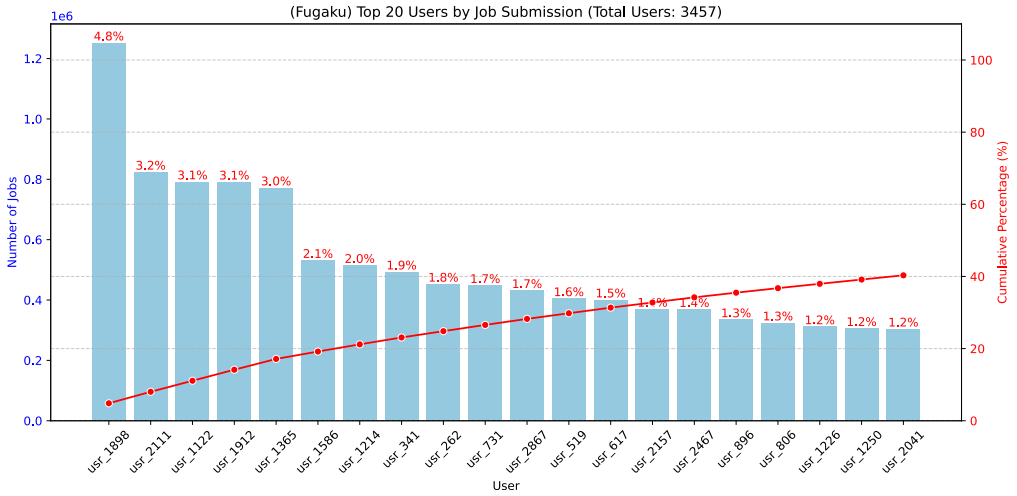


Figure A.19: Top 20 users by job submission count across systems.

(a) Marconi



(b) Fugaku



(c) Eagle

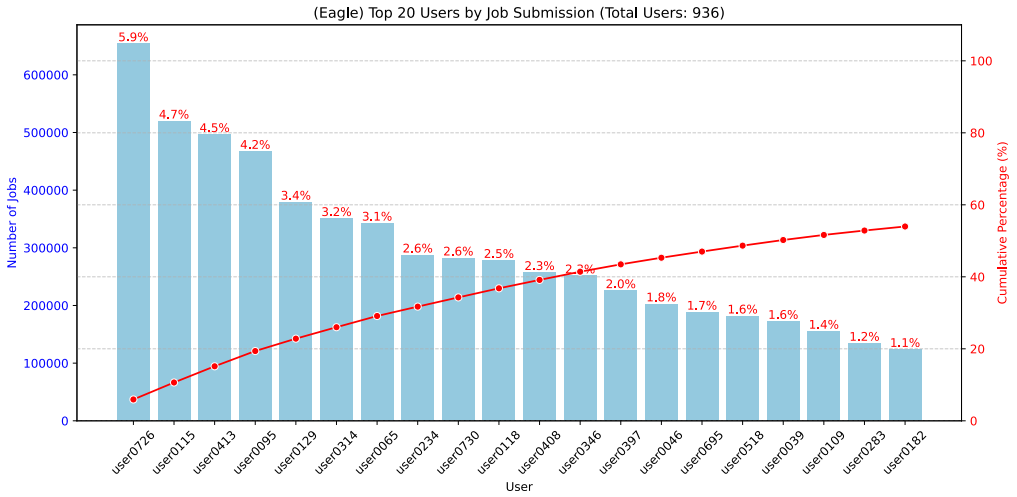
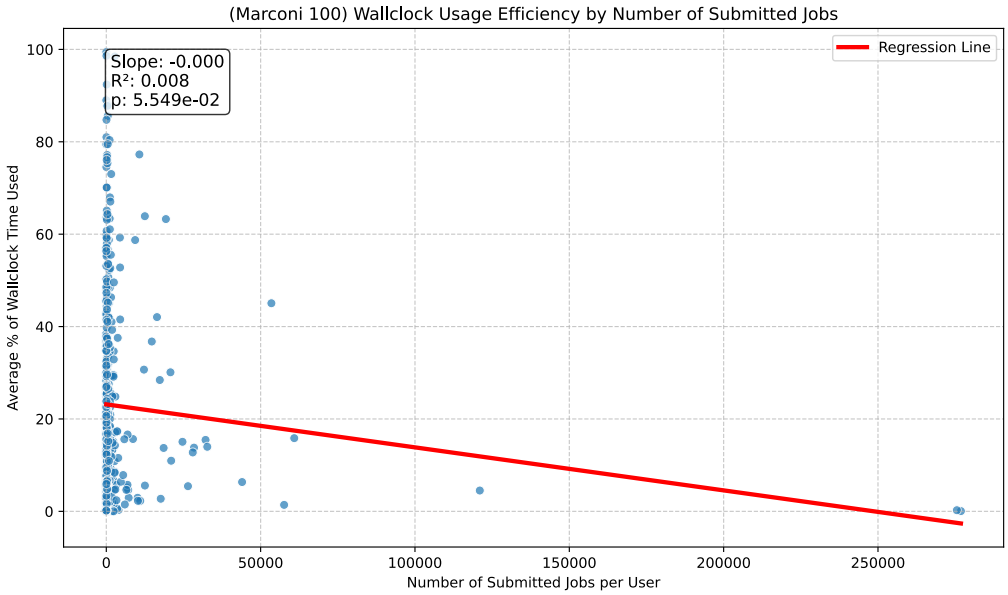
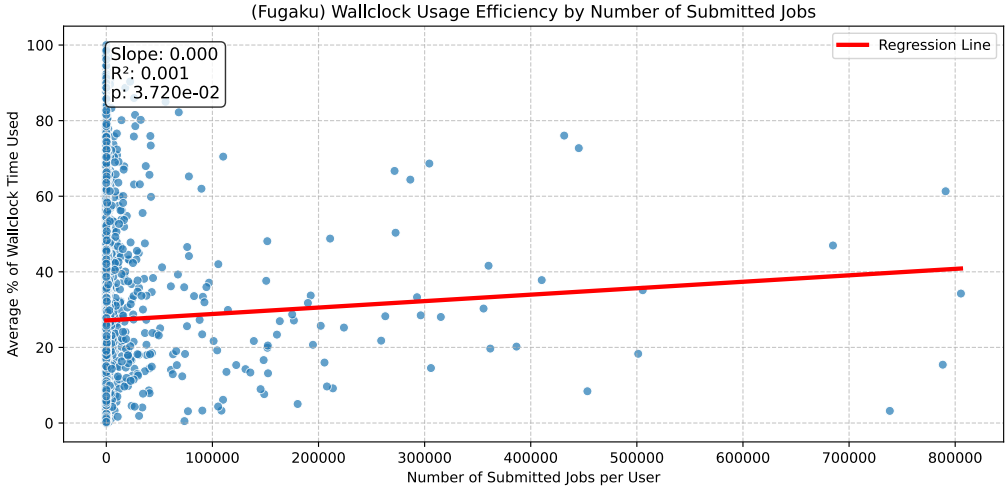


Figure A.20: Wall-clock usage efficiency vs. number of submitted jobs across systems.

(a) Marconi



(b) Fugaku



(c) Eagle

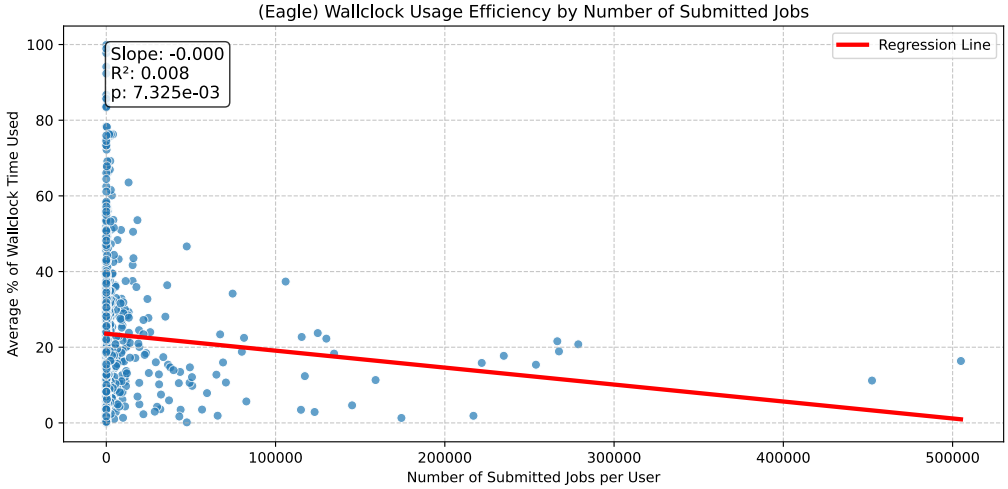
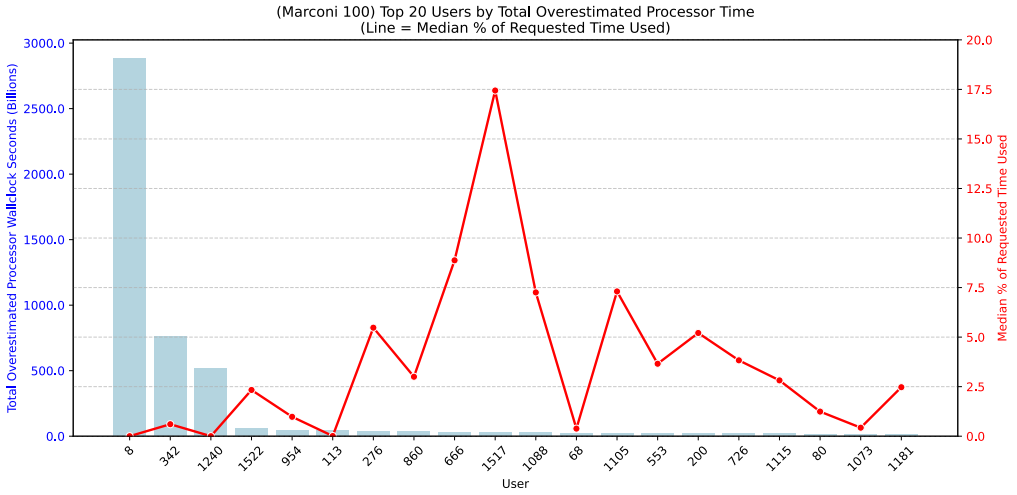
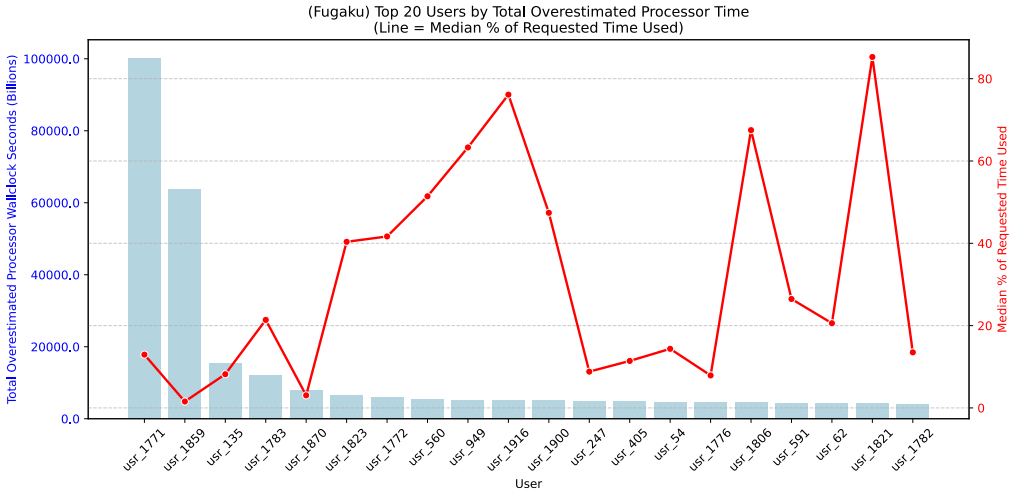


Figure A.21: Top 20 users by total overestimated processor time across systems.

(a) Marconi



(b) Fugaku



(c) Eagle

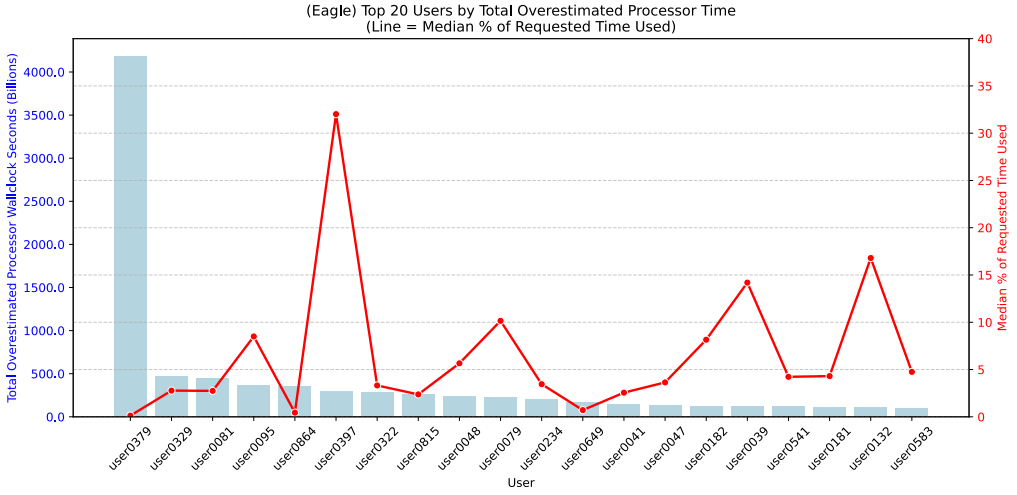
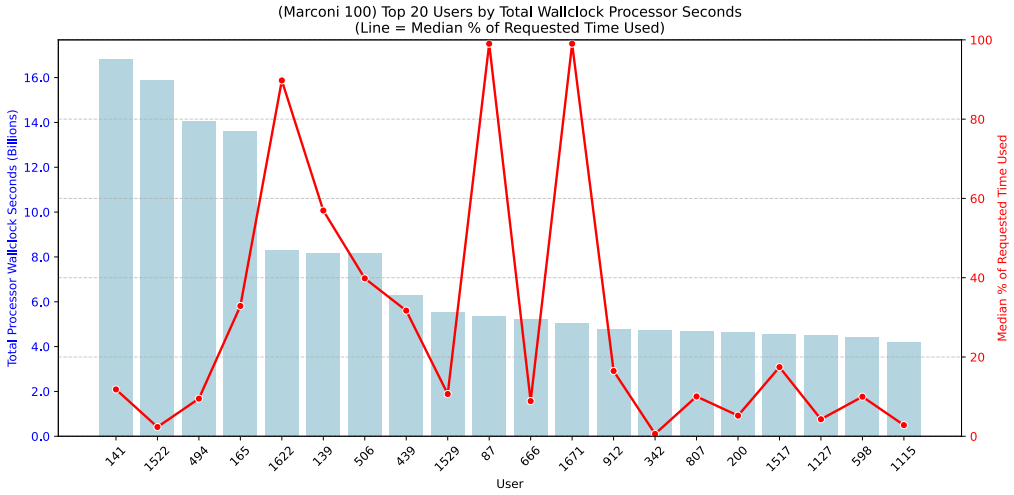
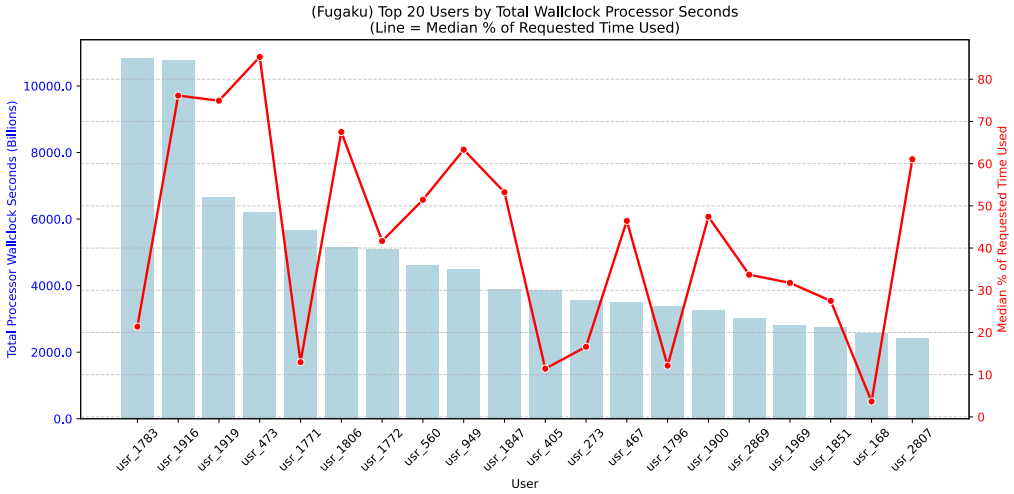


Figure A.22: Top 20 users by total wall-clock processor-seconds across systems.

(a) Marconi



(b) Fugaku



(c) Eagle

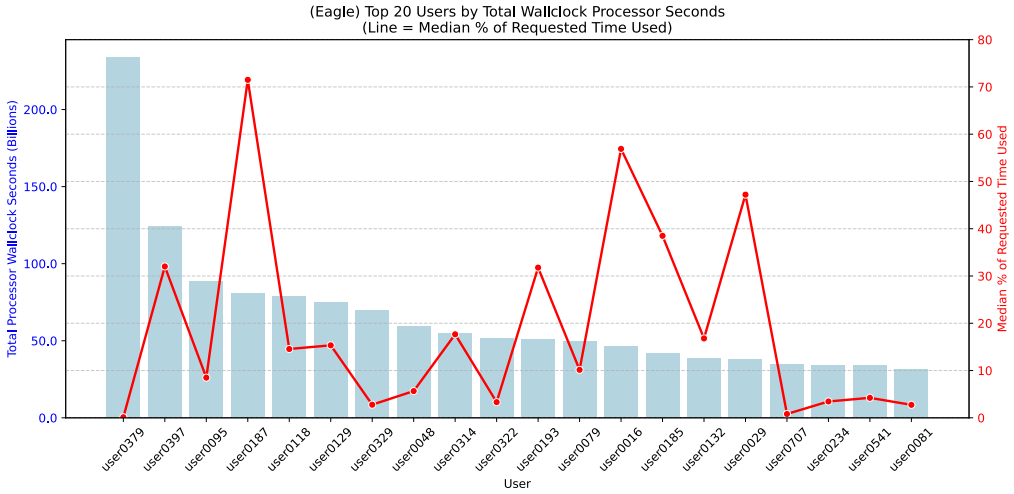
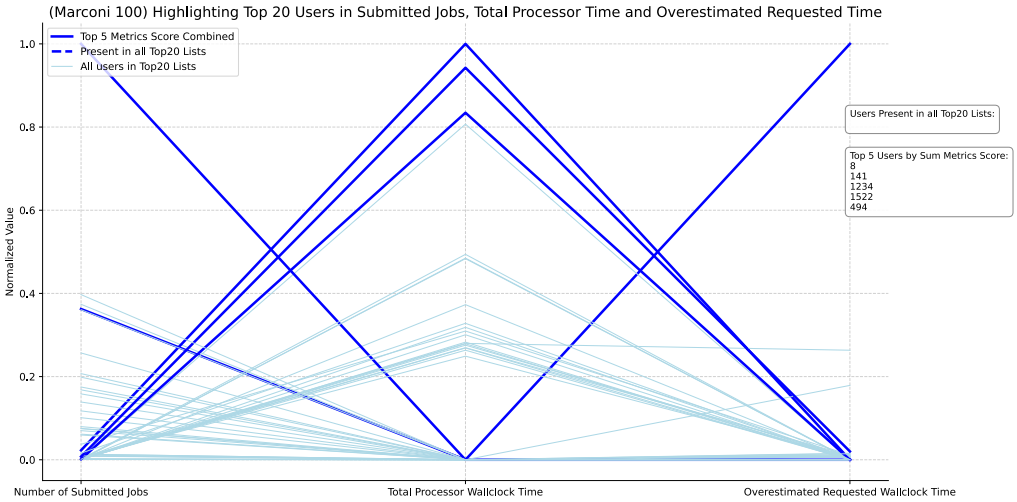
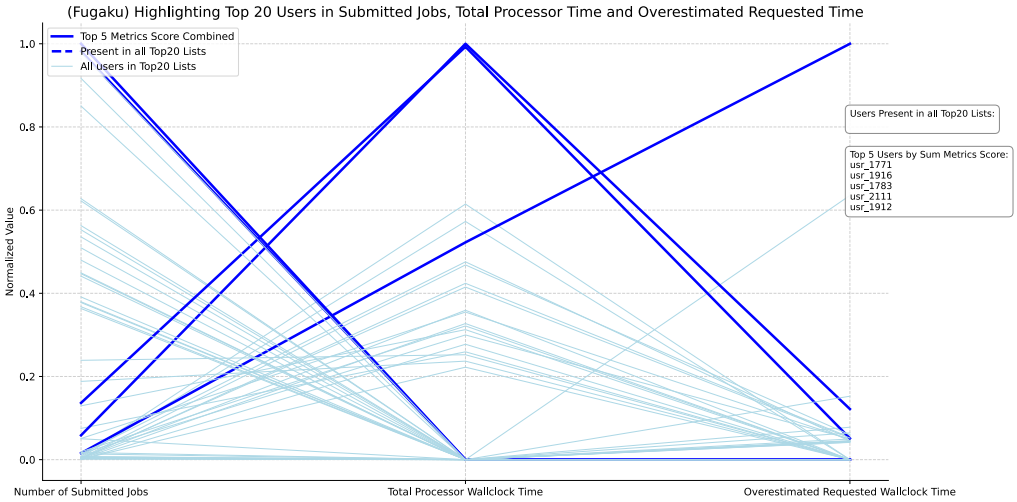


Figure A.23: Highlighting top 20 users by the sum of the normalized score of submitted jobs, total processor time and overestimated requested time.

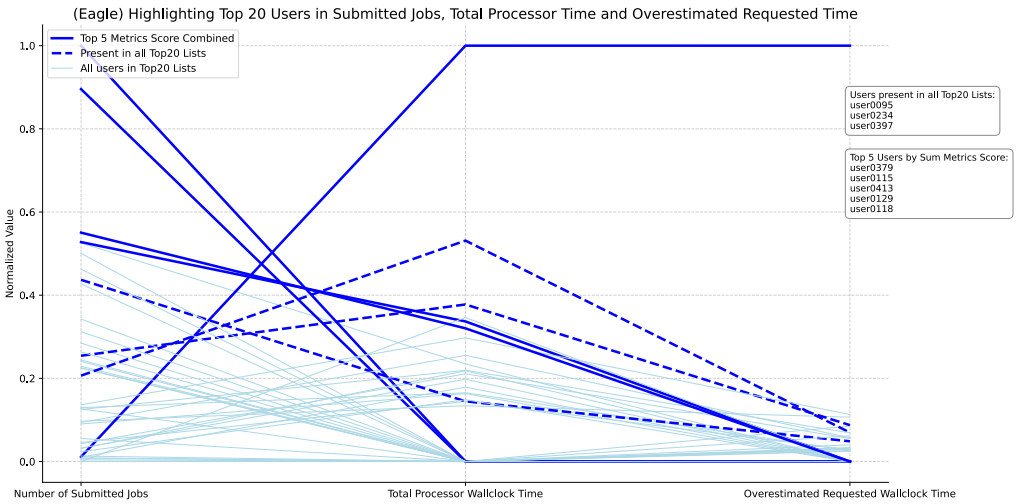
(a) Marconi



(b) Fugaku



(c) Eagle



A.0.2 Fugaku-Specific Analysis

Figure A.24: Fugaku Comparison of execution cycles and wallclock runtime.

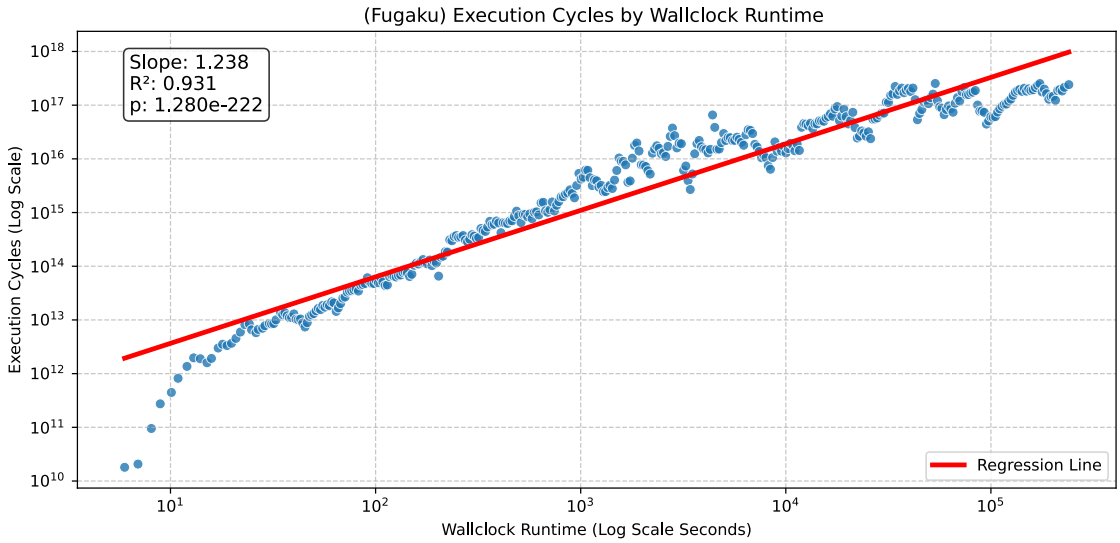


Figure A.25: Fugaku Comparison of sleep cycles and wallclock runtime.

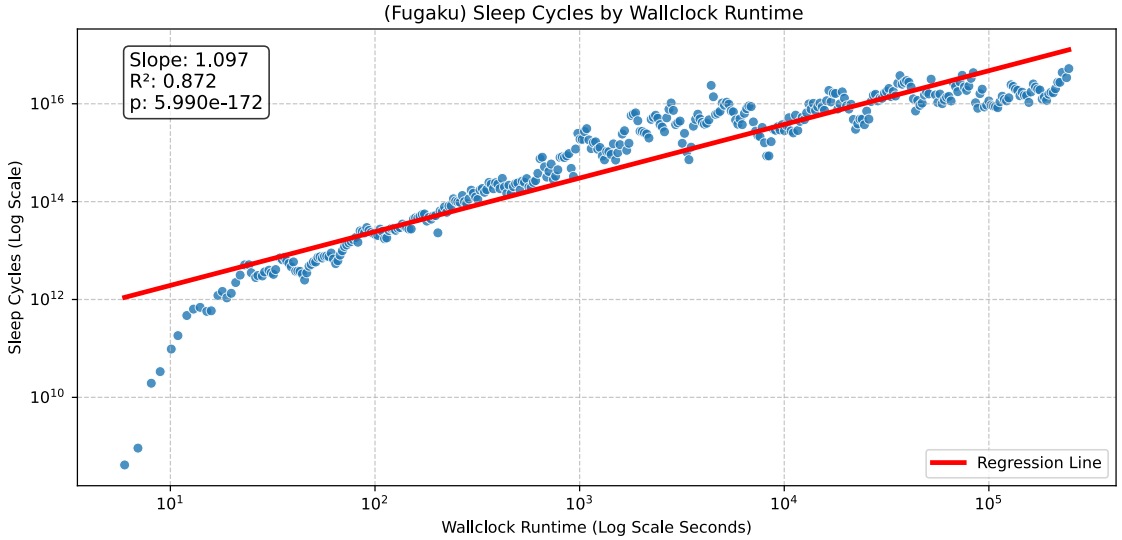


Figure A.26: Fugaku Analysis of the execution-to-sleep cycle ratio compared to wallclock runtime.

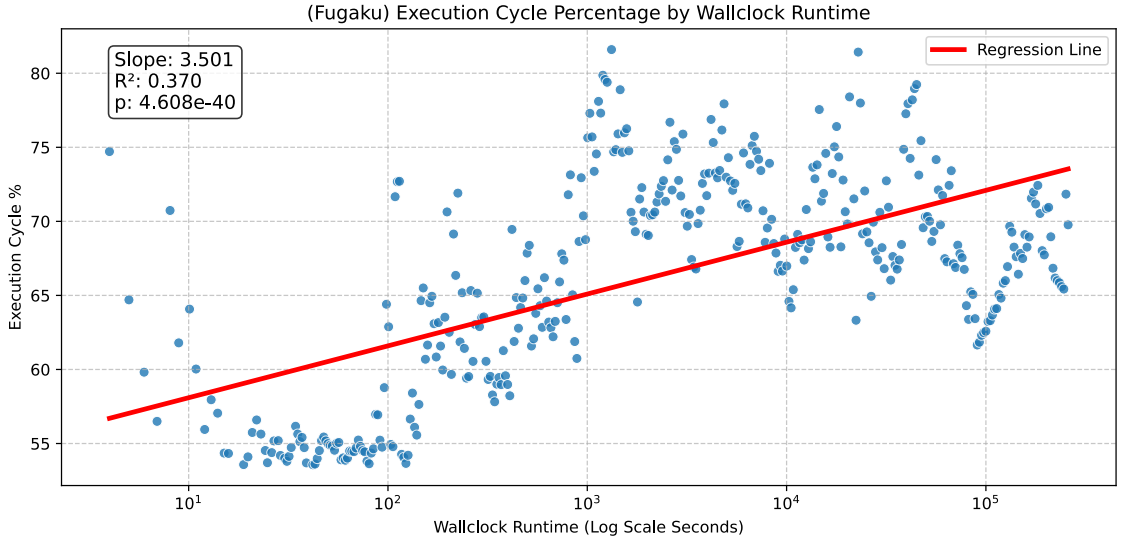


Figure A.27: Fugaku Comparison of requested and used processors.

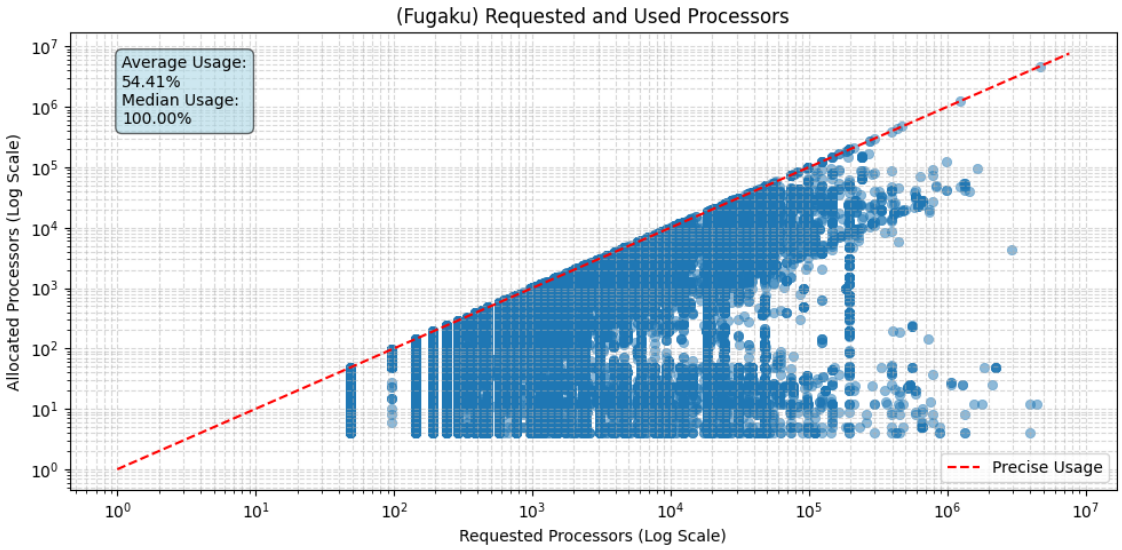


Figure A.28: Fugaku Comparison of requested and used processors grouped by requested processors.

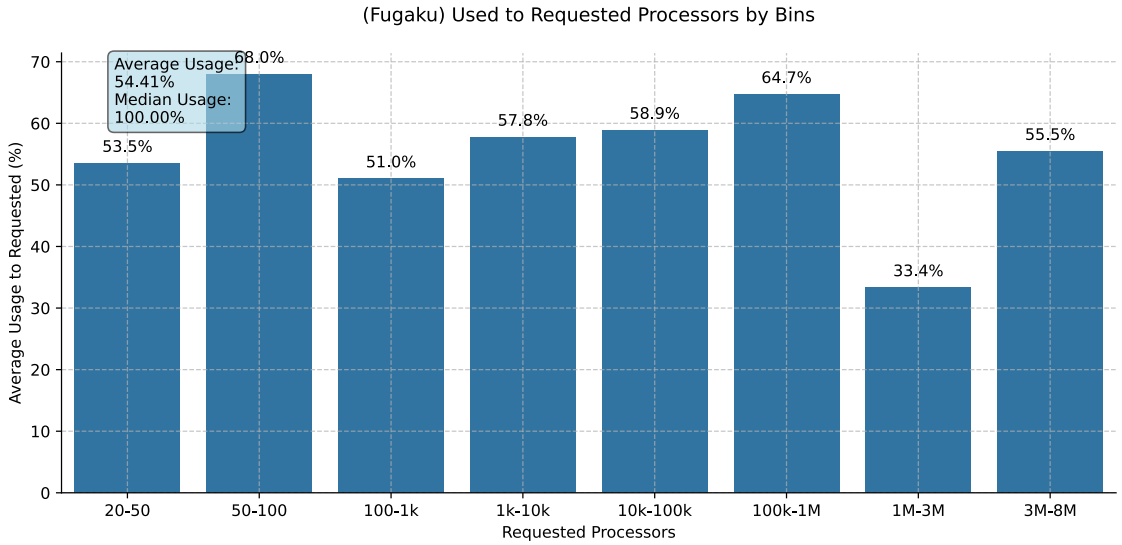


Figure A.29: Fugaku Comparison of requested and allocated nodes.

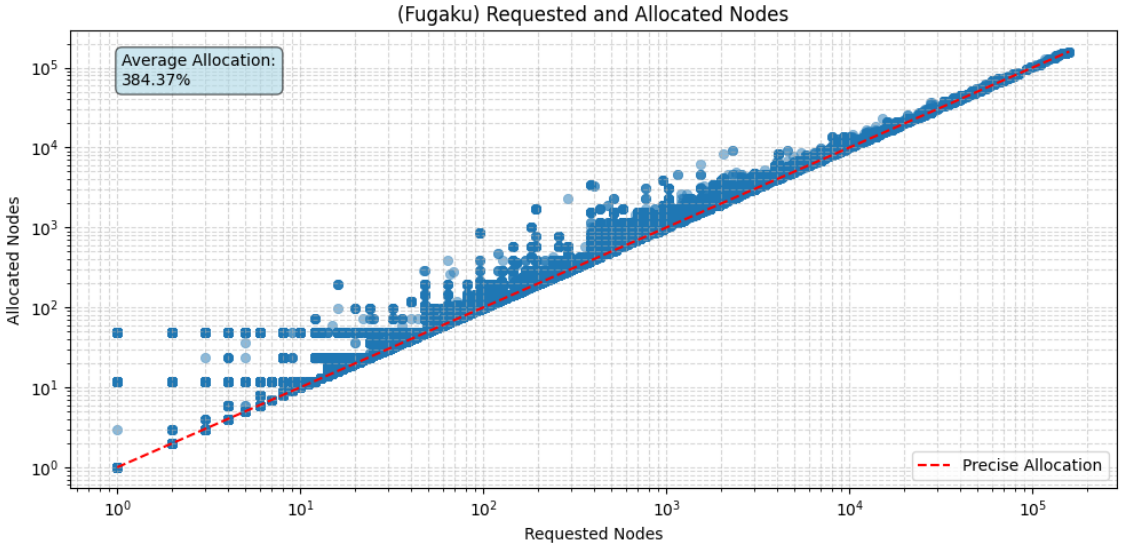


Figure A.30: Fugaku Comparison of requested and used nodes.

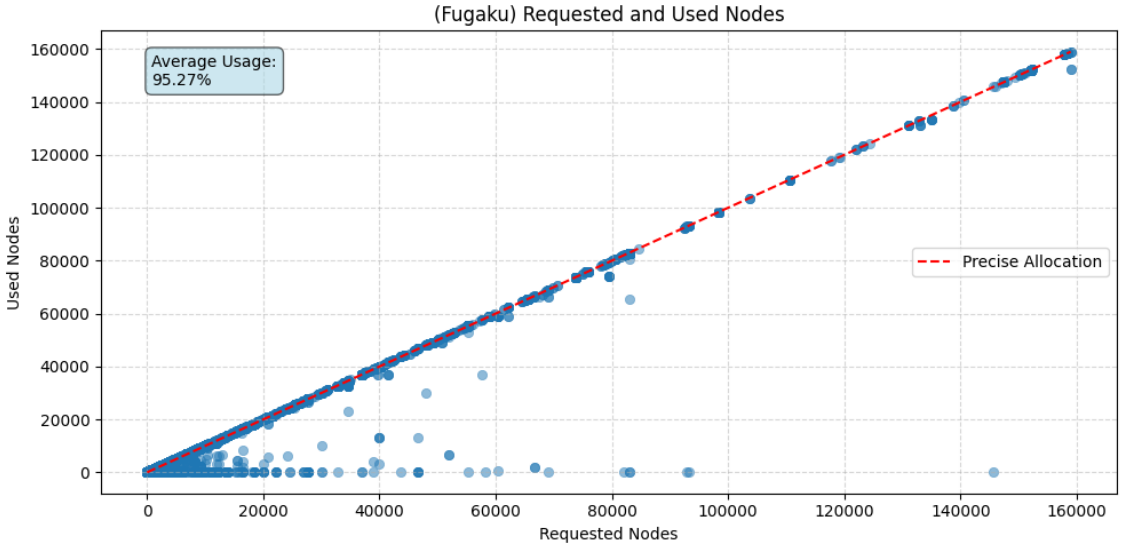


Figure A.31: Fugaku Comparison of requested and used nodes grouped by requested nodes.

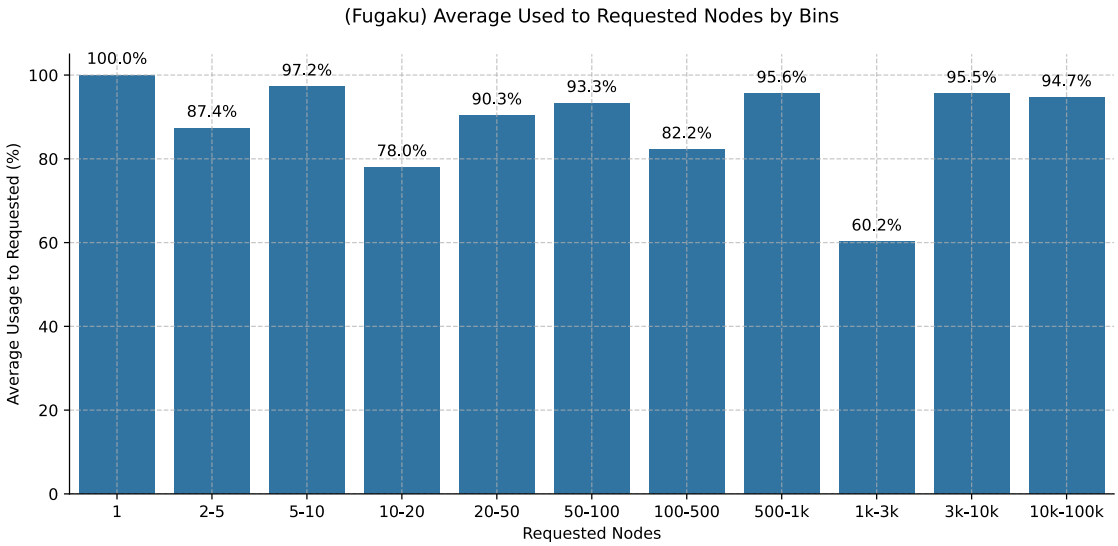


Figure A.32: Fugaku Comparison of allocated and used memory.

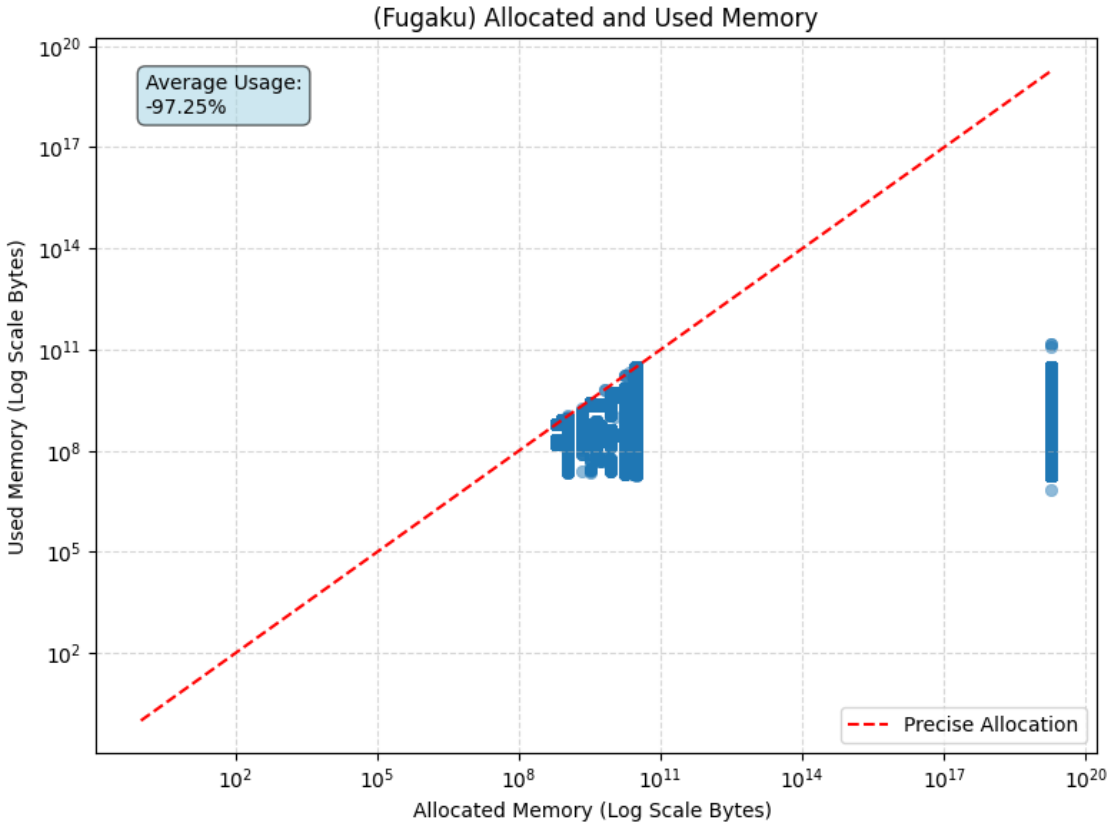


Figure A.33: Fugaku Analysis of used memory and floating-point operations (flops).

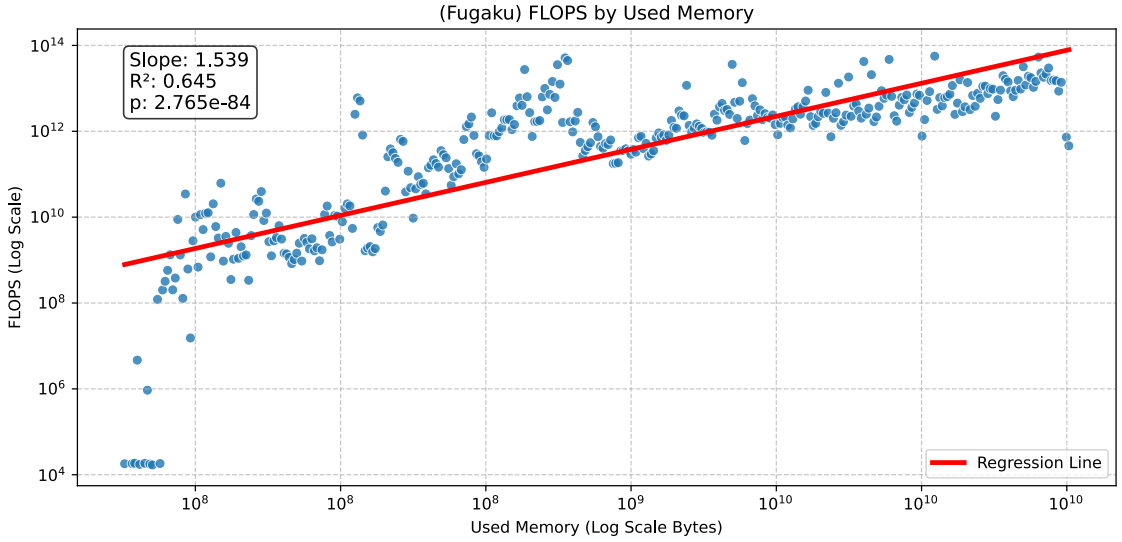


Figure A.34: Fugaku Analysis of flops and memory bandwidth.

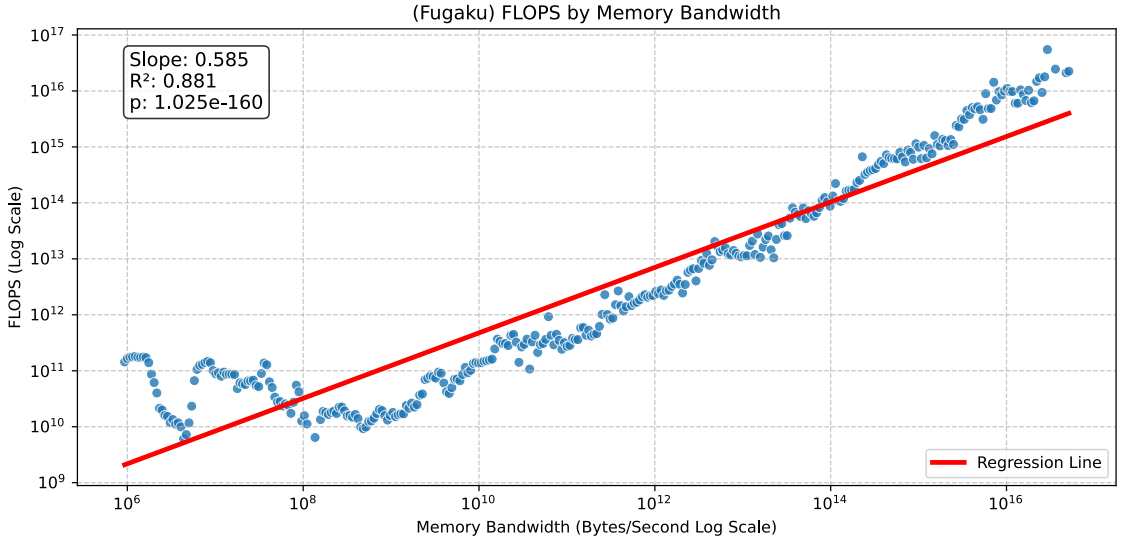


Figure A.35: Fugaku Analysis of used memory vs. memory bandwidth.

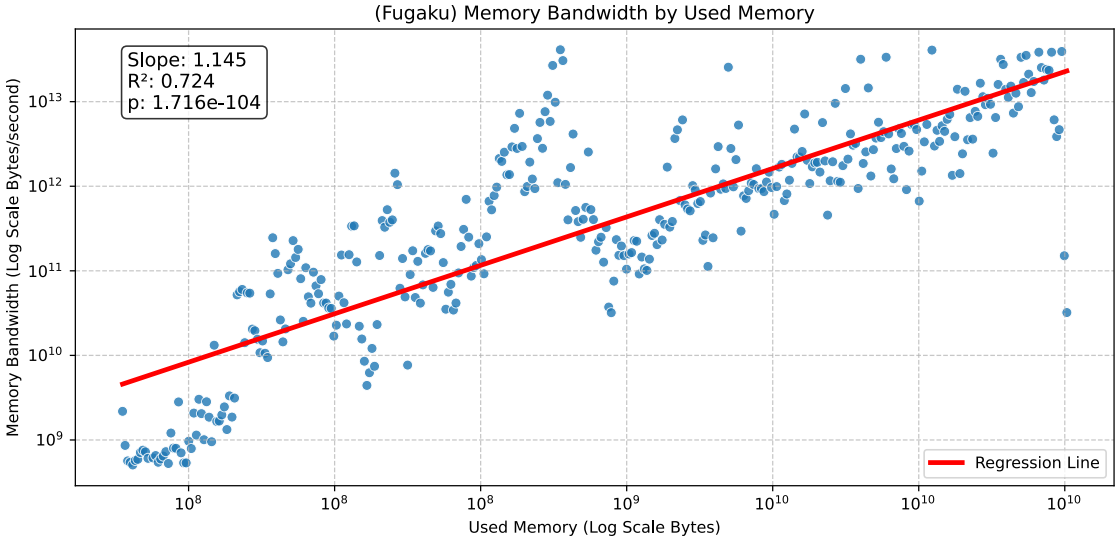


Figure A.36: Fugaku Comparison of requested and allocated nodes grouped by requested nodes.

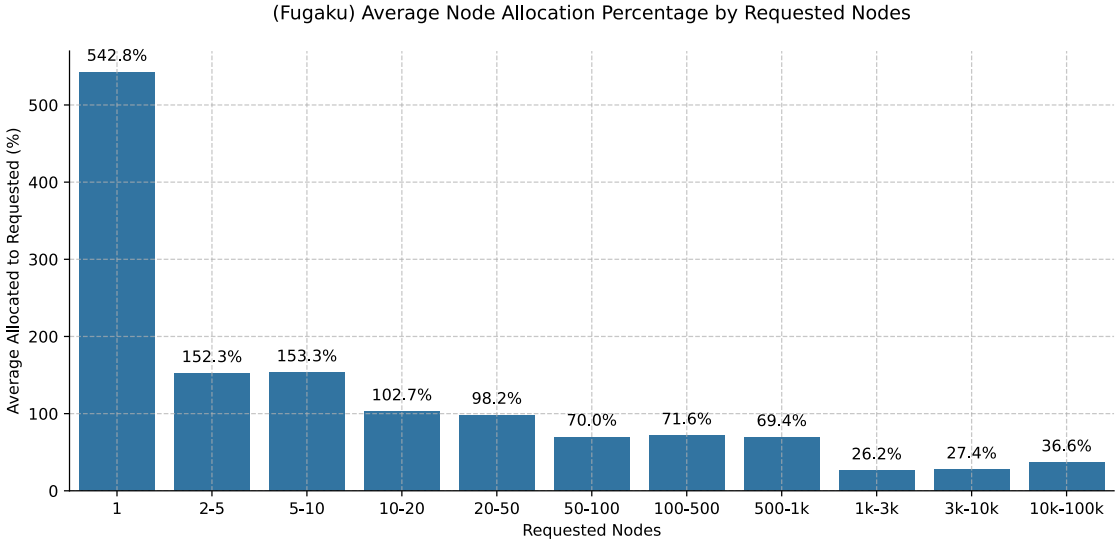


Figure A.37: Fugaku Relationship between user/system CPU time and memory bandwidth.

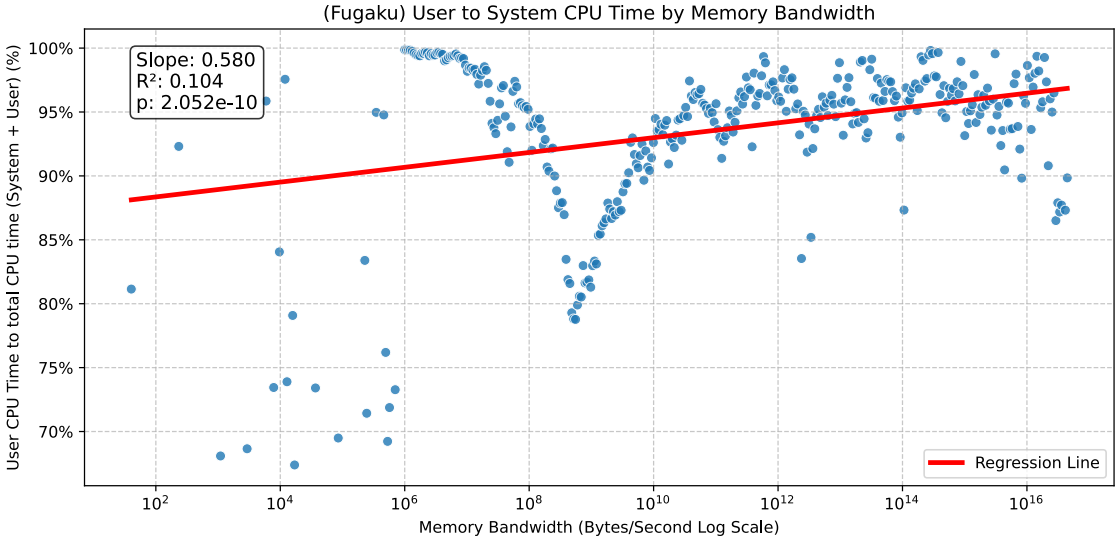


Figure A.38: Fugaku Analysis of average job idle time vs. used nodes.

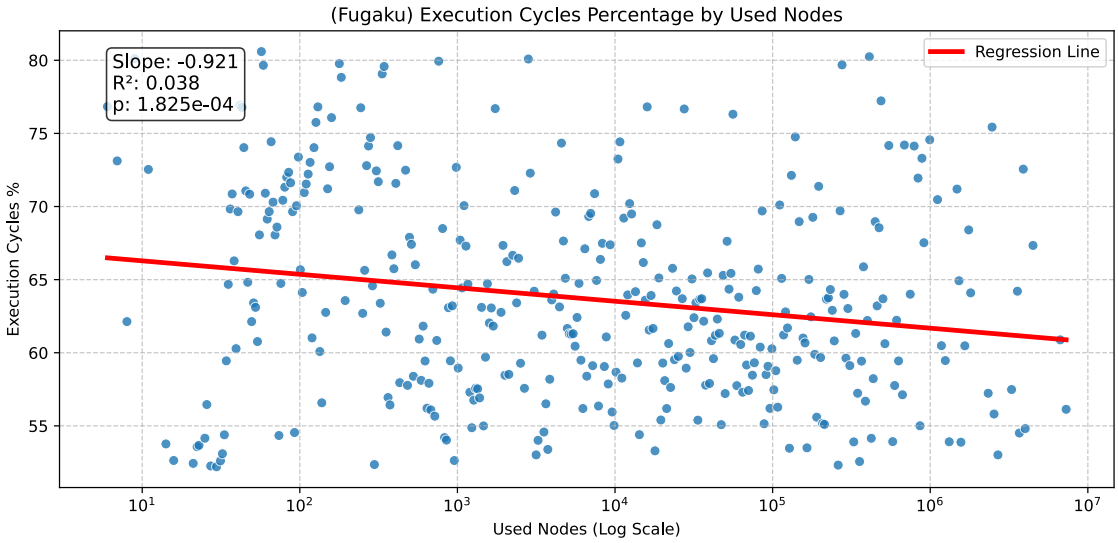


Figure A.39: Fugaku Analysis of execution-to-sleep cycles ratio compared to used processors.

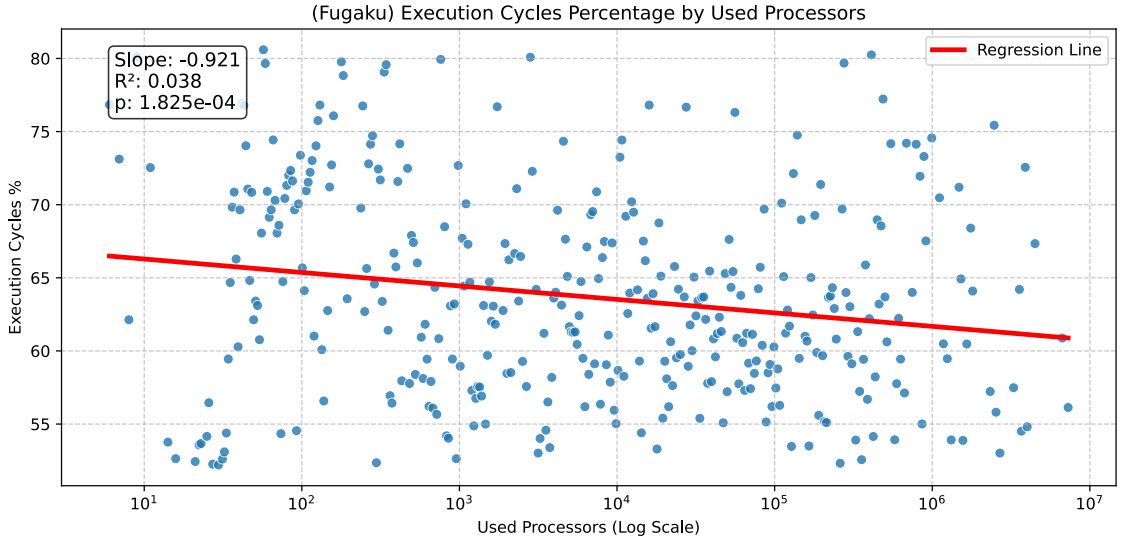
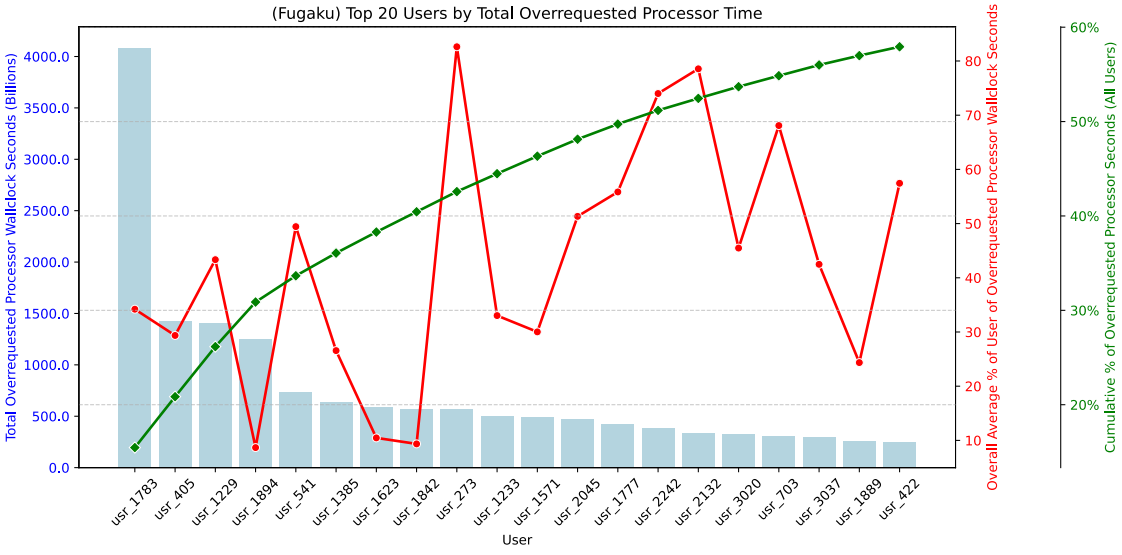


Figure A.40: Fugaku Analysis of user ID vs. underused resources.



A.0.3 Marconi-Specific Analysis

Figure A.41: Marconi100 Histogram of SLURM tasks requested by jobs.

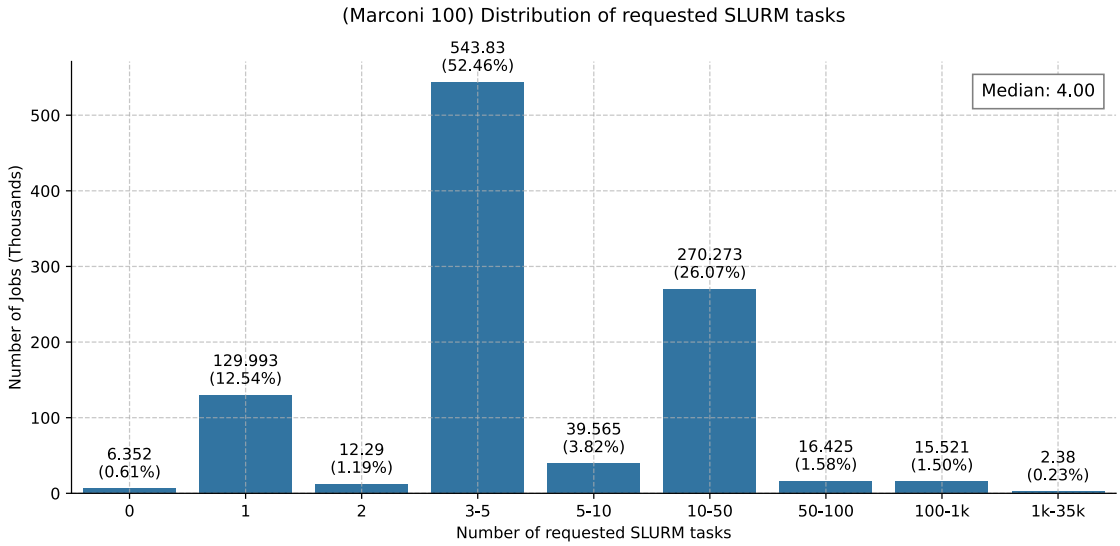


Figure A.42: Marconi100 Histogram of requested memory amount.

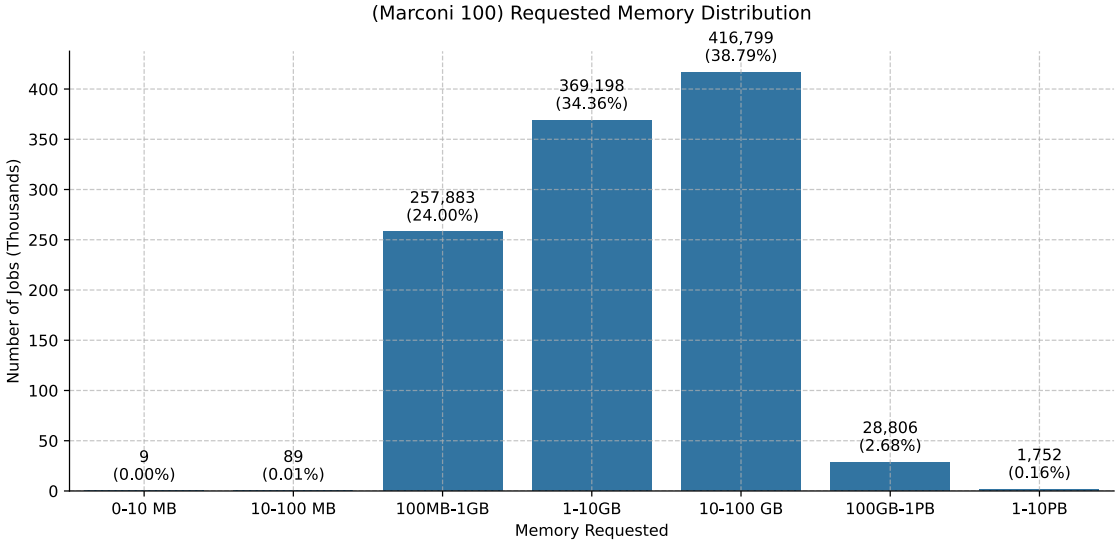


Figure A.43: Marconi100 Histogram of requested GPU amount.

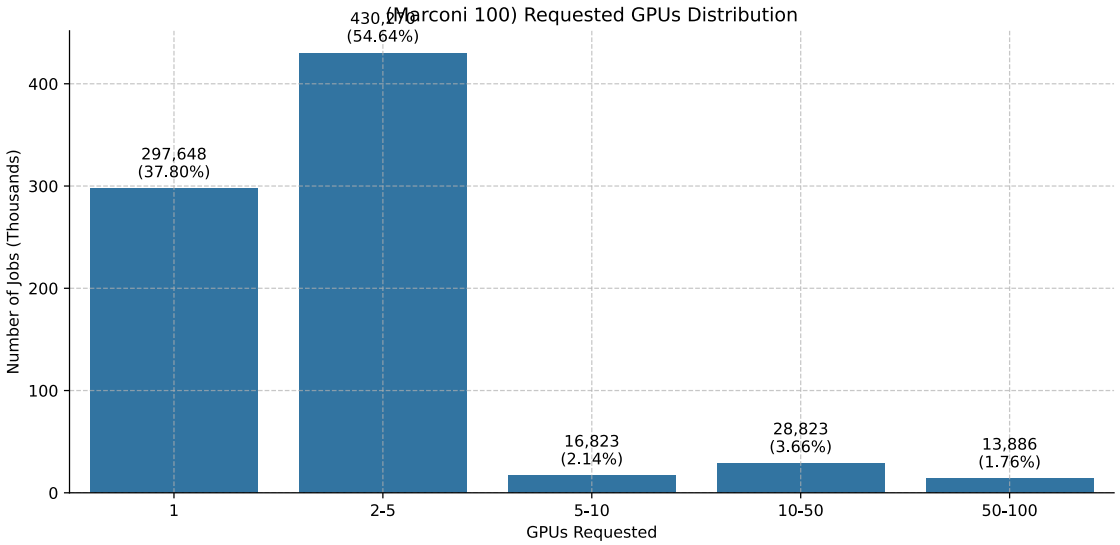


Figure A.44: Marconi100 Relationship between requested CPUs and GPUs.

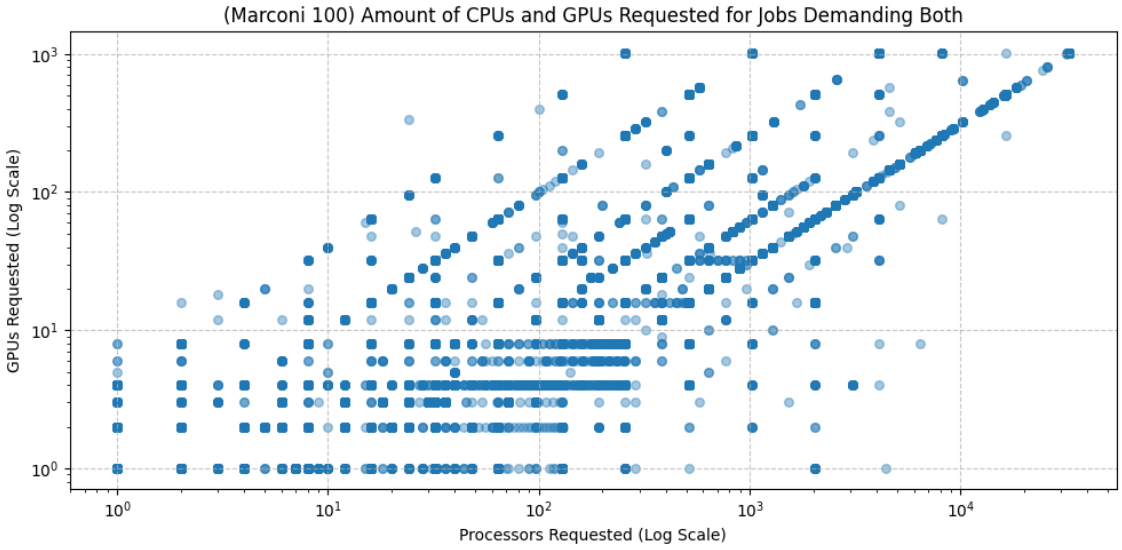


Figure A.45: Marconi100 Comparison of requested vs. allocated CPUs.

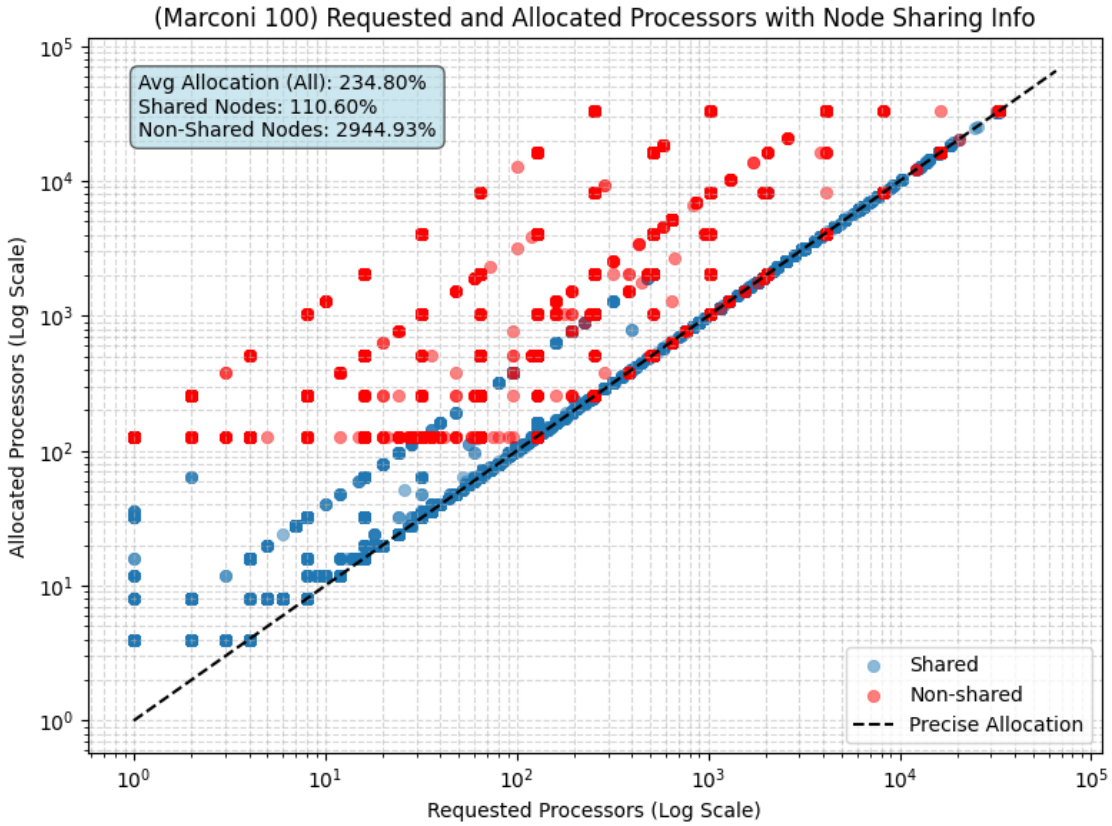


Figure A.46: Marconi100 CPU Allocation Percentage grouped by requested processors.

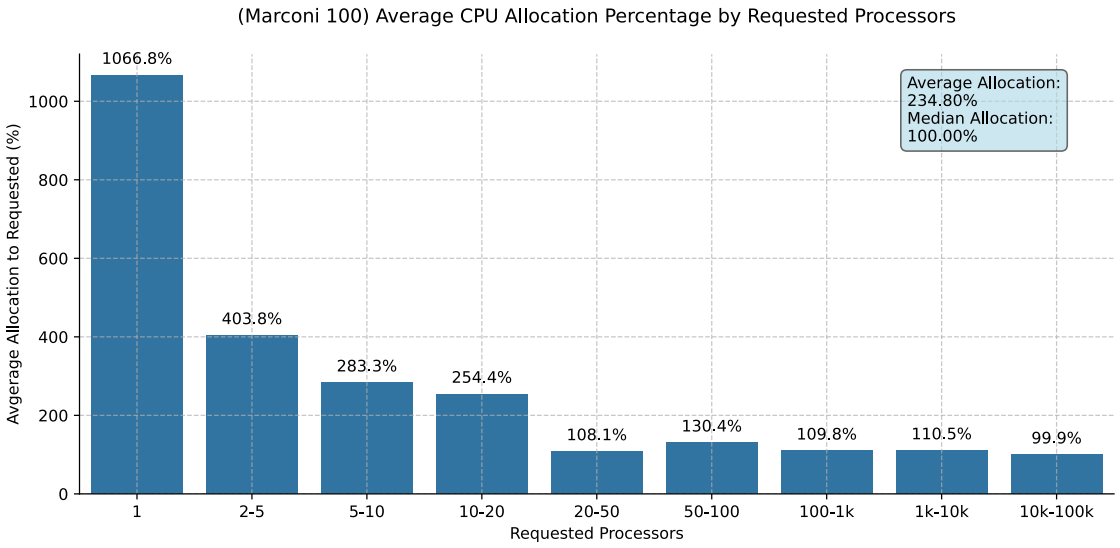


Figure A.47: Marconi100 Comparison of requested and allocated memory.

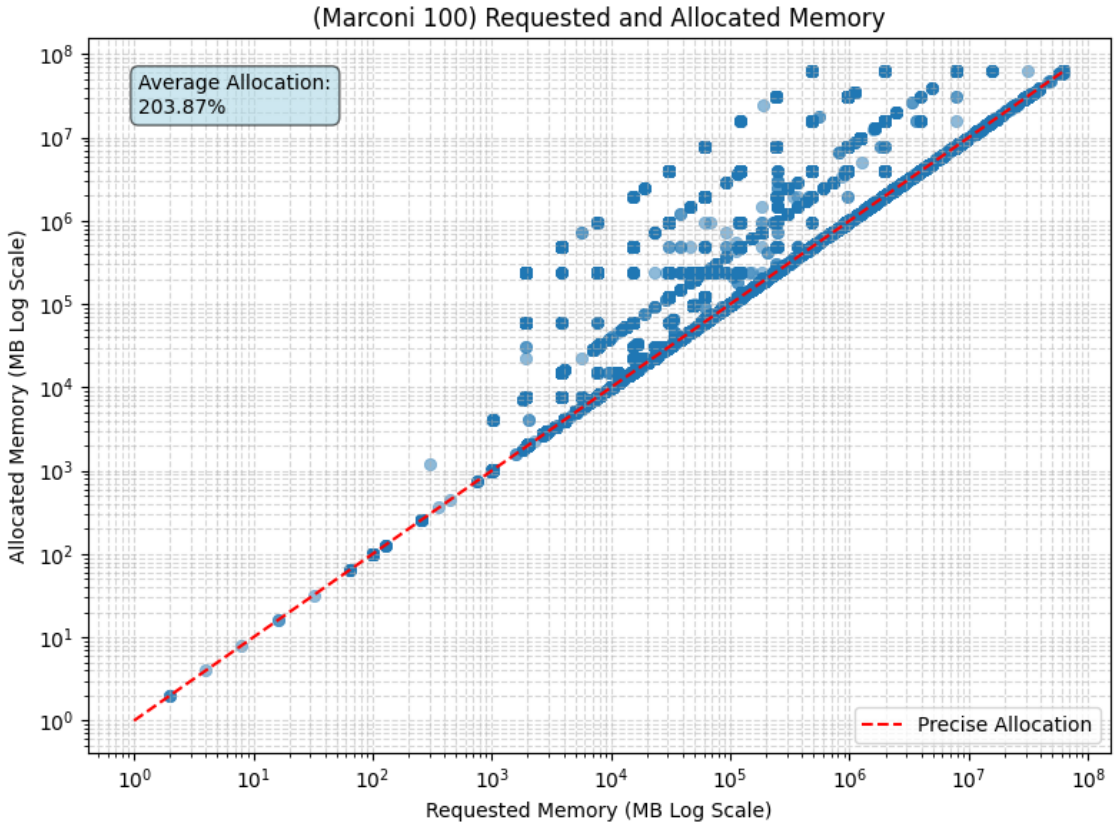


Figure A.48: Marconi100 Memory Allocation Percentage grouped by requested memory.

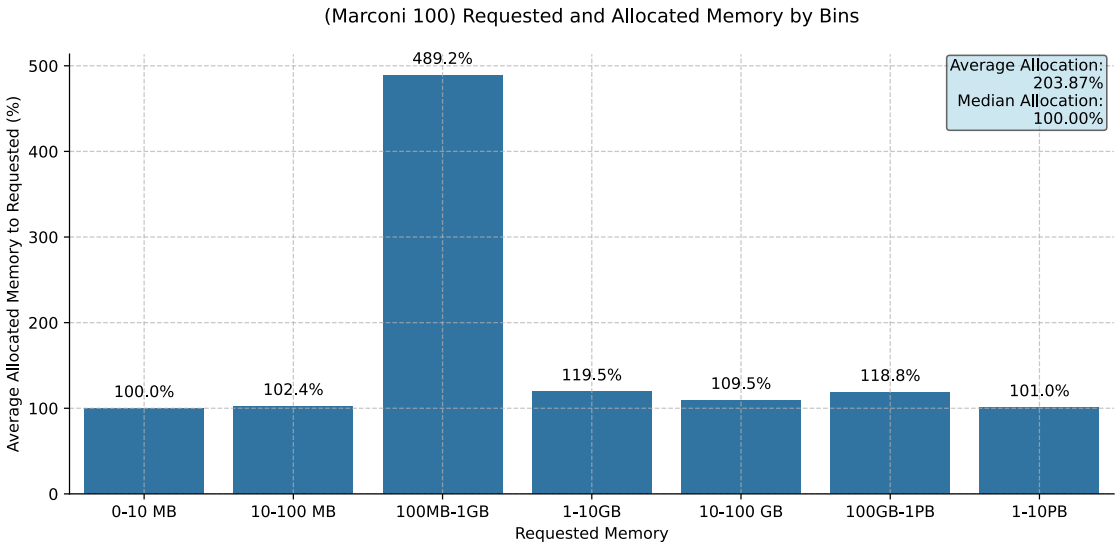


Figure A.49: Marconi100 Comparison of requested and allocated nodes.

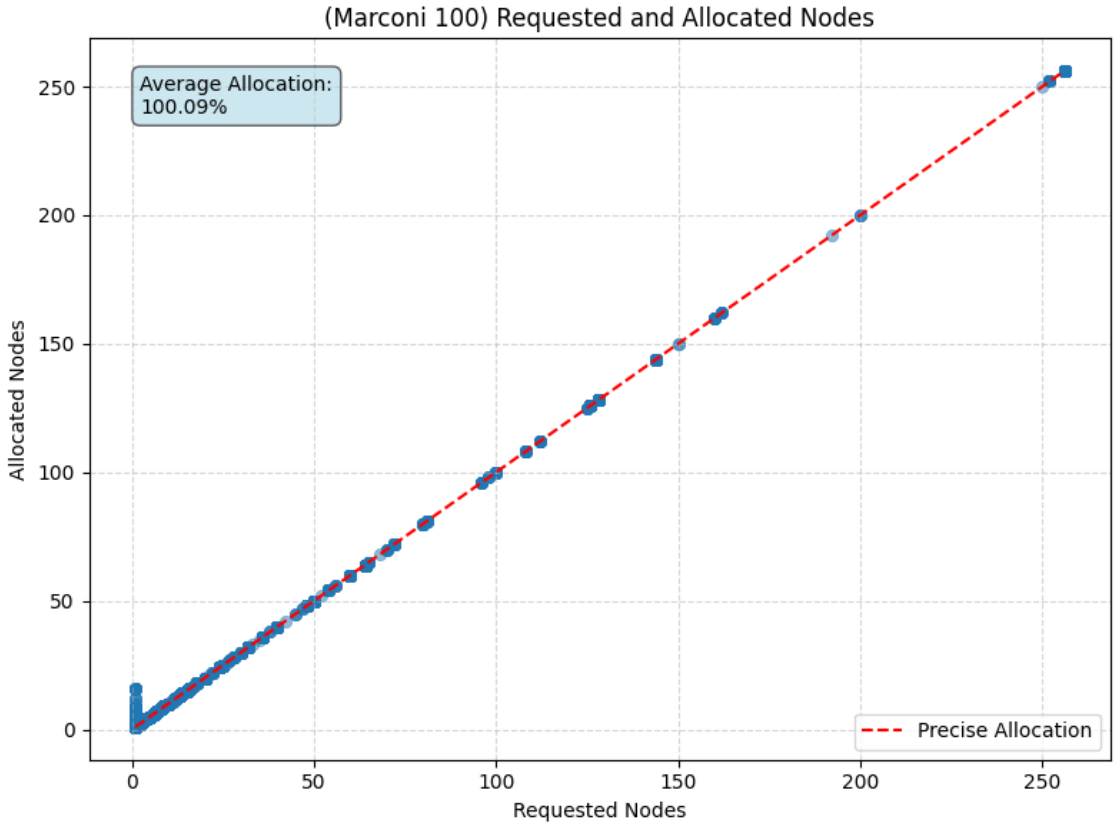


Figure A.50: Marconi100 Comparison of requested and allocated GPUs.

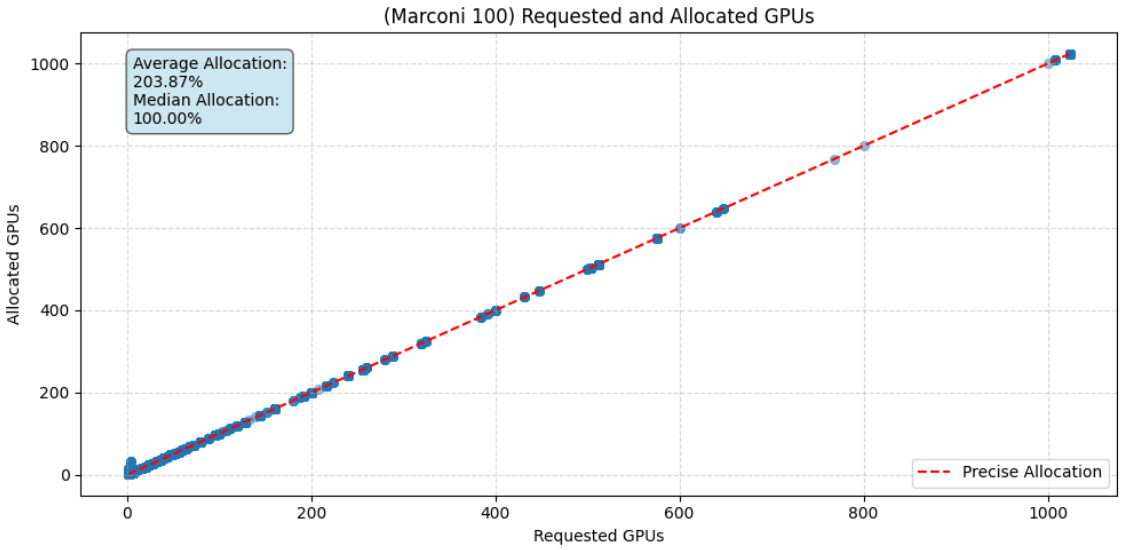


Figure A.51: Marconi100 Comparison of timeout percentage and requested wall-clock time.

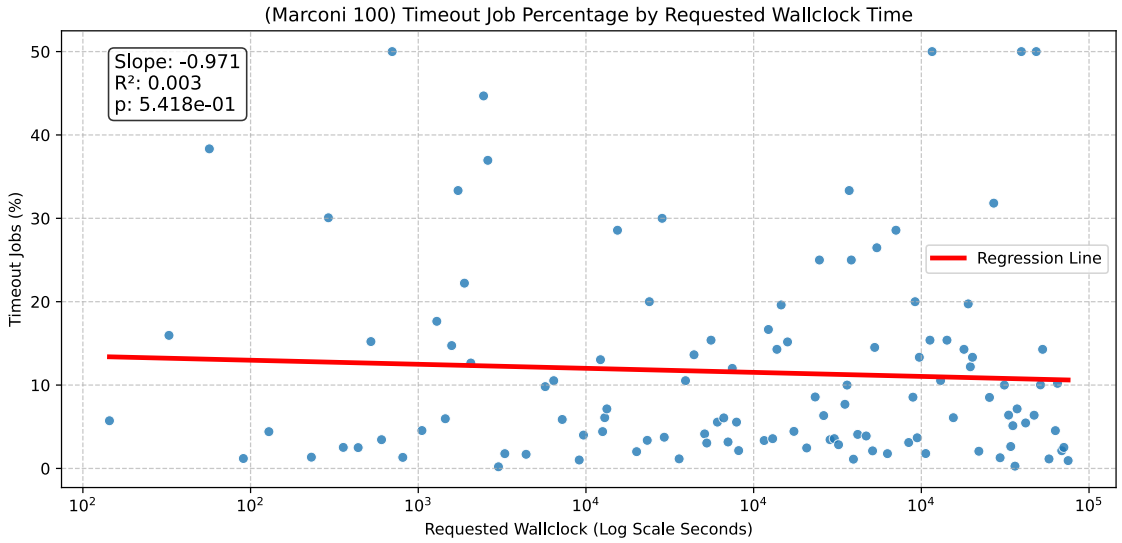


Figure A.52: Marconi100 Comparison of job cancellation percentage and wait time.

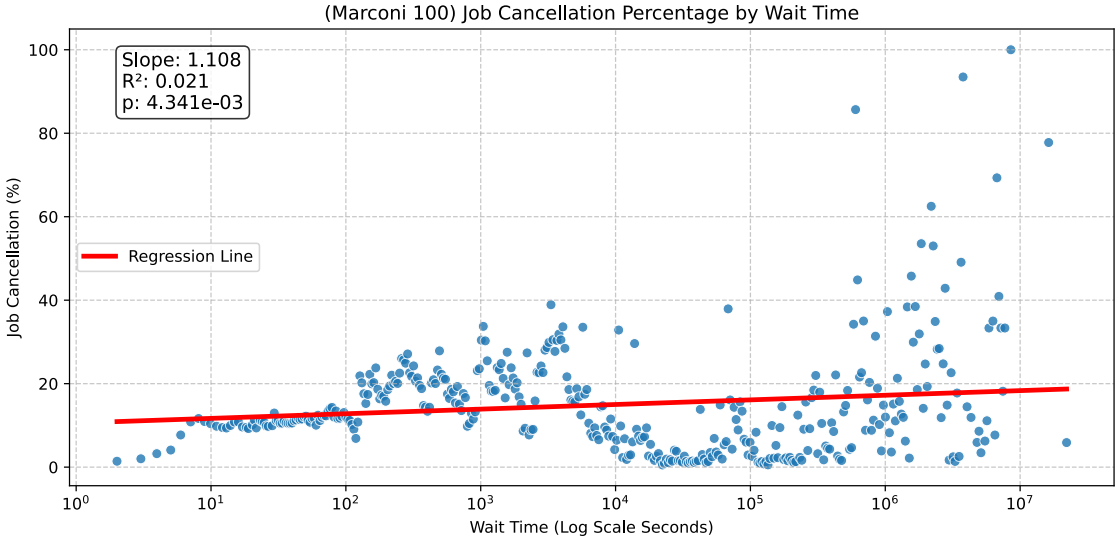
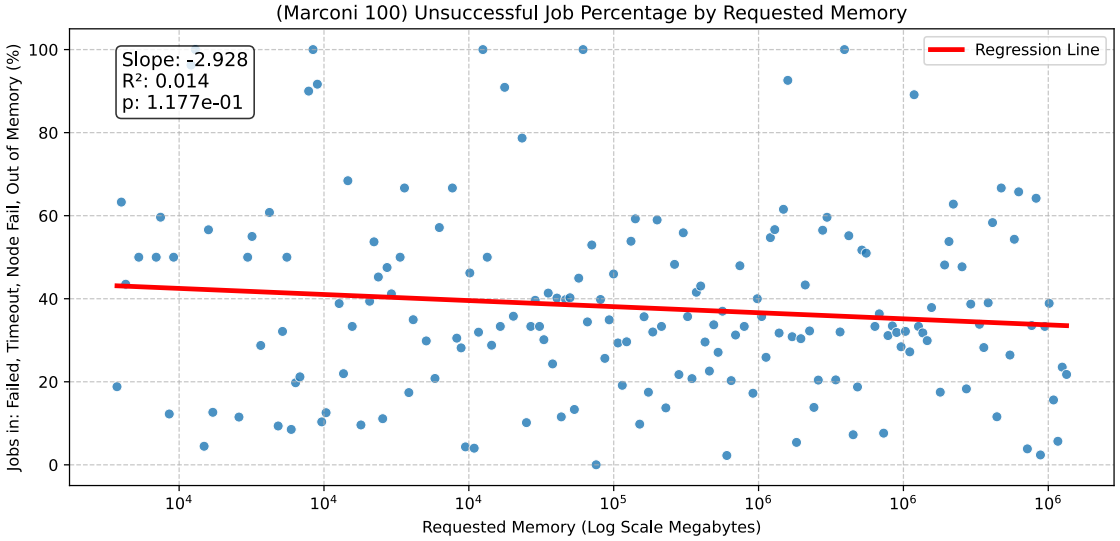


Figure A.53: Marconi100 Comparison of unsuccessful job rate and requested memory.



A.0.4 Eagle-Specific Analysis

Figure A.54: Eagle Histogram of requested memory amount.

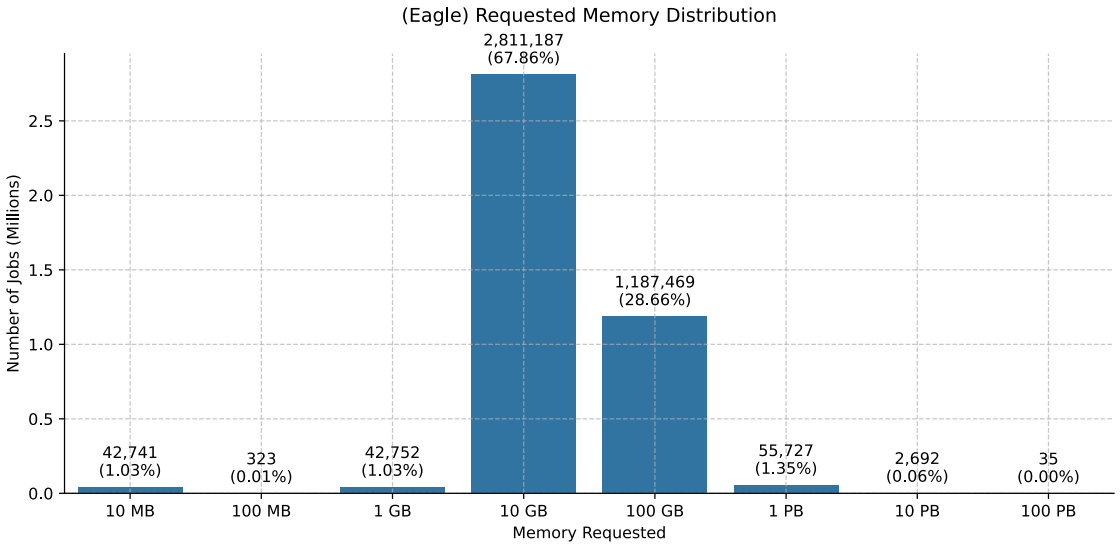


Figure A.55: Eagle Histogram of requested GPU amount.

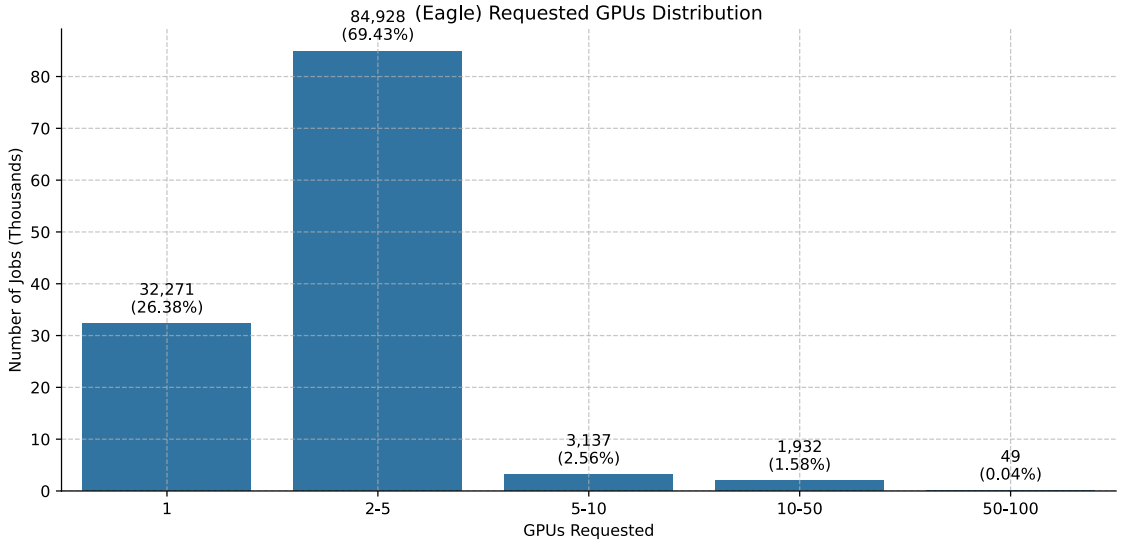


Figure A.56: Eagle Scatter plot analyzing the relationship between requested CPUs and GPUs.

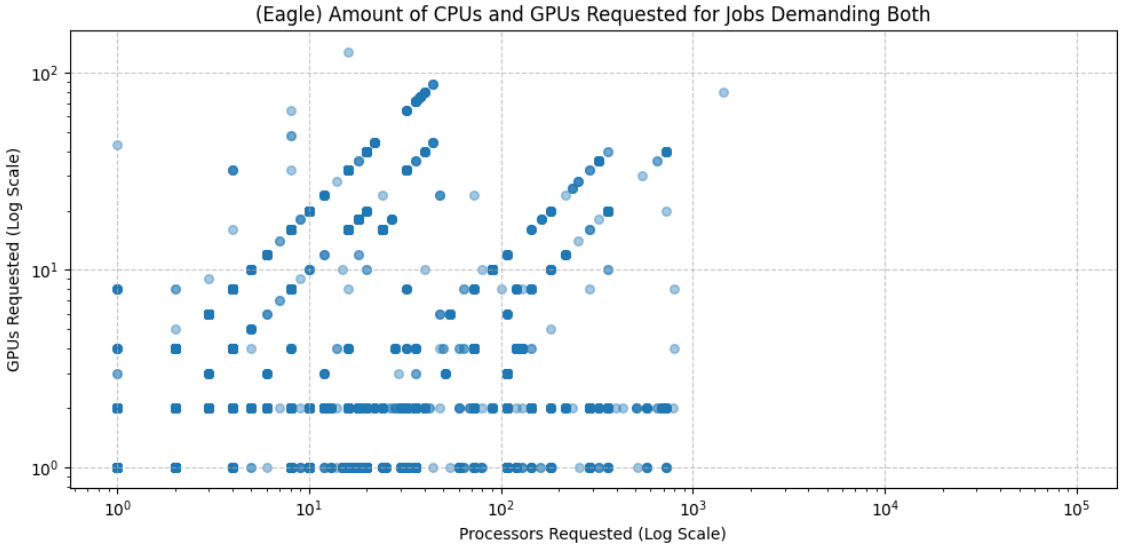


Figure A.57: Eagle Comparison of timeout rate and requested wall-clock time.

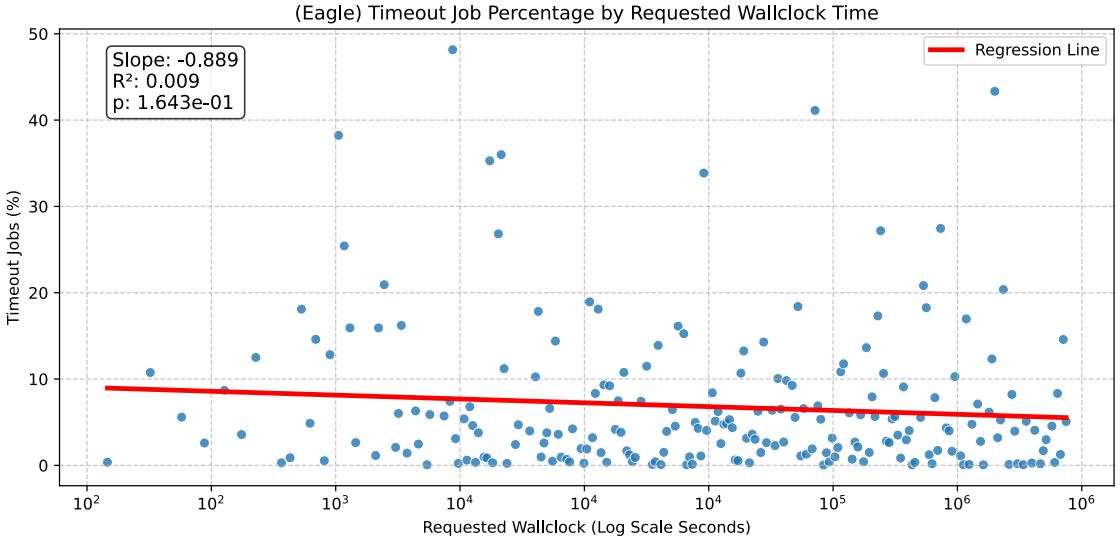


Figure A.58: Eagle Comparison of job cancellation percentage and wait time.

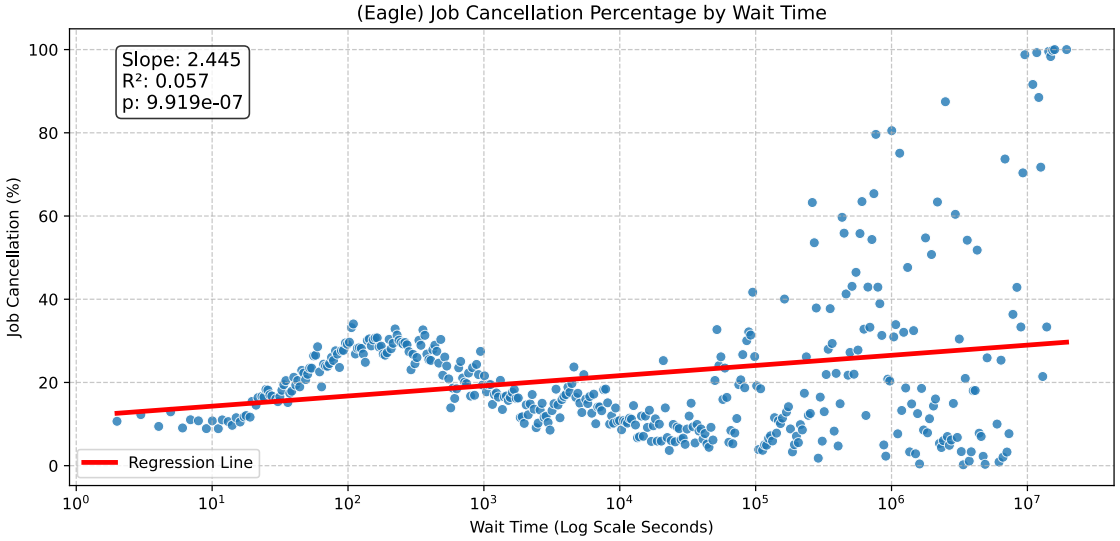
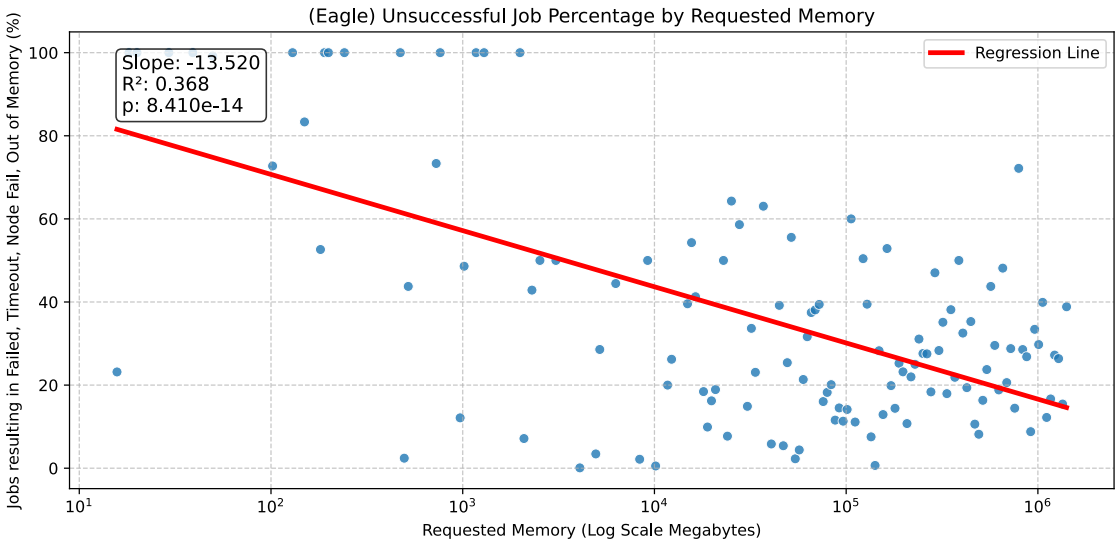


Figure A.59: Eagle Comparison of unsuccessful job rate and requested memory.



Appendix B

Topology Setup

Listing B.1: OpenDC Topology JSON for Marconi100

```
{
  "clusters": [
    {
      "name": "Marconi100",
      "count": 1,
      "hosts": [
        {
          "name": "IBM Power AC922",
          "count": 1,
          "cpu": {
            "vendor": "IBM",
            "modelName": "POWER9 8335-GTG",
            "arch": "ppc64le",
            "coreCount": 32,
            "coreSpeed": 2600,
            "count": 980
          },
          "memory": {
            "vendor": "unknown",
            "modelName": "unknown",
            "arch": "unknown",
            "memorySize": "250880 GB",
            "memorySpeed": -1
          }
        }
      ]
    }
  ]
}
```

Listing B.2: OpenDC Topology JSON for Fugaku

```
{
  "clusters": [
    {
      "name": "Fugaku",
      "count": 1,
      "hosts": [
        {
          "name": "Fujitsu A64FX",
          "count": 1,
          "cpu": {
            "vendor": "Fujitsu",
            "modelName": "A64FX",
            "arch": "arm",
            "coreCount": 48,
            "coreSpeed": 2200,
            "count": 158976
          },
          "memory": {
            "vendor": "HBM2",
            "modelName": "unknown",
            "arch": "unknown",
            "memorySize": "5085594 GiB",
            "memorySpeed": -1
          }
        }
      ]
    }
  ]
}
```

Listing B.3: OpenDC Topology JSON for Eagle

```
{
  "clusters": [
    {
      "name": "Eagle",
      "count": 1,
      "hosts": [
        {
          "name": "Eagle Node",
          "count": 1,
          "cpu": {
            "vendor": "Intel",
            "modelName": "Xeon Gold Skylake",
            "arch": "x86_64",
            "coreCount": 36,
            "coreSpeed": 3000,
            "count": 2114
          },
          "memory": {
            "vendor": "unknown",
            "modelName": "unknown",
            "arch": "unknown",
            "memorySize": "221184 GB",
            "memorySpeed": -1
          }
        }
      ]
    }
  ]
}
```