Energy Consumption Analysis for HPC Systems and Jobs

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: Erdem Koca
Email: erdem.koca@stud.unibas.ch

September 18, 2023

Abstract

As a subject of increasing global concern, energy consumption is a paramount issue, especially in computer science infrastructure where High-Performance Computing (HPC) systems contribute substantially to electrical consumption. The intense processing tasks performed by HPC systems demand considerable energy use and thus represent an opportunity for significant energy savings. This thesis aims to delve into the energy consumption patterns of a small, research and teaching-oriented HPC system over a four-month span, utilizing data from the Slurm energy plugin and system-level monitoring data from Ganglia.

The goal is to identify potential areas for energy conservation and propose strategic solutions. Achieving this requires a comprehensive understanding of job patterns and potential energy drainage points within the system. This bachelor thesis undertakes analysis and visualization of this data with the ultimate goal of formulating strategies to reduce energy consumption.

In the course of this research, we identified specific patterns in both node-level and job-level energy consumption, which can be leveraged to conserve energy. By intelligently powering down nodes, we demonstrated that energy consumption could be reduced significantly. The insights derived from this thesis can serve as a foundation for implementing energy-saving practices in HPC systems.

Contents

Chapter 1

Introduction

In the current climate, energy consumption has gained significant global attention. A critical part of this conversation revolves around the computer science infrastructure, specifically, High-Performance Computing (HPC) systems. A staggering 3% [1] of the world's electrical supply is consumed by these systems, a testament to their heavy energy requirements. This prevalence of HPC systems creates a substantial opportunity for energy savings, an opportunity we seek to explore in this thesis.

Our investigation centers on the University of Basel's High-Performance Computing system, familiarly known as the miniHPC. Drawing from a wealth of data that the miniHPC [24] provides, we aim to thoroughly understand its energy consumption patterns. The objective is to leverage this understanding and the insights gained from our research to propose viable energy conservation strategies that could significantly impact the broader field of HPC.

Given the demanding nature of tasks performed by HPC systems, their energy consumption tends to be high. This necessitates a holistic comprehension of job patterns and potential problem areas, especially if we are to achieve substantial energy savings. By analyzing and visualizing the collected data from the miniHPC, we anticipate to identify these patterns and problem areas, thus formulating strategies to decrease overall energy usage.

The insights revealed through our work are of potential significance not only to the University of Basel but to any institution employing HPC systems. By intelligently managing energy usage in these systems, substantial energy savings can be realized. This contribution is of great importance in our collective journey towards energy conservation, while still ensuring the productive and efficient functioning of HPC systems.

1.1 Motivation

The significance of managing energy consumption is underscored by the alarming global energy consumption trends, driven largely by technological advancements. High-Performance Computing (HPC) systems have emerged as significant energy consumers [1], amplifying the need for efficient utilization strategies. The university's miniHPC presents a case with the potential for energy savings. The potential outcomes of this investigation extend beyond energy conservation, including cost reductions, environmental benefits, and a strong foundation for future analyses. This thesis thus serves as the launchpad for evolving sustainable, energy-saving strategies for the miniHPC.

1.2 Research Questions and Goal

This Bachelor thesis aims to analyze and visualize energy consumption data from the miniHPC by addressing the following research questions:

1. How much energy does an HPC job consume?
2. How much energy do nodes consume in idle/busy mode?
3. Does powering down nodes save energy overall?
4. What is the best strategy for powering up/down nodes?

Guided by these research questions, we will work with two datasets: node-level and job-level. The node-level dataset provides information on the energy consumption of individual nodes in the HPC cluster, while the job-level dataset offers insights into the energy consumption of different HPC jobs. Combining these two datasets will provide a comprehensive understanding of the system's energy consumption and inform potential energy-saving strategies. The ultimate goal is to pinpoint potential areas for energy savings and gain an overarching understanding of energy consumption within the system.

1.3 HPC

High-Performance Computing is a practice to use parallel data processing for improving computing performance with aggregating computing power as a Cluster or computing power of multiple stand-alone computers to perform complex operations and process large volumes of data at high speeds [23]. This way it delivers much higher power and better performance than traditional computers, workstations, and servers. The history of HPC can be traced back to the mid-20th century with parallel processing and cluster computing [8]. The advantage of HPC is that you can solve complex tasks very fast, which means HPC systems can perform massive amounts of calculations very quickly. Nowadays it is indeed important to use HPC systems to process such big data that we have today in the world.

The efficiency that HPC solutions provide facilitates groundbreaking research and helps to drive innovation in many areas. It reduces time to completion and allows researchers to quickly implement and scale their ideas into valuable products for a more significant number of people. Also, HPC enables scientists and engineers to focus on solving scientific problems, such as simulating seismic activities for earthquake predictions or conducting hydrodynamic calculations for flood simulations, instead of managing computer systems. It lays the foundation for a reliable, fast IT infrastructure that can store, process and analyze massive amounts of data for various applications. HPC helps to overcome numerous computational barriers that conventional processors typically face.

1.4 Outline

The thesis is organized as follows: The Background chapter 2 details the necessary scientific and thesis-specific concepts, followed by a Related Work chapter 3 that surveys similar studies. The Methodology chapter 4 then describes the approach, detailing the procedures, steps, and tools used in the study. The Results 5 chapter presents and explains the findings of the analysis, supported by

graphical illustrations. Finally, the thesis concludes with a discussion 6 of the results, a summary of key insights, and suggestions for future work 8.

Chapter 2

Background

In this Background section, the general context and the specific tools used in the research will be explained and will provide an understanding of the components involved in the thesis. Firstly the miniHPC will be introduced and afterward, we will delve into the specifics of the Slurm workload manager and the Ganglia Monitoring Tool.

2.1 miniHPC

The University of Basel owns a high-performance computing system known as miniHPC [24]. This system provides the students working in parallel programming and a controlled experimental platform for researching in the HPC system field.

The architecture of the miniHPC is known as a cluster. An HPC cluster consists of a network of interconnected nodes. These nodes function independently as separate computers. In our case with miniHPC, it is organized in a fat-tree pattern.

The miniHPC system consists of four different types of nodes listed in table 2.1

Table 2.1: Specifications of miniHPC Nodes

Node Type	**CPU**	**No. of Nodes**
Intel Xeon E5-2640	2 CPUs per Node	22
Intel Xeon Phi Knights Landing (KNL) 7210	1 CPU per Node	4
Intel Xeon Gold 6258	2 CPUs per Node	1
AMD EPYC 7742	2 CPUs per Node	1

In figure 2.1 we have a graphical illustration of the architecture of the miniHPC. The 22 blue Nodes on the bottom 2.1 consist of each of two Intel Xeon CPUs. In the thesis, we will focus on examining the first 20 Nodes as these are the nodes processing the miniHPC user's job.

2.1.1 Energy Consumption

As powerful as the HPC systems are they also implicate a high amount of energy consumption. The energy in HPC is measured through embedded devices, and also Software tools [3]. It can be

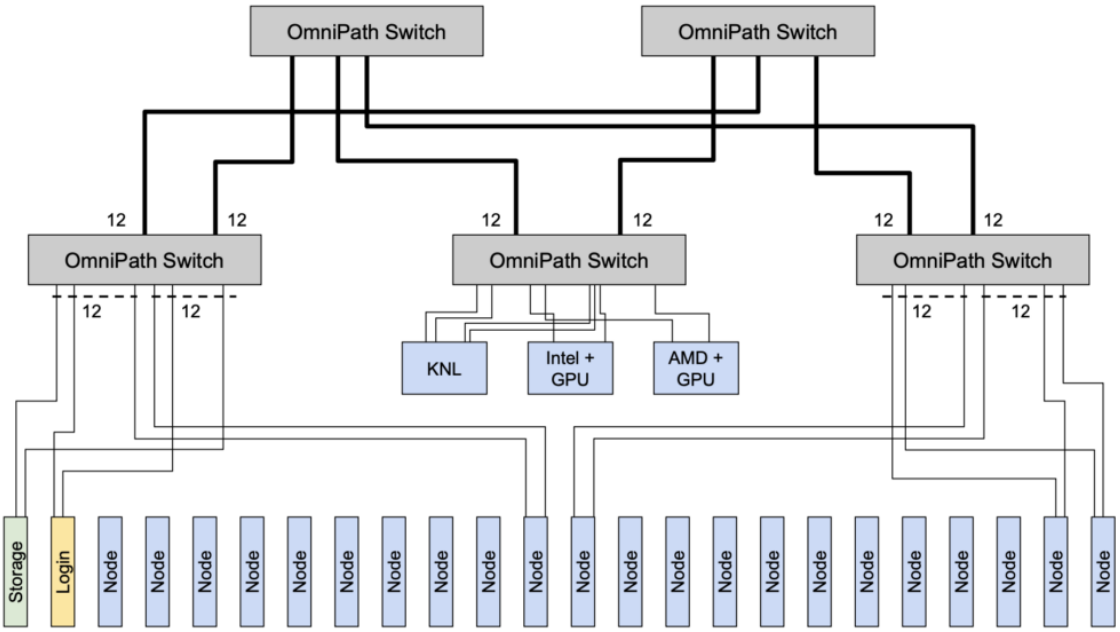

Figure 2.1: Graphical illustration of the miniHPC [24]

directly measured or not while integrating other software tools like Slurm to monitor and manage the energy consumption of HPCs. The most used techniques for energy saving in HPCs are [4]

- dynamic voltage and frequency scaling (DVFS) or
- just turning off idle nodes.

In the DVFS we have an adjustment of power and speed settings on a computing device's various processors or controller chips. The voltage and frequency of a processor are adjusted dynamically and are defined depending on the computational demands of the system. It reduces dynamically the clock speed of the CPU to reduce the amount of heat and energy produced through the CPU. CPU tuning is mostly used in cases where the CPU is overheating or while trying to save battery lifetime. With the CPU Frequency tuning it is possible to save a big amount of energy, while also having a drastic decrease in the performance of the specific CPUs[2].

The energy costs of an HPC system are extremely high and efficient energy use can help improve system reliability and lifespan by reducing the thermal stress on components that are caused through high CPU Frequency and redundant powering on of CPUs. The amount of energy consumed by HPC systems depends on Factors such as the number and type of processors, memory, storage devices, networking equipment, and cooling systems. For this, we will delve into our HPC study case that is provided by the university the miniHPC, and learn more about its components.

2.2 Slurm

Slurm is a highly scalable cluster management and job scheduling system for HPC systems. As these systems contain multiple computing nodes an efficient management of resources is needed in order to maximize the utilization and minimize job completion time. This also involves Job Scheduling based on priorities. The mini-PC uses Slurm as the workload manager. Slurm is used to collect the HPC job data we want to analyze.

Slurm has three main functions [20]:

- Allocating exclusive and/or non-exclusive access to users for some duration of time so they can perform work.
- Providing a framework for starting, executing, and monitoring work.
- Preventing conflicts for resources by managing a queue of pending work.

2.2.1 Slurm Architecture

Slurm uses a hierarchical Software Architecture consisting of three (controller) daemons[25]. Each daemon has its own workspace and there is a well-defined interface for the interaction between them.

1. Slurmctld (Slurm Controller Daemon):

   The central management daemon of Slurm is responsible for allocating jobs to all nodes with its scheduling. It also handles user requests to view or change the state of jobs or nodes. All running jobs continue to run even if the slurmctld daemon is stopped or crashed. By restarting the slurmctld daemon, it will attempt to continue operations as before with log files.

2. Slurmd (Slurm Node Daemon)

   This daemon runs on every node in the Slurm system and is responsible for managing tasks running on the node. The activities are like the starting-ending of the job and reporting the status of the job. Slurmd communicates with slurmctld while receiving commands and sending updates. If a slurmd process crashes or is stopped, all jobs on that node will be lost.

3. Slurmdbd (Slurm Database Daemon)

   The Slurmdbd is an optional daemon that can be used to record accounting information in a database. When used, it collects and stores information about job resource usage, user activities, and more. This data can be used for usage reporting, billing, and system analysis.

2.2.2 Client Commands

In Slurm the client commands are important for the users to be able to interact with the system or to request information about the system state. In figure 2.2 we see a list of commands such as scontrol and sacct. These two commands are especially important as they are used in the research.

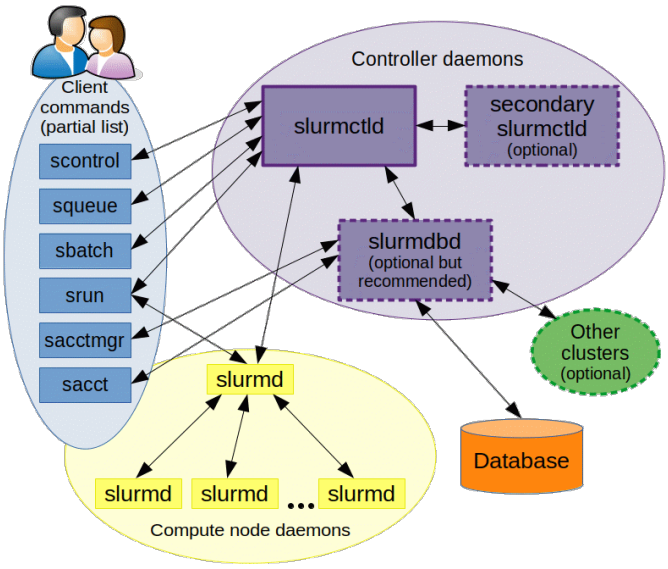

Figure 2.2: Desgin of architecture [19]

- **scontrol:**

  The scontrol command is used to view and modify Slurm configuration and state[14]. In my case, I would need to see the energy consumption of the nodes. This is possible with this command: `scontrol show node cl-node005`. The output is this and the interesting information for To display information about a specific node in the Slurm system, you can use the following command: `scontrol show node cl-node005 | grep -E 'CurrentWatts|AveWatts'`. This command will provide the energy consumption of the specific node.

```
[koca0000@dmi-cl-login ~]$ scontrol show node cl-node005 | grep
-E 'CurrentWatts|AveWatts'
   CurrentWatts=200 AveWatts=189
```

- **sacct:**

  The `sacct` command is reporting about the jobs that are running or completed while interacting with the `slurmdbd` daemon as it contains the accounting information. It is possible to get very detailed information about the jobs [15]. The useful information about jobs for the analysis is information about the start and end time of a job, the number of nodes used and the nodelist, the JobId or the owner of the job, and especially the consumed energy of a job. An example command would be the following one:

```
[koca0000@dmi-cl-login ~]$ sacct -a --starttime 2023-03-01T07:50:00 --endtime
2023-03-01T08:00:00 --format User,State,ConsumedEnergyRaw,NNodes,NodeList
```

```
     User      State ConsumedEnergyRaw   NNodes        NodeList
--------- ---------- ----------------- -------- ---------------
 user12    COMPLETED           2856646        2 cl-node[007-00+
 user3     COMPLETED           2408208        2 cl-node[013-01+
 user12    COMPLETED           2345056        2 cl-node[009-01+
```

2.2.3 Power saving mechanism

As I mentioned the `scontrol` command reports a node-level power consumption and is provided through the External Sensors plugin[18] [17] which is designed to interact with the hardware of the nodes to get energy consumption data. These sensors also provide information about the state of the node, such as the temperature or fan speed. There is also the Energy Accounting Plugin that is responsible for retrieving energy data of individual jobs. These plugins extend the core functionality and offer an interface to add new capabilities to the HPC system.

Therefore Slurm itself does not directly measure energy consumption, but there are configurations possible to combine Slurm with other software tools like plugins. Slurm provides itself a power-saving mechanism for powering down idle nodes[21]. After powering down they stay in their state until a job is assigned to them. This configuration is named in the Slurm documentary as the power-saving mechanism.

`Configuration:`

There is a Slurm-Configuration file where the `SuspendTime` can be defined. This will power down the nodes that are that long idle. The default value is -1, which means that per default the nodes will not be powered down with being idle. In the configuration file, there is also another important parameter: the `SuspendExcNodes`. This parameter defines which nodes in the HPC system will never be placed in power-saving mode.

In our miniHPC, the computing nodes for jobs are named as follows: node001, node002, ..., node020. During the period of the analysis, nodes 1-4 and nodes 17-20 were saved in the configuration file so as to never place in power in power-saving mode.

2.2.4 Job Steps

In Slurm Jobs can be complex entities consisting of multiple steps inside a specific job. A step could be a task in a parallel computation or an individual part of a job. After submitting a job to Slurm it gets an individual JobID and each individual step of the job gets an additional suffix as an identifier. So a jobId might look like `123` but their steps might look like `123.0` or `123.1`. So the number before the dot is the jobId and the number after the dot is the job-step ID. These steps are also called normal Steps. However there are also other steps like the batch step or the extern step [16].

```
JobID,User,State
1260826|user1|CANCELLED by 96648
1260826.batch||CANCELLED
1260826.0||CANCELLED
1271157|user2|COMPLETED
```

```
1271157.batch||COMPLETED
```

We have above an example of what this could look like. We see that only the JobID has the User Information. These are also the ones that we are going to use. In our analysis, we will not involve the steps because of several reasons.

Firstly, the collected energy data from Slurm for job steps is inconsistent, some steps are overlapping (e.g. the batch step) and include energy data from the following steps. Secondly, we are interested in reporting energy consumption for whole jobs and not only for parts of a job. The inconsistent nature of energy data for job steps makes it difficult to analyze the energy consumption for the whole job if we would split it into job steps.

Therefore in our job-level dataset, we looked just at the JobID as a whole with its steps together.

Slurm jobs can have several different states that reflect the current status or the final result of the job. Let's take a closer look at each of these states [14]:

- **Completed:** This state signifies that the job was successfully executed and completed without any issues. The task finished executing all the commands, and there were no errors during its run time.
- **Canceled:** This state indicates that the job was terminated before it could finish. A job might be canceled for several reasons, such as system issues, or it may have been manually canceled by a user or an administrator due to some factors like resource requirements or time constraints.
- **Timeout:** When a job exceeds the allocated time limit, it enters the timeout state. This usually means that the job took longer than expected or allowed, and hence the system automatically terminated it.
- **Failed:** If a job encounters an error during its execution that prevents it from completing successfully, it's considered to have failed. This could be due to issues in the script, software problems, or hardware failures.

2.3 Ganglia Monitoring

Ganglia is a widely used monitoring tool considered for HPC systems. It offers a web-based graphical interface where one have access to the past and present data of the cluster's performance [11]. Ganglia uses a hierarchical design in the architecture that is optimized for clusters and grids[12]. Ganglia gets its data from the **gmond**(Ganglia Monitoring Daemon) daemons running on each node for collecting node-level data, such as consumed Energy, CPU Usage, Memory Usage, and more. This data is alsocommunicated to other gmond daemons and to the gmetad (Ganglia Meta Daemon) daemon as graphically illustrated in figure 2.3.

For our thesis, Ganglia has a major impact on the result, because, unlike job-level data with sacct command, it is not possible to get or scontrol the data from the past. This work is taken away from Ganglia, as Ganglia provides in the Web application the dataset of each individual node with the data needed. Without Ganglia there would be written a script that would have run for a long period of time to obtain a node-level dataset. Thanks to Ganglia this work is completely omitted.

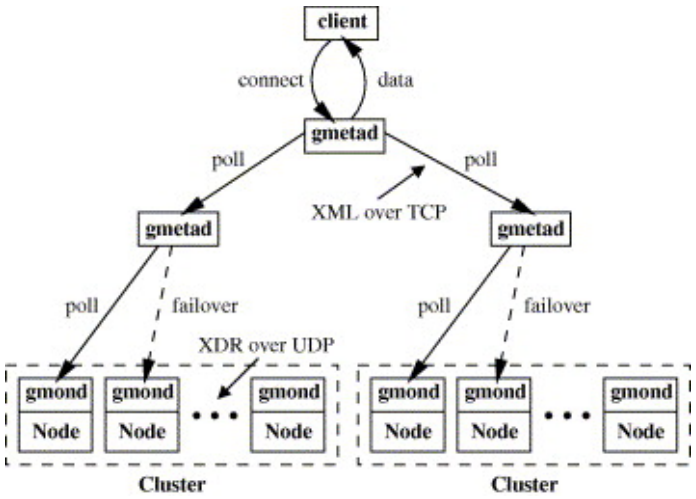

Figure 2.3: Desgin of architecture [12]

Chapter 3

Related Work

In this chapter, we will shortly see other similar works that have been done that are similar to our thesis.

3.1 HPC Job-Monitoring with SLURM, Prometheus, and Grafana

In the bachelor thesis `HPC Job-Monitoring with SLURM, Prometheus, and Grafana` Mr. Kunz researched the enhancement of monitoring frameworks in the miniHPC. He worked with Prometheus and Grafana to advance the observability of the miniHPC with monitoring and visualization tools. Mr. Kunz showed the way how the data could be collected and visualized to gain insight into the HPC system. The motive was to enhance the existing monitoring system and also to prevent potential system misuse, like unauthorized activities such as Bitcoin mining. The thesis described the limitation of the existing monitoring tool, such as the inability to visualize job-related metrics. The author proposed a new monitoring setup with its installation and configuration. The results showed improvements in being able to have observability in system.[10]

3.2 Analysis of Historical HPC Job Data

In her bachelor thesis, Mrs. Keller presents an analysis of historical job data. The aim of the thesis is to optimize the miniHPC at the University of Basel by understanding job-specific metrics and their usage. Mrs. Keller shows that efficient storage of job data can help to develop models for identifying and terminating misuse processes, such s Bitcoin mining. The relevance of visualization tools for the system would prevent potential bottlenecks in system efficiency and would produce the detection of job anomalies. Mrs. Keller configures the new monitoring frameworks as in Mr. Kunz's thesis and facilitates users to query over 20 metrics of running jobs and also provides opportunities for future developments. The author also points out the need for further research to understand the anomalous behavior of jobs and to improve anomaly detection. The suggestion of the author is to build models using historical data to identify outliers at the runtime[9].

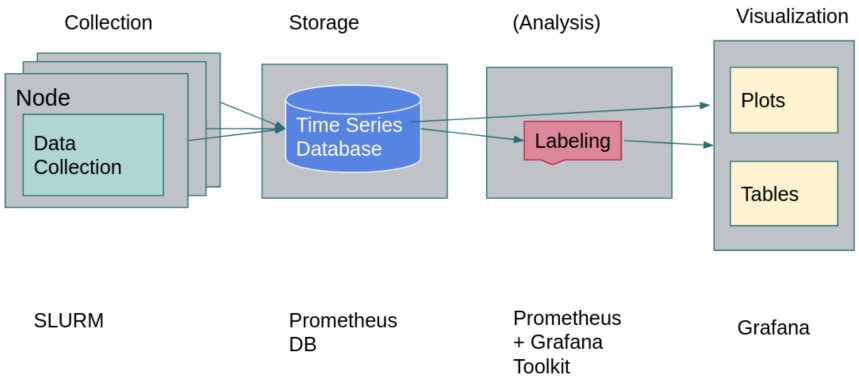

Figure 3.1: Monitoring Framework with the differentiation between collection, storage, analysis, and visualization modules

3.3 Energy-Aware HPC

The paper describes various system and device types, such as single devices, clusters, grids, and clouds, as well as CPUs, GPUs, multiprocessors, and hybrid systems. There were many optimization goals reviewed, such as energy consumption or temperature. The author looked to control methods such as DVFS or power capping which were essential information for the thesis. The paper also presented an analysis of existing tools and APIs for energy management and simulation tools for energy consumption in modern HPC systems with programming examples. There were open areas and challenges identified regarding methods and tools for energy-aware HPC usage, such as the need for a uniform power-aware API or the development of performance-energy models. The article provided an overview of energy-aware HPC highlighting the need for improvements in this research area.[4]

3.4 Energy management framework for HPC

The focus of Corbala, the author of the paper, lies in EARL's energy optimization. EARL is a runtime library providing energy optimization and control. The plugin dynamically selects the optimal CPU Frequency based on configured policies and other application characteristics. The framework employs a strategy to be highly scalable by allowing each node to operate independently. A novel algorithm, DynAIS, is integrated within EARL, enabling it to detect the iterative regions making the framework's optimization dynamic and transparent. Corbalan's work has demonstrated the ability to dynamically identify application internal structures with very little overhead. EARL's integration with DynAIS offers a robust runtime system for energy optimization and control. The framework manages energy consumption by detecting situations that need fixing, broadcasting the warning level to nodes, and allowing nodes to react and redirect the situation locally.[3]

Chapter 4

Methodology

In this chapter, there will be explained the process of the analysis by getting the datasets and the necessary steps done to get the analysis result.

4.1 Solution Approach

4.1.1 Overview

The goal of the thesis was to use the data of the miniHPC to find out information about energy consumption and potential energy-saving strategies.

4.1.2 Idea

In section 2.2.3 we saw that Slurm provides a power-saving mechanism for powering down idle nodes to be able to save energy. The miniHPC has currently set 8 of the 20 nodes that are processing the jobs of the users to power-saving mode as illustrated in table 4.1. After some idle time of these nodes, the Slurm will power down these nodes. The approach of this thesis is to analyze the current energy consumption of the miniHPC and find statistical improvements with the data available.

The current `SuspendTime` that is set in the Slurmconfiguration file defines for the power-saving nodes the time when the nodes will be powered down and are to 7200 seconds (2 hours). To set this `SuspendTime` to a very low number would cause a lot of unnecessary energy consumption due to unnecessary powering up and down processes. The idea would be to power down nodes then when possibly no jobs will appear in a long time period. In this case, it could be that the node will be powered up and down several times in a very short time period.

4.2 Design of Observations

In table 4.1 we have illustrated the entire analysis environment with the different data sources and the specifications of the nodes. The table shows the most important information about the analysis environment.

Table 4.1: Design of Observations

Nodes	Data Sources	Metrics	CPU Model	Count	Node Names
Not Powered-saving Nodes	Ganliga (Node-Level)	Watt	Intel Xeon E5-2640	8	001, ..., 004
	SLURM (Job-Level)	Joule			017, ..., 020
Powered-saving Nodes	Ganliga (Node-Level)	Watt	Intel Xeon E5-2640	12	005, ..., 009
	SLURM (Job-Level)	Joule			010, ..., 016

All analyzed nodes have the same CPU Model. The key difference in the analysis is the nodes that are configured as Power-saving Nodes. In table 4.2 a detailed Version of the node information such as the memory size and the CPU Speed.

Table 4.2: Node Information

CPU Model	CPU Count	CPU Speed (GHz)	Cores	Threads	RAM (GB)	Cache
Intel Xeon E5-2640	2	2.4	10	20	64	L3: 25MB

4.3 Energy Metrics

As we listed in the table 4.1 the data sources are in two different metrics. The node-level data gained from Ganglia consists of Energy Data in Watt. In the job-level dataset, we have the energy consumption of the jobs in Joule. To be able to work with these data we have to understand the difference between the metrics and calculate the energy consumption in the datasets.

4.3.1 Joule

A Joule is a unit of energy. One Joule represents the energy consumed when applying a force of one newton through a distance of one meter: 1 J = 1 Nm.

In the context of a computing system, energy usage represents the total amount of energy consumed from a job.

4.3.2 Watt

The Watt-unity describes an amount of energy per unit of time and is known as the power[7]. One watt is equivalent to one Joule per second. The power written in Watt-unit is written as W = Wattseconds: 1 W = 1 J/s.

In our thesis, the node-level energy consumption, represented as Watt in the dataset, shows the power that is transferred per node over a specific time period. In table 4.3 we see an illustration of it.

Job ID	Start Time	End Time	Duration (s)	Energy (J)	Power (W)
1234	2023-06-21 09:00:00	2023-06-21 09:00:30	30	3000 J	100 W
1235	2023-06-21 09:01:00	2023-06-21 09:01:10	10	500 J	50 W

Table 4.3: Example of job records in Slurm with energy consumption

To manage and work on the datasets it is necessary to understand these metrics. The job-level dataset gained with Slurm contains job entries with the energy data consumed per each job and the node-level dataset gained with Ganglia contains the energy consumed per these nodes over a period of time in Watt metric.

Depending on the scenario energy consumption can be expressed in different metrics. In the context of the HPC system the Watt (W) and the Joule (J) are the necessary units. To be able to compare different scenarios, it could be essential to mention the conversions between different units of energy. Here is a table 4.4 summarizing some common conversions among Watt-seconds (W), Watt-hours (Wh), Kilowatt-hours (kWh), and Megawatt-hours (MWh).

Unit	**Conversion to Watt-seconds (W)**
1 Watt-second (W)	1
1 Watt-hour (Wh)	3600 (W)
1 Kilowatt-hour (kWh)	3.6e+6 (W)
1 Megawatt-hour (MWh)	3.6e+9 (W)

Table 4.4: Energy Unit Conversions

4.3.3 Startup Process

Powering nodes up and down can be very energy-intensive for several reasons such as the start process also named as inrush current where the capacitors and inductors need to reach their operating charge [22]. Processes like the POST (Power-On Self Test) or loading of the operating system with its data loadings into RAM or I/O operations consume a lot of energy [13].

To minimize the powering up and down of the nodes while also having the nodes powered down in the job-busy periods the analysis of the job data is necessary. With the analysis results it is possible to see patterns in job scheduling and propose ways for possible energy-saving strategies.

4.3.4 Datasets

There are two different datasets used in the analysis. They consist of different data information and differ in structure. The node-level dataset from Ganglia consists of the power data of each node presented in a specific time period. This data represents the energy consumption from the node perspective without involving the fact that some node consumption is there without knowing if they are idle.

Furthermore, there is a job-level dataset in the analysis that is obtained from Slurm's `sacct` command containing all the jobs with their start and end time or number and name of the nodes involved in this specific task or the number of the entire consumed energy from the specific jobs.

To have the Watt value without knowing the duration of a job would not give representative information about the energy consumption from the job, as a job that consumes a specific amount of power (in wWtts) over hours would have the same power value as a job with the same power consumption but with a much less duration as we can see an example in the table 4.5 demonstrated.

In the node-level data, where the data represents the energy consumption over time the Power metric makes sense because we want to get information about the nodes' overall energy consumption over time. Therefore having a metric that is related to time is useful.

Job ID	Start	End	Duration (s)	Consumedenergy (J)	Power (W)
1234.0	09:00:00	09:00:10	10	500 J	50 W
1234.1	09:01:00	09:02:00	60	3000 J	50 W

Table 4.5: Example of job records in Slurm with energy consumption

4.3.5 Data Correlation

The idea is to correlate the two different datasets, where we then could see the differences in Energy consumption to be able to say which amount of the node-level energy was effectively used for the processing job and which energy was used by nodes during idle periods.

But to be able to correlate and compare the two datasets they need to have the same data structure. The main work of restructuring the dataset occurs in the job-level data where the details are explained in section 4.6.

With these steps, we are able to correlate two same structured datasets and able to plot these data in meaningful and valuable charts and visualizations where we can spot patterns where we find opportunities to improve energy consumption.

4.4 Analysis Environment

This section outlines the tools utilized in this study for data manipulation, analysis, and visualization.

4.4.1 Python and Plotting Tools

The programming language Python was used throughout the study due to its data analysis libraries. With the Matplotlib library, the visualization and representation of data were possible in an efficient way. The library provides a powerful platform for plotting graphs. Data obtained from Ganglia and Slurm in CSV format could be straightforwardly manipulated using Python.

4.4.2 Libraries

The following Python libraries were critical for data manipulation and analysis in this study:

- Pandas:

  Pandas offers data structures and functions essential for manipulating datasets. It excels when working with labeled data, such as CSV files, enabling efficient cleaning, filtering, and analysis of data.

- Numpy:

  Numpy was used for handling large, multi-dimensional arrays of numerical data, facilitating computations.

- CSV:

  The CSV library simplifies the process of reading and writing data and from CSV files.

- Matplotlib:

  Matplotlib was used for creating 2D graphics and charts, making data visualization straightforward and effective. The Matplotlib library was used throughout the thesis process for plotting. It has proven to be efficient for this purpose. By the end of the thesis, we utilized a data visualization tool named Flourish [6].

- DateTime:

  The DateTime module allows easy manipulations of dates and times, adding crucial functionality for this study.

These selected libraries provided an efficient method for manipulating and analyzing datasets.

4.5 Node-Level Dataset

This section, focusing on the node-level dataset, will be concise as the dataset already had the necessary structure and content for analysis. Still, we will detail the process used to gather it.

While Slurm provides a command `scontrol show node=<node-name>` to obtain the instantaneous state of specific nodes, including the `ConsumedEnergyWatts` parameter, it does not offer historical energy consumption data. This type of data needs continuous collection over time, necessitating storage.

This is where the Ganglia Monitoring tool comes into play, serving as an interface that collects this data over time (see section 2.3). Ganglia is designed to record statistics for HPC systems, including nodes' power usage.

4.5.1 Web Interface Ganglia

The Ganglia web interface offers a graphical illustration of the data representing one or multiple nodes. While these graphics were not directly used in the analysis due to their limited personalization, they were still highly beneficial to the thesis. They provided insights into the overall behavior of the nodes' power consumption showing patterns in the data.

They helped in the beginning stages of the thesis by making assumptions and observing differences between the nodes. The web interface allows possible aggregation of the power data from several nodes and offers a comprehensive view of energy consumption. The power-saving nodes were visible in the plots, as in idle periods the nodes were not present in the plots.

4.5.2 Characteristic

Ganglia's dataset is available in different time periods and can be presented with varying granularities, such as last hour, day, week, month, or year. This flexibility aids in pattern identification and problem detection.

Ganglia employs a Round-Robin Database (RRD) to store the node-level data. The RRD is a system that collects and stores time-specific data and generates graphs for data visualization. As the timestamp period lengthens in higher-period datasets, the granularity correspondingly decreases. The RRD supports the consolidation of high-granularity data into lower-granularity data [5]. This

process involves calculating the average (default), minimum, maximum, or last value over a given period. The configuration for this aggregation method is saved in the `gmetad.conf` file, as the gmetad daemon is responsible for data collection (refer to figure 2.3).

4.6 Job-Level Dataset

As mentioned in section 2.2.2, the sacct command was used to obtain the job-level dataset from Slurm. This section will describe the several steps taken to process this job-level dataset.

The Slurm Database, or slurmdbd daemon, stores accounting data, such as resources allocated to users, user activity logs, jobs, and job steps. This is detailed in the Slurm architecture section 2.2.1. The slurmdbd daemon interfaces with a database system to store and retrieve this data.

Slurm can provide real-time status data such as node and job status. The job-level dataset includes this information:

- JobID: A unique identifier for each job.
- User: The owner of the job.
- Start: The start time of the job.
- End: The end time of the job.
- State: The state of the job if it is completed, failed, or canceled
- ConsumedEnergyRaw: The energy consumed for that job, in Joules. This is raw data, not formatted into larger units like megajoules.
- NNodes: The number of nodes involved in the job.
- NodeList: A list of nodes presented in a regular expression format, like node[001-005].

This list defines the structure of the job-level data obtained with the `sacct` command. To facilitate further analysis of this data, several processing steps will be undertaken, as outlined in the following sections.

4.6.1 Retrieval

The retrieval of the data started as possible and the data :

```
    [koca0000@dmi-cl-login ~]$ sacct -a --starttime 2023-02-00 --endtime 2023-06-01
--format JobID,User,State,Start,End,ConsumedEnergyRaw,NNodes,NodeList --parsable2 | sed 's/|/,/g'

    JobID,User,State,Start,End,ElapsedRaw,ConsumedEnergyRaw,NNodes,NodeList
    2109092,username,COMPLETED,2023-05-11T23:10:31,2023-05-12T00:08:35,3484,0,1,cl-node027
    2109092.batch,,COMPLETED,2023-05-11T23:10:31,2023-05-12T00:08:35,3484,0,1,cl-node027
    2109092.0,,COMPLETED,2023-05-11T23:10:36,2023-05-12T00:08:35,3479,0,1,cl-node027
```

The period of analysis was set from the beginning of February to the end of May 2023. In January there were major changes in the configuration file of Slurm, such as adding nodes to the power-saving list or changing the `SuspendTime`. To have a consistent dataset the period from February to the start of May was chosen. The periods cover main University periods such as the end of a semester, the semester break, and the period during a semester. This range in period will give us the opportunity to see patterns overall in a University cycle and cover all possible scenarios.

The sacct commands mentioned above consist of some parameters. The `-a` flag specifies that I want to accounting information from all users. After the flag, we limited the job information between the corresponding date frame. The `--format` specifies the format of the output with the specific columns listed afterward, such as JobID, and User. The `--parsable2` is an additional formatting option that separates all entries with a separator character. In the end, I used the stream editor to replace the separator with a ',' character as I faced problems with the previous separator in the process.

4.6.2 Cleaning

As mentioned in the Job Steps section 2.2.4 we will not consider in the analysis the job steps. For that, we deleted these entries with a script from the datasets.

```
# Filter out entries with a dot in the JobID
df = df[~df['JobID'].str.contains('\.')]
```

The script went through all JobID entries and deleted the entries that contained a point in the name, representing. The JobID represents with a suffix that it is a step of a Job. This way we deleted the substep information in the dataset, representing all jobs at one time. This step also was important due to the analysis. It would not have been possible to show the energy consumption in total with overlapping energy consumption data.

4.6.3 Modify

The goal of the modification of the job-level dataset is to have the same structured dataset as the node-level dataset from Ganglia. Therefore we would be able to compare these two datasets and get insightful plots.

In the job-level dataset, there were some additional columns needed for being able to correlate the dataset with the node-level dataset. The `consumedEnergyRaw` column consisted of the energy consumed per that job completely. Hence if there were multiple nodes dedicated to a job, the energy data involved energy consumption together from all nodes. Hence to transform the data in a node-based dataset we needed to split the energy data across the nodes to get a node-specific data consumption on that job.

The next added column was the duration of the job. We added a duration column where the difference between the start and end timestamps was calculated. This was needed for the next step.

There was also used a second script where each user contained in the dataset gets a respected alias name as ¡user¿¡Identifier Number¿ to be able to keep the privacy of the users.

We created parallel another dataset that will be filled up based on the modified job-level dataset. The newly created dataset has in the first column the timestamp, the same period as the dataset from Ganglia. The next columns represented the nodes starting from the first node 001 till the last node 020.

4.6.4 Populate

After we got each job their energy consumption per node that was involved in the job process, we needed to add this energy add this data to the newly created dataset. There were two cases in the script that was handled.

If the Job's time period fits within a timestamp in the one timestamp the consumed energy data was added up into the specific nodes that were involved in the right timestamp.

If the job spanned across multiple timestamps the energy consumption of the nodes that were involved in that job processed split the energy consumption proportionally related to their duration in these timestamps. In this case, it was not considered if the job could have consumed more energy at the beginning of a job or at the end for instance. We do not have this specific data.

There could have been involved in this case the steps of the jobs to see in which period of time the job consumed more energy. But this approach would have involved the analysis of inconsistencies due to incorrect energy data information. The Slurm is not able to provide consistent energy consumption across multiple steps yet.

With these steps, we would have a dataset with the nodes and their consumed energy based on the jobs placed in the right timestamp. The last step was to populate this energy consumption data that is in Joule to the same structured dataset where the data converted from energy to power with the transformation rule explained in section 4.3.

With this approach, we gained a dataset consisting of nodes with their energy eonsumption in Watt-metric for the respective timestamps.

Chapter 5

Results

First, we analyze the overall plot of the data. This initial step is necessary to obtain comprehensive information and understand the characteristics of the dataset.

5.1 Overall analysis know more information about the dataset

As the energy consumption depends also on the number of jobs, we will frequently consider the plots. This is because high energy consumption with a low number of jobs can have the same priority for powering on as nodes with a large number of jobs. We begin with a plot where we chart the Energy Consumption together with the number of Jobs allowing us to observe the periodical trend of high demand on the miniHPC and the periods of most energy consumption.

In the plot 5.1, we see that the end of March had an extraordinarily high number of jobs on two days, but this did not correspond with high energy consumption compared to other periods. On these two specific days, the system processed a combined total of approximately 1.2 million jobs, initiated by a single user. Surprisingly, these jobs didn't consume any energy due to the extremely short duration of each job, with an average rate of about 16.1 jobs executed per second over a 21-hour period. This unusual scenario, while significantly affecting our visualizations, is atypical in real-world situations, where a user initiating such a large number of jobs is highly irregular.

Our goal is to identify patterns in the energy consumption of the jobs. We will therefore for our analysis in this thesis filter out those jobs that did not consume any energy, such as these who have a zero value in the `ConsumedEnergyRaw` column.

We will be able to exclude these special cases and focus on the representative set of jobs that consume energy, allowing for more meaningful analysis. This step will also provide a more representative plot to better visualize the relationship between job number and energy consumption as we can see in the figure 5.2.

Now we can see periods where energy consumption and the number of jobs correlate during a semester. Starting from mid-April, we notice that the miniHPC was mostly used in the second half of the semester and during the semester-end holiday. Most energy consumption happened before the semester started, indicating that these jobs were not lecture-based, but possibly personalized jobs for theses or research. The two main rushes in energy consumption are in the second half of the semester and during the end of the semester break.

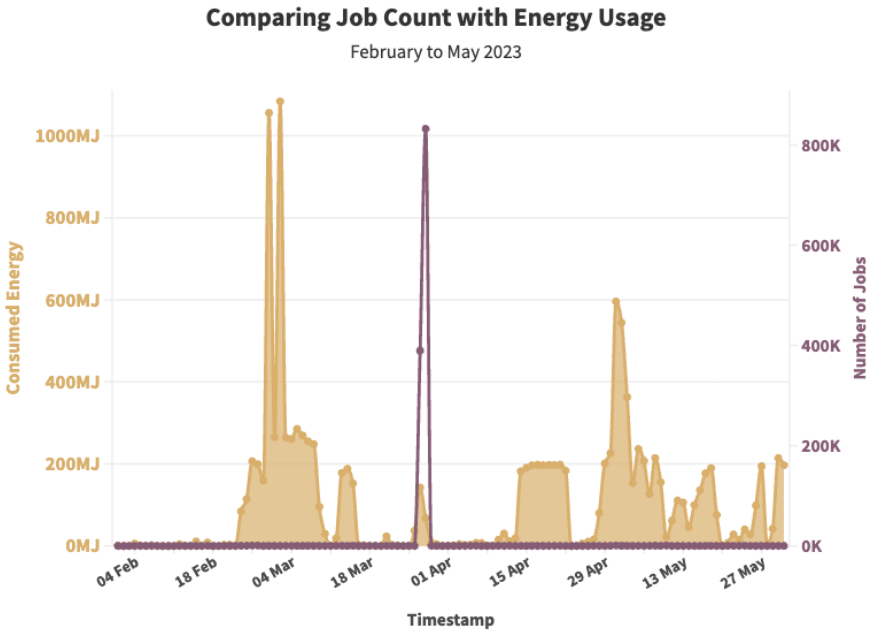

Figure 5.1: Relation between Jobs and their Energy Consumption containing all Jobs.

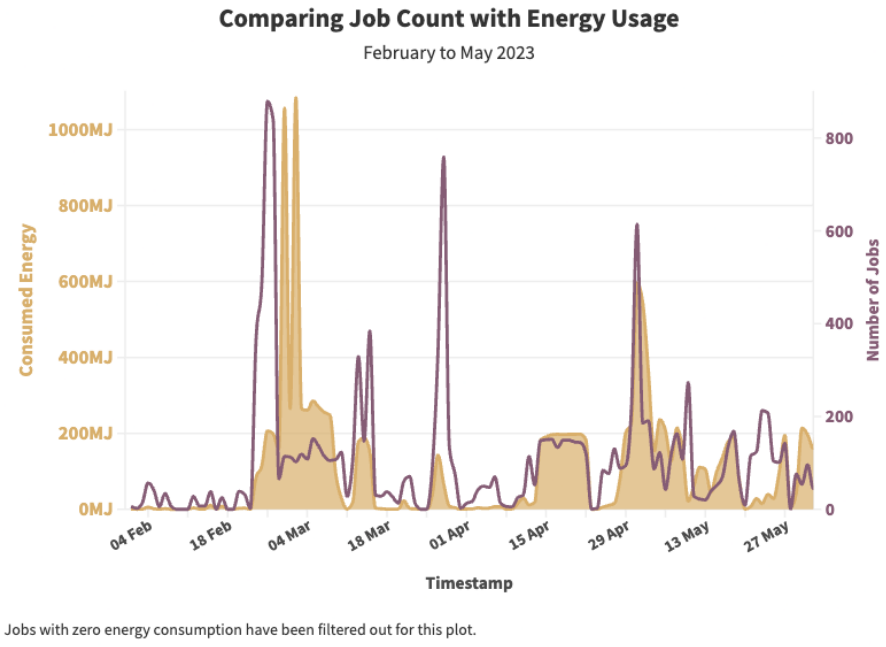

Figure 5.2: Relation between Count and Consumption of energy consumed Jobs.

5.1.1 User-specific information

If we look into the usage of the miniHPC we will find that in the period from February to May, there were two main energy consumers on the miniHPC 5.3. The jobs of these two users differ as User2 has more energy-intensive jobs compared to User1. For further analysis of job energy consumption in this period of time, the jobs of these two users could be investigated more as they represent most of the energy consumption.

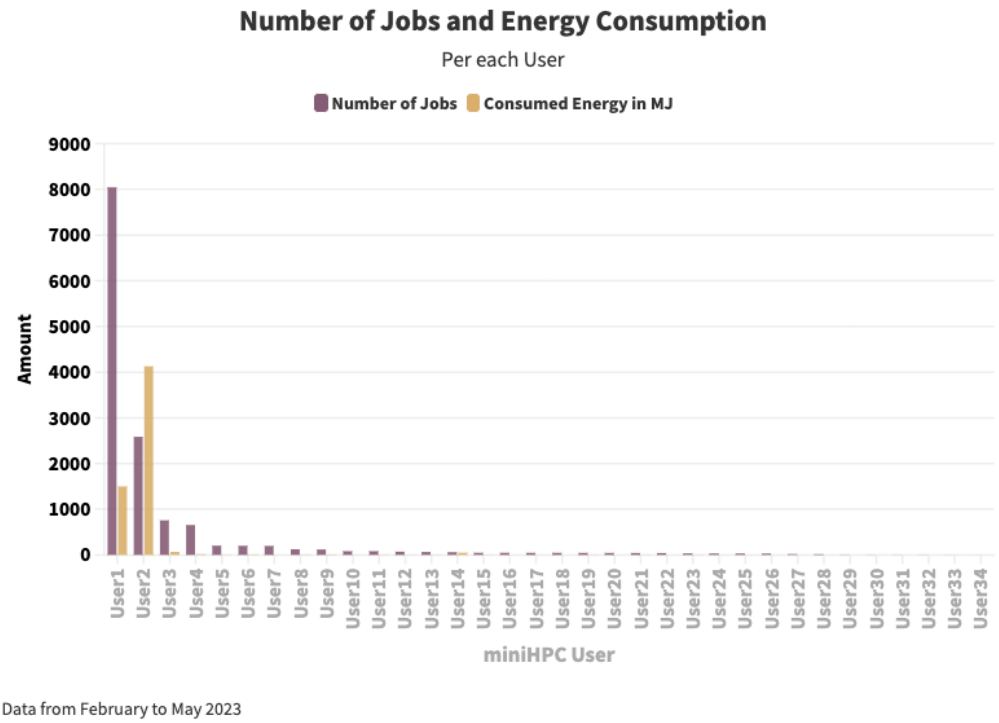

Figure 5.3: Job distribution over Users related to Energy Consumption.

5.2 Jobs' Energy Consumption

The second question of the thesis is the amount of energy the jobs use. For that will analyze the jobs' energy consumption, which node received the most jobs, and the relation between energy consumption and duration.

5.2.1 Number of Jobs

In figure 5.4 we see the total amount of jobs allocated to nodes that also consumed energy from February to May 2023. We notice that the first nine nodes handle a high number of energy-intensive jobs and then drop down to the next nodes. The nodes 18 to 20 had almost no jobs allocated during

the entire period. This plot shows us that most of the jobs are carried by the first nine nodes. We also notice that although the number of jobs from nodes 12 to 16 is not so high the consumed energy of these jobs consumed is comparable to the first nodes.

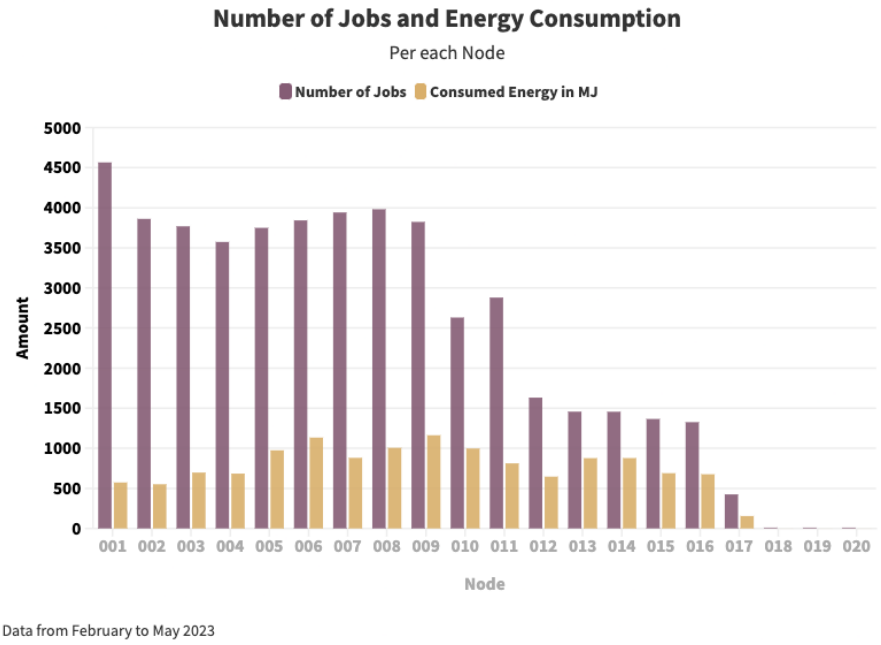

Figure 5.4: Node-Level Visualization of Job Count and Consumption.

5.2.2 Number of Nodes used per Job

Jobs can be allocated to several nodes. It's also important to note the number of nodes most jobs use to identify a pattern. In figure 5.5, we see that most jobs used either single, two, or ten nodes. The nodes listed in the table are all included, but we do not see the bars as the number of jobs requiring such a number of nodes are very low. We observe over the period of time that the most used number of nodes for jobs was two, followed by ten nodes and then a single node. From the plot, we can deduce that the jobs with ten nodes have more high-intensive tasks than the single or dual-node jobs.

5.2.3 Job Duration and Energy Consumption

Before answering how much energy a job consumes, it's worth examining the relationship between the duration and consumption of a job and delving into the correlation.

In the scatter plot 5.6, we see specific trend lines where the jobs can be divided into two groups: those that use a GPU Node and those processed by our case study's Intel Xeon CPU node (listed in 2.1). But the average trendline of the job consumption and duration correlation is about 1.1 MJ per hour.

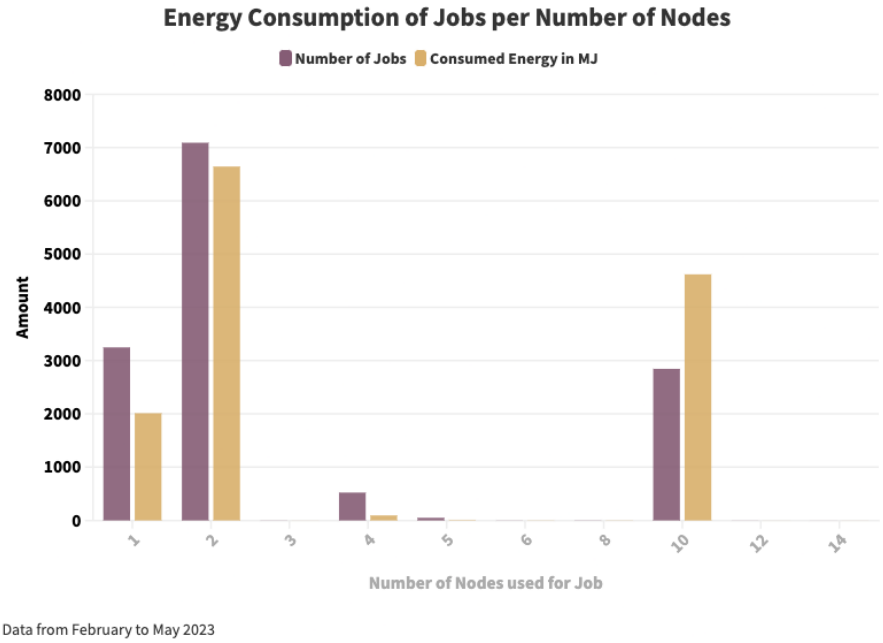

Figure 5.5: Distribution of Consumption and Count by Number of Nodes Utilized

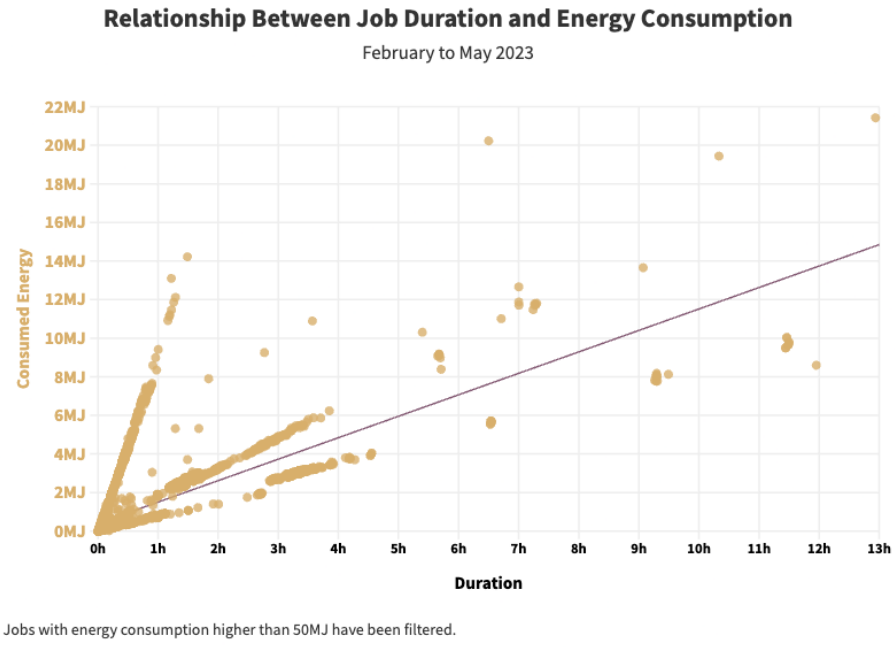

Figure 5.6: Energy Consumption per Job vs. Job Duration less than 50MJ.

We filtered the outliers from the plot to show them specifically in another scatter plot 5.7. As these are just a few jobs with extraordinary energy consumption, we divided the plot into these two figures and are able to see the correlation between job duration and energy consumption.

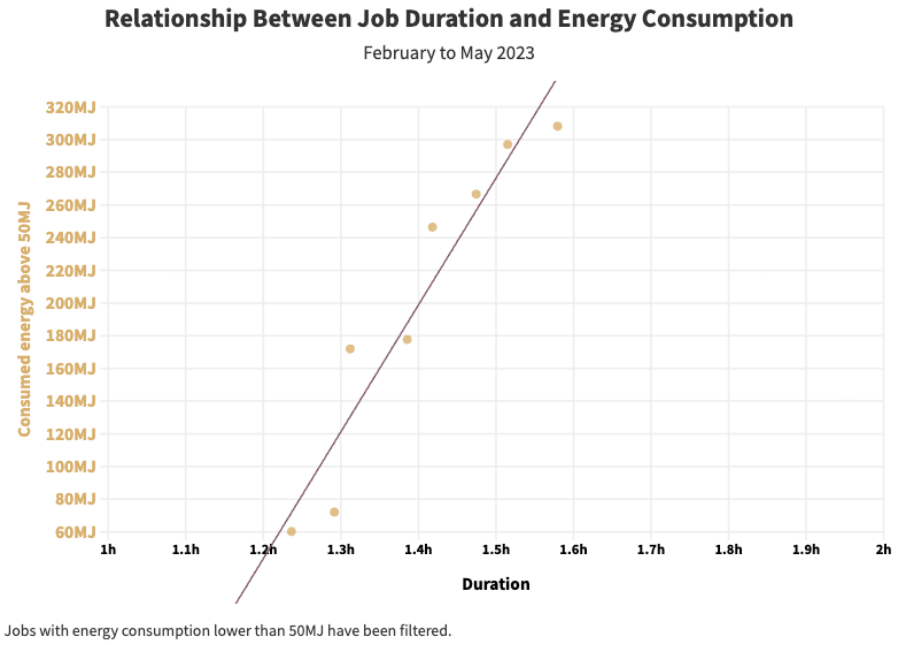

Figure 5.7: Consumption vs. Duration of Jobs with high Energy Consumption.

Monitoring energy consumption of GPU nodes will be mentioned in the future work chapter 8.5.

5.2.4 Distribution of Job's Energy Consumption

With this analysis, we can quantify the amount of energy a job consumes. We will illustrate this with a distribution plot 5.8, where we have the bins in gaps of 0.5MJ on the x-axis.

Most occurrences happen in the first bin that represents jobs consuming fewer than 0.5MJ, decreasing rapidly in the next bin. We see that after 4MJ, there are no more visible occurrences. The average energy consumption of a job is about 0.23 MJ.

5.3 Idle vs Busy Mode

The Ganglia dataset provides us with the average energy consumption of nodes, with each node exhibiting varying levels of energy consumption. A node can either be idle, implying no active job to process, or it can be engaged with a task, thus consuming more energy. With only the node-level dataset at hand, determining the average energy consumption during idle or busy states is unachievable. However, the job-level dataset offers insight into periods when nodes are actively

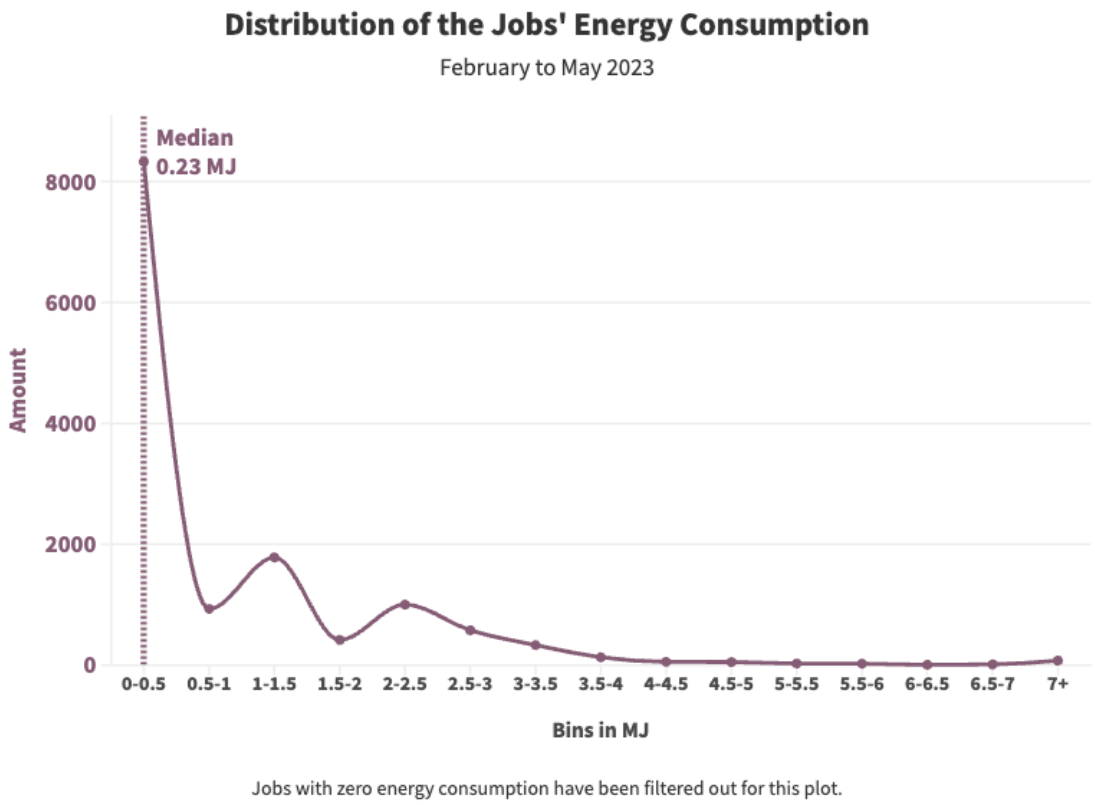

Figure 5.8: Distribution of the Jobs regarding Energy Consumption.

processing jobs and when they're idle. By integrating both datasets, we can segment the energy consumption data of all nodes into two distinct categories: idle periods, identified when no jobs are allocated to a specific node, and busy periods, established when a job is allocated to the nodes at a specific timestamp. This segmentation allows us to compute the average energy consumption of the node-level dataset based on whether the nodes are idle or busy.

In this thesis, we don't distinguish between high or low-energy intensive jobs, though this factor will be discussed in the future work chapter 8.

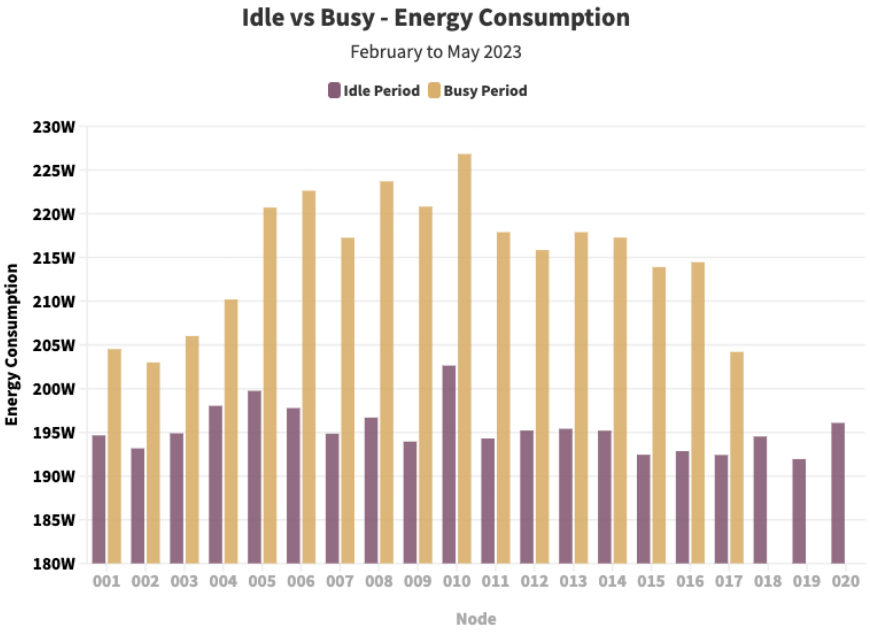

Figure 5.9: A comparison of average energy consumption for each node, distinguishing between idle and busy operational states.

Figure 5.9 presents this analysis visually. We can infer that, in general, the energy consumption during idle periods is relatively consistent across all nodes, averaging about 193 Watts. We observe that nodes 18 to 20 do not register any energy consumption during busy periods. This absence is because these nodes did not partake in any jobs during the period from February to May. Nodes 5 to 16 exhibit higher energy consumption during busy periods compared to the power-saving nodes 1-5 and 17. This finding is unexpected as it suggests that power cycling—powering up and down—might induce higher energy consumption during busy periods, likely due to energy-intensive startup processes as mentioned in subsection 4.3.3. The average energy consumption of nodes during idle periods is about 195 Watts whereas the average energy consumption during busy periods is 215 Watts.

5.4 Correlated Jobs Plot

With the node-level consumption data at hand, we can accumulate the energy consumption values of all nodes at each timestamp. Doing so provides the total node-level energy consumption for all 20 nodes. Similarly, using the job-level dataset, we can compile the energy consumption for each timestamp.

Plotting the correlation of these two total energy consumption datasets results in Figure 5.10. As previously observed in Figure 5.2, the job-level energy consumption is nearly zero at the beginning of February. A crucial observation is that the difference between these two datasets represents periods when nodes were idle and not processing jobs. The objective of energy-saving strategies would be to minimize this gap, thereby maximizing the utilization of node-level energy for jobs.

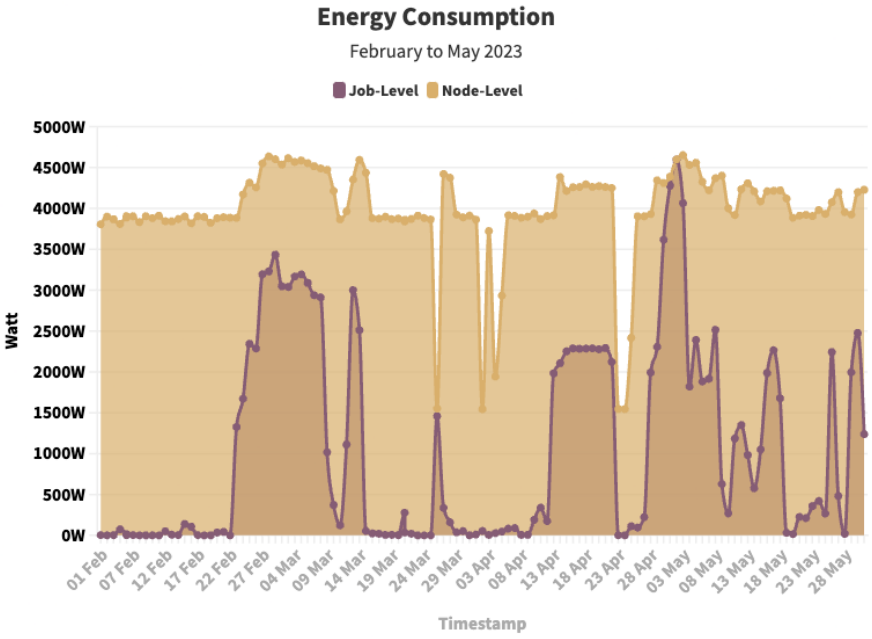

Figure 5.10: Correlation of Job-Level and Node-Level Dataset.

This correlation reveals the consumed energy without processing jobs. Figure 5.10 indicates the proportion of node-level consumption not allocated to jobs, i.e., energy potentially wasted during idle periods. For instance, in early February, the energy consumption difference is vast due to a minimal number of running jobs.

By consolidating the energy consumption data from both datasets of each timestamp, we obtain the following in table 5.1:

As seen in the table, the node-level dataset had an energy consumption of about `40.8 GJ` over the period from February to May, while the job-level dataset had an energy consumption of around `12.8 GJ`. The job-level dataset represents 31.4% of the total node-level consumption, suggesting that approximately $\frac{2}{3}$ of the nodes' energy consumption during idle periods could potentially be saved with power-saving mechanisms.

Dataset	Average Power	Energy Consumption Feb to May 2023
Node-Level	3946 W	40.8 GJ
Job-Level	1240 W	12.8 GJ

Table 5.1: Calculating the consumed Energy in Joule.

However, this correlation 5.10 does not identify which nodes significantly contribute to the energy difference. This identification could guide us in understanding why certain nodes consume excessive energy and how to mitigate this to reduce overall energy consumption.

A possible graphical representation could involve delineating the difference in energy consumption per node, displayed as a stacked chart. However, such a visualization may not effectively highlight the outliers and high energy-consuming nodes due to the complexity of representing all 20 nodes in a stacked format.

The data will be instead present in the column chart shown in Figure 5.11.

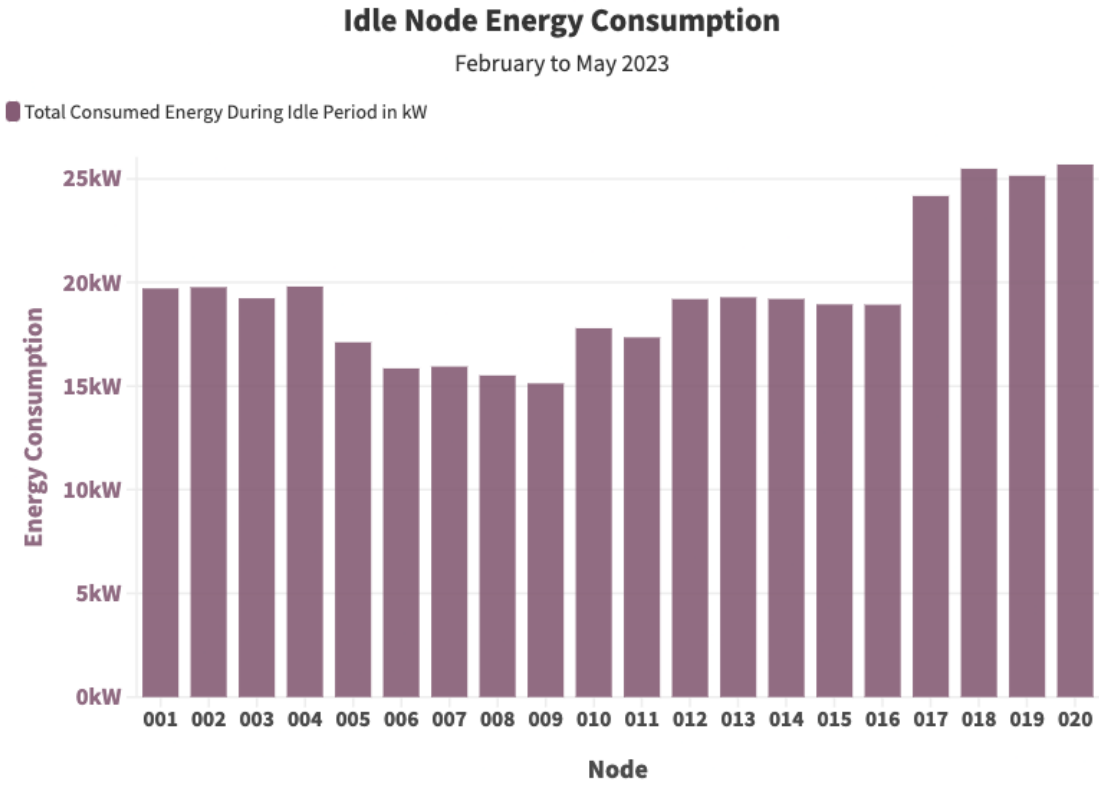

Figure 5.11: Energy consumed per node during idle periods (difference between Datasets).

Figure 5.11 displays the node-level energy consumption of the difference previously seen in Figure 5.10. The aim here is to identify which nodes were primarily responsible for the energy consumption

during idle periods, with energy consumption summed up as idle period energy consumption from February to May 2023.

Compared to Node 20, Node 9, which is in power-saving mode and powers down during idle periods after 2 hours, has 40% lower energy consumption. Conversely, Node 20 is not in power-saving mode. This visualization underscores that Nodes 17-20 consumed substantial energy during idle periods, which aligns with the fact that these nodes are not in power-saving mode and do not process many jobs as seen in Figure 5.4. In fact, Nodes 18-20 did not process any jobs during the period from February to May. Additionally, we observe that power-saving Nodes 5-9 consumed about 4kW less energy over the period compared to Nodes 1-4, although they have higher energy-intensive and more processed jobs.

5.5 Parameter

In figure 5.12 we have a representation of the nodes where the average arrival time of the jobs is represented. The key consideration is how to set the parameter for `SuspendTime`, which determines when the nodes should be powered down during idle periods. The calculation was done by calculating the time between two jobs for each node.

If we compare the visualization with the number of jobs of nodes 5.4 we would realize that the more number of jobs a node has the lower the average idle time of the node. If we consider that most jobs are submitted in mass and processed each other we could imagine that this would result in an average lower waiting time.

During the calculation, many edge cases such as simultaneous jobs and overlapping jobs were encountered. The calculation handled these scenarios by searching for the times when a node was idle and calculating the waiting time for the next job for each idle period. As we are not actually interested in the inter-arrival time of the jobs but rather in the inter-arrival time where the nodes are idle. So we did not calculate the time differences between the jobs.

As shown in the plot, we do not see nodes 18-20 as these nodes did not have jobs to process in the period.

The lower the average waiting time of a node the higher the probability that these nodes will get the next job. Therefore the lower the number the better to put these nodes not in power-saving mode as these would potentially get the next job rather than the other nodes. We see that the first nine nodes have an average waiting time of below 40 seconds. Related to nodes 12-20 nodes 1-11 would better suit not to be put in power-saving mode.

5.6 Power-saving mode

During a three-month period under consideration, eight nodes operated in power-saving mode as listed in 4.1. A plot 5.13 was then generated to evaluate the amount of energy saved through this mode. The average energy consumption of nodes in idle mode was calculated in section 5.3, which is used to simulate energy consumption over the last three months as if the power-saving mode has not been implemented.

The following figure provides a depiction of energy saved via power-saving mode as it currently stands:

The energy consumption with power-saving mode consumes energy ranging from 1500W to 4000W depending on the powered-on nodes. The table below illustrates the effect of power-saving

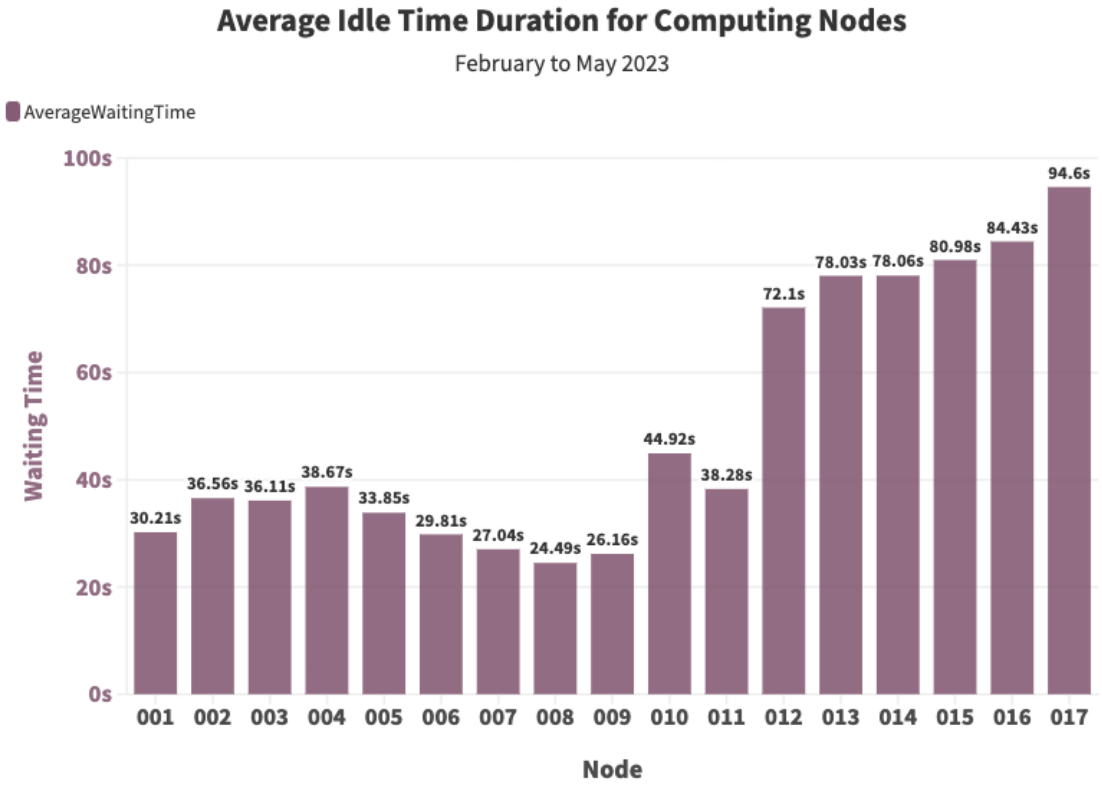

Figure 5.12: The average idle time duration between processed jobs for each node.

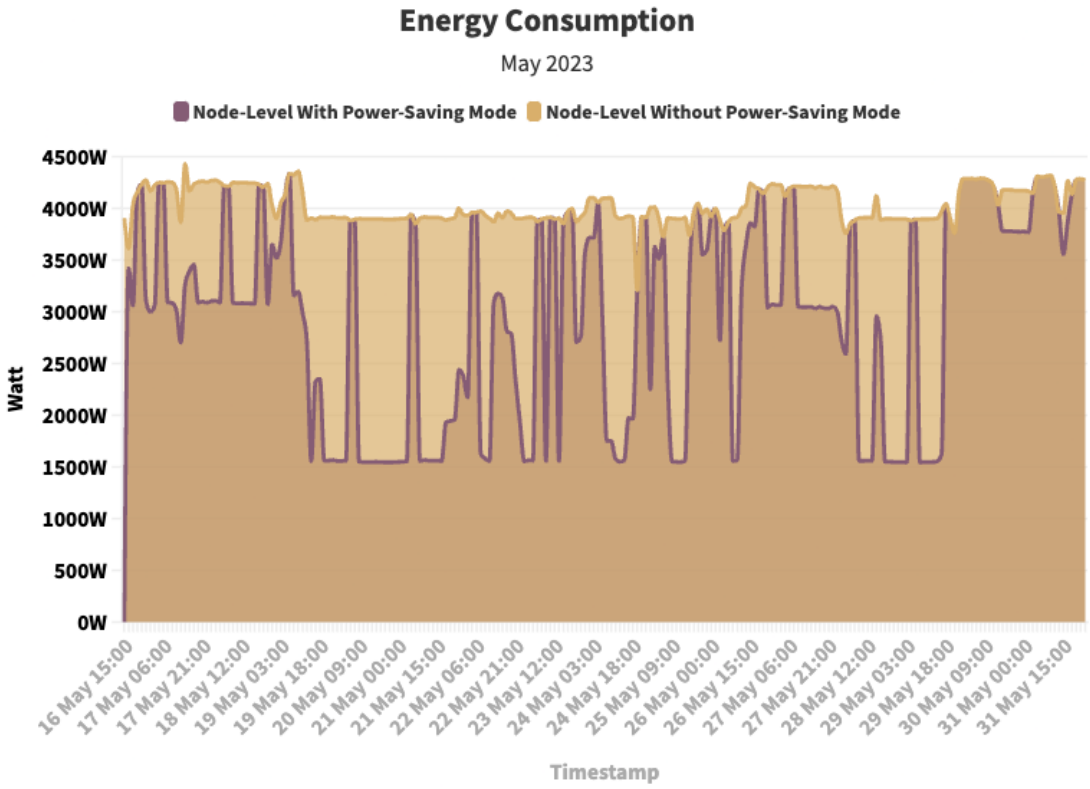

Figure 5.13: Comparison of energy usage: With vs Without Power-Saving Mode.

nodes referring to the data from figure 5.13:

Dataset	Average Power	Energy Consumption 15-Days
Without Power-Saving Mode	4016.8 W	5.2 GJ
With Power-Saving Mode	2933.6 W	3.8 GJ

Table 5.2: Energy Saving with current scenario.

With the current energy mechanism, 27.0% of energy consumption was saved. Without power-saving nodes, these energy savings would not have been realized.

5.7 Simulation of Result

5.7.1 Enhancement of Power-Saving Approach

The analysis offers an opportunity to further enhance the existing power-saving mechanism to better optimize energy consumption. We noted that nodes 18-20 were not utilized for any jobs over the past three months, indicating that they could be switched to power-saving mode.

In light of this, reallocating the eight non-power-saving nodes according to their inter-arrival times could serve as a simple solution. Given that nodes 18-20 were never used, and node 17 was seldom utilized, it seems reasonable to configure these as power-saving nodes. The proposed model suggests keeping only five nodes active, as the jobs processed did not require more than this. The remaining three nodes (18-20) are effectively dormant, contributing to unnecessary energy consumption.

Node 17, being the fifth active node, may also be better replaced by one of the first nine nodes, given their lower average inter-arrival times.

In order to simulate the potential energy-saving benefits of this revised approach, we'll modify the existing node-level dataset based on the average idle and busy energy consumption calculated earlier. We'll use past job data to simulate and estimate potential energy savings.

Under this new strategy, nodes 1-4 remain unchanged. For node 5, idle energy consumption will be added for periods it is powered on but not processing any jobs. Nodes 6-16 will continue to operate as power-saving nodes.

For node 17, periods of idle time will be noted and for these, we will allocate zero consumption as the node will be in power-saving mode in this scenario. Nodes 18-20 will be powered down entirely due to their lack of use.

The table and figure below show a clear potential for energy saving with this updated configuration.

Dataset	Power	Energy Consumption 15-Days
Existing Power-Saving Configuration	2933.6 W	3.8 GJ
Updated Power-Saving Configuration	2366.2 W	3.1 GJ

Table 5.3: Energy Saving with current scenario.

With these adjustments, we can improve the existing power-saving configuration by an additional 18.4%.

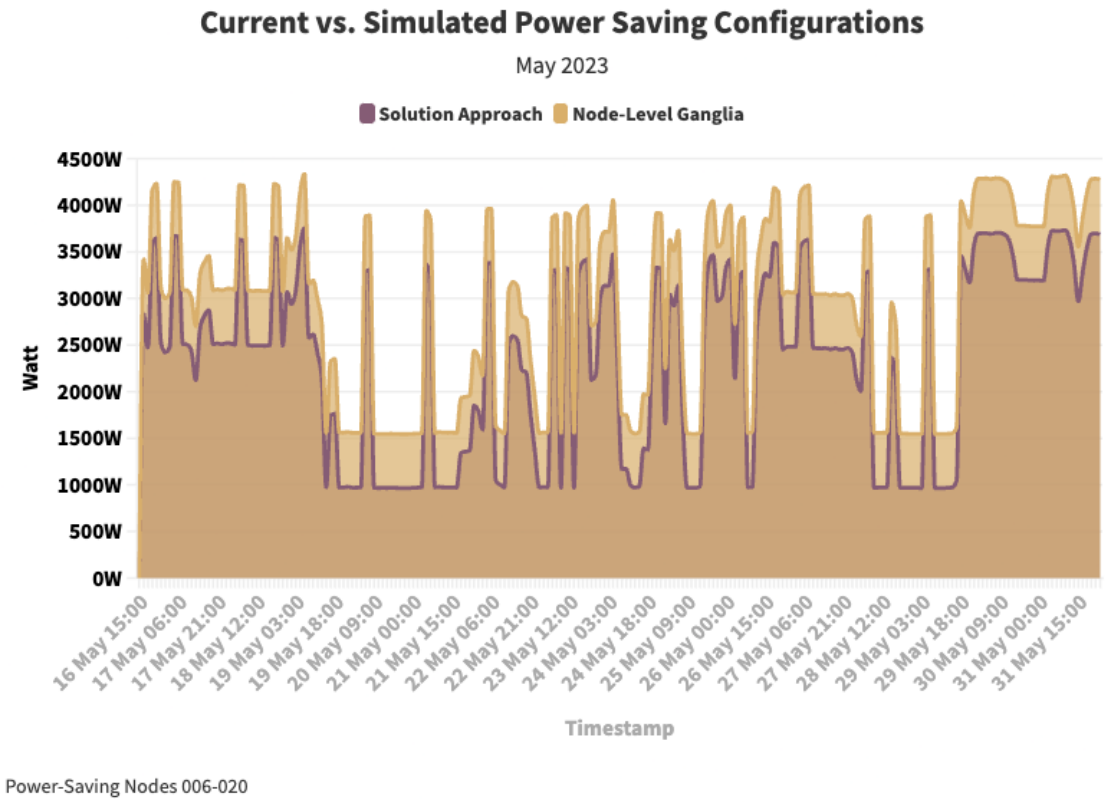

Figure 5.14: Comparison between Current and Changed Power Saving Configurations

5.7.2 Reducing SuspendTime

Given that the average arrival time is calculated in minutes, reducing the `SuspendTime` may be a wise move. Initially reducing the suspension time from two hours to one is recommended. This precaution aims to mitigate the risk of unnecessary power cycling of the nodes.

Observation of the nodes will be necessary to check if the frequency of powering up and down increases with this adjustment. If this proves to be problematic, it will need to be improved.

With a reduction of `SuspendTime` by an hour, energy equivalent to an hour of idle time is saved. This translates to approximately 69.5MJ of energy saved for each power-down process, given the average idle energy consumption is about 193 Watts. The earlier power-down time, achieved by reducing the `SuspendTime` from two hours to one, results in substantial energy savings.

Chapter 6

Discussion

This thesis initially posed four key questions, which will be addressed in this discussion section:

1. How much energy does an HPC job consume?
2. How much energy do nodes consume in idle/busy mode?
3. Does powering down nodes save energy overall?
4. What is the best strategy for powering up/down nodes?

6.1 Jobs Energy consumption

We presented plots displaying the energy consumption of HPC jobs during February and May 2023, and depicted the distribution of these nodes, observing that the average energy consumption of nodes is approximately 0.23 MJ. The distribution of jobs declines sharply with jobs exceeding 0.5 MJ. The correlation between the job duration and energy consumed indicated that there are two main groups of jobs, the distinguishing factor being the involvement of the GPU node. The correlation between job duration and energy consumption is approximately 1.1 MJ/hour.

6.2 Idle vs Busy Nodes

As mentioned in section 5.3, the idle and busy modes do not markedly impact energy consumption as they consume nearly equivalent amounts of energy. The average energy consumption of nodes during idle periods is about 195 Watts, whereas during busy periods it is 215 Watts. This minimal difference in energy consumption underlines the importance of finding solutions for idle nodes, such as powering down.

6.3 Energy-Saving with Power-Saving Mechanism

Section 5.7 effectively demonstrates how the analysis yielded energy-saving strategies. An outstanding case was the power-saving mechanism strategy which resulted in energy savings from February

to May 2023 by powering down nodes and implementing improved strategies such as placing nodes 18-20 in power-saving mode, obtaining five not power-saving nodes instead of eight nodes, or adapting the chosen power-saving nodes regarding the average arrival time plot. These could potentially lead to further savings in the future based on these results. The existing power-saving mechanism 5.6 reduced overall energy consumption by 27%. Additionally, the proposed enhancement of the existing power-saving configuration improved energy consumption by 18.4

6.4 Strategy for powering nodes up and down

The average idle time duration suggested, with relatively short waiting time, the best candidates for nodes that should not be placed in power-saving mode, as these nodes are more likely to receive jobs than others. It was found that nodes 12-20 should be prioritized for power-saving mode as they had the longest waiting times. Additionally, the `SuspendTime` could potentially be reduced from its current 2-hour setting. The overall low waiting time, ranging between 30 and 90 seconds, implies that most nodes will receive their jobs within a short period. This would permit a reduction in `SuspendTime`, enabling quicker power-down of configured nodes. The current 2-hour parameter could be tested at shorter intervals, perhaps initially at 1 hour, and then with further observation, possibly reduced to 30 minutes. This precaution aims to mitigate the risk of unnecessary power cycling of the nodes.

Chapter 7

Conclusion

One of the primary objectives of this thesis was to delve deeply into the provided dataset, unearthing its hidden potential for valuable insights, particularly concerning energy consumption. The discovery process carried out in this study brings new knowledge to the field of High-Performance Computing (HPC), relating specifically to the questions investigated.

An important finding of this study is the impact and efficiency of current power-saving mechanisms. According to our analysis, these mechanisms save an overall 27% of energy consumption. More importantly, it was found that there is room to enhance this saving by an additional 18.4%. This discovery opens up a path for future work focused on enhancing power-saving mechanisms in HPC.

Our examination of energy consumption within the miniHPC from February to May 2023 revealed a total consumption of 40.8 GJ. Interestingly, only 34% of this consumption was attributed to actual job processing. This implies a significant opportunity for energy savings through improved strategies that manage the remaining 66% of energy consumption more efficiently.

Moreover, the analysis also led to some important conclusions regarding the utilization of nodes 12-20. Due to their low job frequency, high waiting times, and substantial energy consumption during idle periods, it's clear that these nodes should be prioritized for power-saving mode. This strategy could play a crucial role in achieving more efficient energy use in HPC systems.

Notably, the data showed little difference in energy consumption between idle and busy nodes. This finding challenges the common assumption that busy nodes consume significantly more energy. It suggests that alternative energy-saving strategies need to be considered and developed to minimize energy waste, given that idle nodes consume almost as much energy as busy nodes.

Chapter 8

Future Work

This thesis is just the start, and it has opened up many new topics. The need to reduce energy use in High-Performance Computing (HPC) systems is only growing. This work has revealed many opportunities for further, more detailed study of energy use in HPC systems, and highlighted many potential areas for future research.

8.1 Automation and Anomaly Detection

The plots that were created could be automated, and alerts could be sent when a node is exceptionally consuming energy without receiving a significant number of jobs. The goal would be to automate changes to the settings in the future so that the configuration files will be automatically updated. The initial approach could involve automating the parameter update and tracking overall node usage to determine which nodes were used efficiently and which were not.

8.2 CPU Frequency Tuning

For some nodes, it might be possible to reduce the CPU frequency without impacting the execution time. While this may not be feasible for all programs, for those that do not require high frequencies, this could offer significant energy savings. Investigating this further and implementing it where feasible could reduce overall energy consumption. The challenge would be to identify which nodes or jobs require the frequency and which do not.

8.3 Energy Capping for Users

An intriguing prospect for future work would be to explore energy capping for users. Each user could be assigned an "energy budget". If a user exceeds this budget, they would need to request more energy. An alert system could be implemented to notify administrators if a user consumes more energy than usual.

8.4 Job Categorization

The scatter plot from the job duration and energy consumption analysis revealed two main groups of jobs based on their energy consumption. Future work could involve categorizing jobs into subgroups, determining unique characteristics of each group, and discovering potential energy-saving insights.

8.5 Expanding Energy Monitoring to GPU Nodes

One potential future work, following the findings and insights from this thesis, is extending our energy monitoring strategies to encompass GPU-based nodes in HPC systems. Jobs that utilize GPU resources can often consume substantial amounts of energy. Consequently, analyzing and minimizing the energy usage of GPU jobs could lead to significant savings in power consumption, further optimizing the overall efficiency of the HPC system.

Bibliography

[1] Amax. Power usage effectiveness, May 2023.

[2] Siegfried Benkner, Franz Franchetti, Hans Michael Gerndt, and Jeffrey K Hollingsworth. Automatic application tuning for hpc architectures (dagstuhl seminar 13401). In *Dagstuhl Reports*, volume 3. Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik, 2014.

[3] Julita Corbalan and Luigi Brochard. Ear: Energy management framework for supercomputers. *Barcelona Supercomputing Center (BSC) Working paper*, 2019.

[4] Pawel Czarnul, Jerzy Proficz, Adam Krzywaniak, et al. Energy-aware high-performance computing: survey of state-of-the-art tools, techniques, and environments. *Scientific Programming*, 2019, 2019.

[5] Luca Deri, Simone Mainardi, and Francesco Fusco. tsdb: A compressed database for time series. In *Traffic Monitoring and Analysis: 4th International Workshop, TMA 2012, Vienna, Austria, March 12, 2012. Proceedings 4*, pages 143–156. Springer, 2012.

[6] Flourish. Flourish: The platform for data visualization and storytelling, 2023. Accessed: 2023-06-15.

[7] José Goldemberg and Oswaldo Lucon. *Energy, environment and development*. Earthscan, 2010.

[8] Anthony JG Hey. High performance computing: Past, present and future. 1996.

[9] Katja Keller. Analysis of historical hpc job data. Bachelor's thesis, University Basel, 2022.

[10] Pascal Kunz. Hpc job-monitoring with slurm, prometheus and grafana. Bachelor's thesis, University Basel, 2022.

[11] Matt Massie, Bernard Li, Brad Nicholes, Vladimir Vuksan, Robert Alexander, Jeff Buchbinder, Frederiko Costa, Alex Dean, Dave Josephsen, Peter Phaal, et al. *Monitoring with Ganglia: tracking dynamic host and application metrics at scale*. " O'Reilly Media, Inc.", 2012.

[12] Matthew L Massie, Brent N Chun, and David E Culler. The ganglia distributed monitoring system: design, implementation, and experience. *Parallel Computing*, 30(7):817–840, 2004.

[13] K. Rajamani and C. Lefurgy. On evaluating request-distribution schemes for saving energy in server clusters. In *2003 IEEE International Symposium on Performance Analysis of Systems and Software. ISPASS 2003.*, pages 111–122, 2003.

[14] SchedMD. Slurm workload manager: Sacct-command. Retrieved from https://slurm.schedmd.com/scontrol.html, 15th May 2023. Accessed on 21th May 2023.

[15] SchedMD. Slurm workload manager: Sacct-command. Retrieved from https://slurm.schedmd.com/sacct.html, 15th May 2023. Accessed on 21th May 2023.

[16] SchedMD. Job launch design guide. Retrieved from https://slurm.schedmd.com/job_launch.html, 1th Augsut 2022. Accessed on 21th May 2023.

[17] SchedMD. Energy Accounting & External Sensors Plugins. Retrieved from https://slurm.schedmd.com/SUG13/energy_sensors.pdf, 2013. Accessed on 10th April 2022.

[18] SchedMD. Workload Scheduling and Power Management. Retrieved from https://slurm.schedmd.com/SLUG18/power_management.pdf, 2018. Accessed on 10th April 2022.

[19] SchedMD. Slurm workload manager: Overview. Retrieved from https://slurm.schedmd.com/overview.html, 2023. Accessed on 10th April 2023.

[20] SchedMD. Slurm Design. Retrieved from https://slurm.schedmd.com/slurm_design.pdf, 23th June 2003. Accessed on 10th April 2022.

[21] SchedMD. Slurm workload manager: Overview. Retrieved from https://slurm.schedmd.com/power_save.html, 3th May 2023. Accessed on 21th May 2023.

[22] Yu Tang, Shaojun Xie, Chaohua Zhang, and Zegang Xu. Improved z-source inverter with reduced z-source capacitor voltage stress and soft-start capability. *IEEE Transactions on Power Electronics*, 24(2):409–415, 2009.

[23] TechTarget. High-performance computing (hpc). https://www.techtarget.com/searchdatacenter/definition/high-performance-computing-HPC. Accessed on April 4, 2023.

[24] University of Basel. minihpc. https://hpc.dmi.unibas.ch/research/minihpc/. Accessed on June 4, 2023.

[25] Andy B Yoo, Morris A Jette, and Mark Grondona. Slurm: Simple linux utility for resource management. In *Job Scheduling Strategies for Parallel Processing: 9th International Workshop, JSSPP 2003, Seattle, WA, USA, June 24, 2003. Revised Paper 9*, pages 44–60. Springer, 2003.