Vrije Universiteit Amsterdam

Universiteit van Amsterdam


Master Thesis

# Data Characterization and Anomaly Detection for HPC Datacenters Using Machine Learning

**Author:** Wenjun Liang (2726770)

*1st supervisor:* Prof. dr. ir. Alexandru Iosup
*daily supervisor:* Xiaoyu Chu, M.Sc.
*2nd reader:* Prof. dr. Tiziano De Matteis

September 13, 2023

# Abstract

In the domain of High-Performance Computing (HPC), anomaly detection emerges as a pivotal challenge. This research delves deeply into the architecture of Lisa and presents a thorough analysis of time-series heatmaps, accentuating anomalies. Diverging from conventional methods that primarily utilize node log data, this study integrates and rigorously examines both node and job data from slurm. This approach provides a comprehensive view of HPC clusters, revealing their structure, patterns, and potential anomalies. Recognizing the inherent challenges of labeled data, this work explores the efficacy of various unsupervised learning models for anomaly detection. Through established metrics such as FPR and AUC-ROC, a quantitative evaluation is presented, assisting stakeholders in making informed model selections. During the data analysis phase, visualization techniques are employed to enhance the understanding of HPC datasets, refining the approach to anomaly detection.

# Acknowledgements

First and foremost, I would like to express my deepest gratitude to my advisors, Xiaoyu Chu and Alexandru Iosup, for their invaluable guidance, patience, and unwavering support throughout this journey. Their expertise and insights have been instrumental in shaping this research.

I owe a debt of gratitude to my parents, whose financial support made my academic pursuits possible. Their belief in my potential has been a constant source of strength and motivation.

Special thanks go out to my friends, who have been my emotional pillars, always there to provide encouragement and a listening ear during challenging times.

I would also like to extend my appreciation to the staff of SURFsara for their technical assistance, which was crucial in the realization of this work.

Lastly, I want to acknowledge my own perseverance. Over the past two years, I have faced and overcome numerous obstacles, each step bringing me closer to this milestone. This achievement is a testament to the hard work, dedication, and resilience that I have mustered.

# Contents

# List of Figures

# List of Tables

# Chapter 1

# Introduction

High-Performance Computing (HPC) clusters have become increasingly important in various sectors, from scientific research to industrial applications. However, the complexity and scale of these systems also make them susceptible to operational anomalies. In the following subsections, we will delve into the importance of HPC, and how our research aligns with existing visions and challenges in the field.

## 0.1 The Growing Importance of High-Performance Computing

High-Performance Computing (HPC) clusters have become indispensable in various fields, from scientific research to industrial applications [12]. Their capabilities extend to complex simulations, large-scale data analysis, and the development of artificial intelligence algorithms. However, the increasing complexity and scale of these clusters also make them susceptible to operational anomalies, which can severely impact their performance and reliability.

Our research on anomaly detection in High-Performance Computing (HPC) clusters aligns closely with existing visions in the field. Specifically, the work of Iosup et al. [9] provides a foundational perspective for our study. By focusing on the reliability and efficiency of HPC clusters through unsupervised machine learning techniques, our work contributes to improving the manageability of these complex systems. This focus aligns with the call for responsible computer infrastructure, as improved anomaly detection enhances the reliability and dependability of HPC clusters.

Also, our research addresses several key challenges outlined in the Manifesto by Iosup et al. [8]. Effective anomaly detection can optimize resource allocation, contributing to sustainability goals. Our work also aims to make HPC clusters more reliable and, therefore, more usable in critical sectors like healthcare and national security. By addressing these

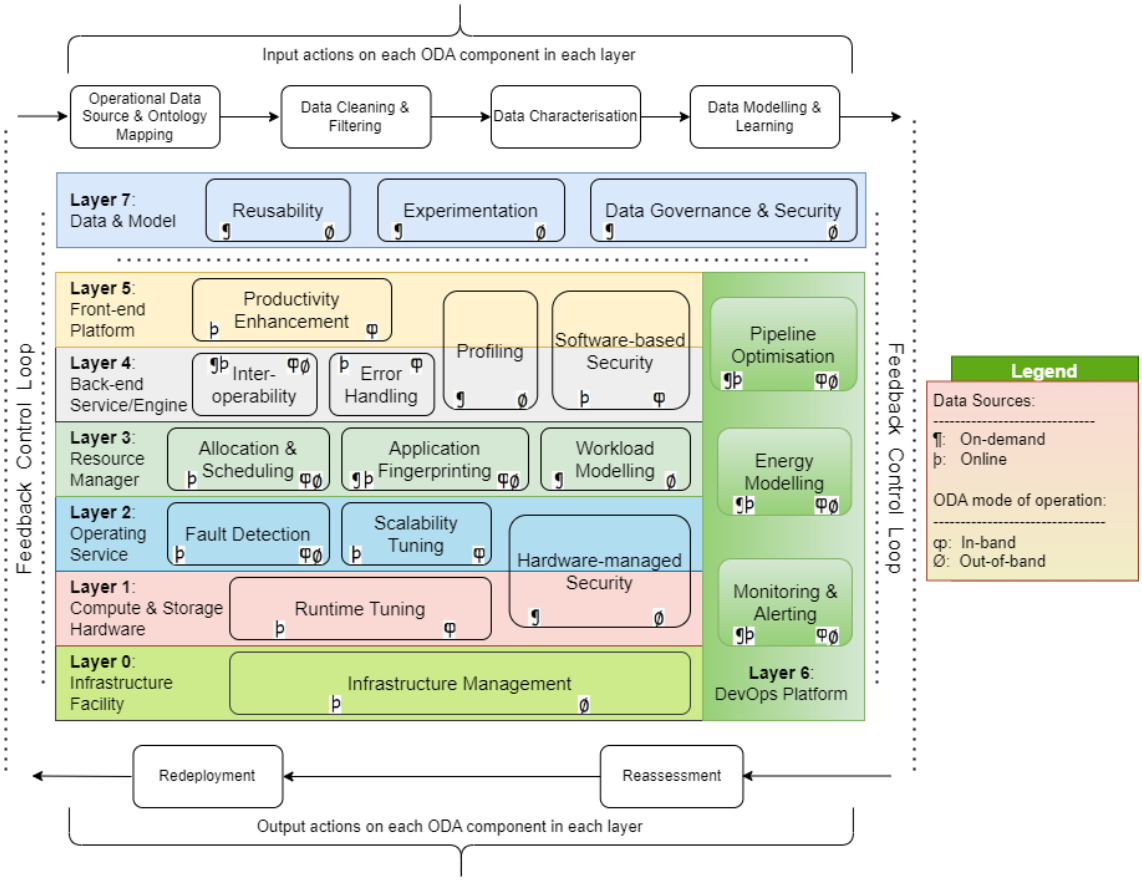


**Figure 1.1:** ODA architecture. Source: [18]

challenges, our research contributes to shaping a national strategy for advancing computer systems and networking research.

Additionally, Our research aligns closely with the Operational Data Analysis (ODA) architecture depicted in Figure 1.1. Specifically, the datasets we utilize fit into distinct layers of the ODA model:

- **Layer 0: Infrastructure Facility** - The PROMETHEUS dataset, which provides metrics related to the underlying hardware and software infrastructure, originates from this layer.
- **Layer 3: Resource Manager** - The SLURM dataset, which offers insights into job scheduling and resource allocation, is sourced from this layer.

Both datasets are categorized as operational data and have overlapping collection times,

making them complementary for analysis and particularly useful for anomaly detection.

Challenges and Limitations in Anomaly Detection

Anomalies in HPC clusters can lead to a range of adverse effects. For instance, an undetected anomaly in a weather simulation cluster could result in inaccurate forecasts, affecting sectors like agriculture, aviation, and public safety. One of the most significant challenges in anomaly detection is the scarcity of labeled data. This limitation often leads to imbalanced datasets and makes supervised and semi-supervised learning methods less suitable. Consequently, there is a growing interest in employing unsupervised learning techniques for anomaly detection in HPC clusters.

# 1 Problem Statement

Existing anomaly detection methods in HPC clusters often focus solely on node log data, neglecting other potentially valuable data sources like workload or job data. Additionally, the requirement for labeled data in supervised and semi-supervised learning models presents practical challenges. These limitations hinder the development of comprehensive and interpretable anomaly detection methods. This study aims to address these challenges by proposing a more holistic approach that integrates multiple types of operational data and employs unsupervised learning techniques.

# 2 Research Questions

The overarching research question guiding this study is:

**How can we design and evaluate an effective unsupervised machine learning model for anomaly detection in large-scale HPC clusters by integrating multiple types of operational data?**

To dissect this main question, we formulate the following sub-questions:

RQ1. **What are the characteristics of HPC Node data and Job Data?**
Understanding the characteristics of HPC Node data and Job Data is foundational for any subsequent analysis or modeling. This question aims to provide a comprehensive understanding of the data's structure, distribution, and inherent patterns.

RQ2. **How can node data be integrated with job data for the purpose of anomaly detection?**
The integration of node data with job data represents an innovative approach to

creating a more holistic view of the system's operation. This question is significant because it explores the synergistic effects of combining different types of data. The integration could potentially lead to more accurate and reliable anomaly detection models.

RQ3. **How do various unsupervised machine learning models perform in detecting anomalies in large-scale HPC clusters?**
This question is crucial for the practical application of our research. Given the scarcity of labeled data in the domain of HPC anomaly detection, unsupervised machine learning models offer a viable alternative. This research question aims to evaluate the performance of various unsupervised models in a real-world setting, providing insights into their effectiveness, reliability, and limitations. The findings could guide future research and practical implementations, helping to optimize the selection of machine learning models for anomaly detection in HPC clusters.

# 3 Contributions

In the following subsections, we will elaborate on the specific societal and scientific contributions, as well as the innovative approaches that set this work apart from existing studies in the field.

## 3.1 Societal and Scientific Contributions

This research aims to make significant contributions both to society and the scientific community. Effective anomaly detection mechanisms can substantially improve the reliability and efficiency of various critical infrastructures, such as healthcare systems and financial institutions. On the scientific front, this work aims to enrich the field of computer science by introducing a comprehensive, unsupervised learning-based approach to anomaly detection in HPC clusters.

## 3.2 Innovative Approaches

- **Comprehensive Data Integration:** This study goes beyond traditional methods that focus solely on node log data. By integrating node log data with Slurm job data, we offer a more comprehensive understanding of the system's state.
- **Exploration and Assessment of Unsupervised Learning:** Given the scarcity of labeled data, this research employs various unsupervised learning models for anomaly

detection. These models are rigorously evaluated using established metrics such as False Positive Rate (FPR) and Area Under the Receiver Operating Characteristic Curve (AUC-ROC).

- **In-Depth Data Characterization:** This work performs an in-depth analysis of relevant datasets provided by Surfsara. This characterization serves as a foundation for the subsequent stages of anomaly detection, making our approach more robust and insightful.

- **Visual Analysis:** Our work incorporates advanced visualization techniques such as heatmap, density plot and other kinds of charts to better understand the data's characteristics and anomalies. These visual insights not only enhance our understanding of the data but also provide system administrators and researchers with actionable information.

# 4 Significance of the Study

This section aims to elucidate the broader implications of our study, breaking them down into economic and societal benefits as well as technological and academic advancements.

## 4.1 Economic and Societal Benefits

Effective anomaly detection can prevent costly service disruptions and optimize resource allocation in HPC clusters. It also has a ripple effect on sectors vital to societal functioning, such as healthcare and national security, and contributes to energy savings.

## 4.2 Technological and Academic Advancements

This study contributes to machine learning innovation by promoting unsupervised learning models and paves the way for more holistic monitoring solutions. The methodologies and findings serve as a foundation for future research in anomaly detection and HPC.

# 5 Plagiarism Declaration

I confirm that this thesis work is my own work, is not copied from any other source (person, Internet, or machine), and has not been submitted elsewhere for assessment.

# Chapter 2

# Background

This chapter provides an overview of the foundational concepts and technologies that underpin this research.

## 1 HPC System Model

High-performance computing (HPC) clusters refer to a group of interconnected computers that work together to solve complex computational problems. HPC clusters are commonly used in scientific research, data centers, and other computing-intensive fields due to their capability to process large amounts of data and perform computationally intensive tasks quickly.

In an HPC cluster, each computer is referred to as a node, and each node can handle multiple job submissions, which are computational tasks assigned to the node. Nodes are interconnected through a high-speed network, allowing for efficient communication and data exchange. The resource manager software, such as SLURM, is used to manage and schedule jobs across the cluster based on the available resources. As the HPC cluster becomes larger and more complex, it becomes increasingly difficult to ensure its reliability and availability.

Figure 2.1 [3] illustrates the system model referenced. Jobs characterized in the study were submitted to the Lisa HPC cluster. This HPC cluster consists of several racks, with each rack housing multiple servers. These servers are interlinked using a high-speed network connection

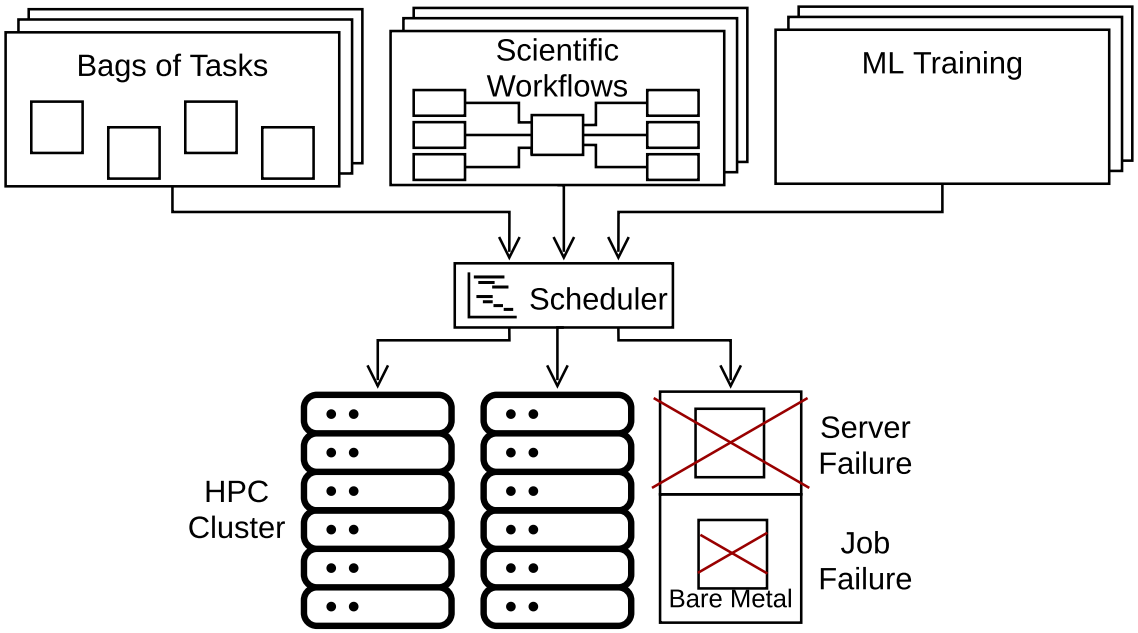


**Figure 2.1:** Architecture of Lisa, Source: [3]

# 2 Unsupervised Machine Learning for Anomaly Detection

Anomaly detection in High-Performance Computing (HPC) data modeling is a pivotal research area focusing on identifying unexpected patterns or outliers in data that might indicate incidents like cyber-attacks, system faults, or performance issues. Given the complexity and voluminous data generated by HPC systems, automated anomaly detection algorithms are indispensable for spotting abnormal patterns that suggest potential problems or improvement areas. Various techniques, from statistical methods leveraging data distribution for identifying significant deviations to machine learning to approaches such as supervised, unsupervised, and semi-supervised learning, encompassing clustering, classification, and neural networks, are employed. This anomaly detection plays a crucial role in maintaining the efficiency, reliability, and security of HPC systems by enabling system administrators to proactively deal with potential issues, enhancing overall system performance, and facilitating predictive maintenance to preempt system breakdowns.

Unsupervised learning does not rely on pre-labeled datasets. Unsupervised learning models try to identify anomalies that deviate from these patterns by learning patterns of normal behavior. The advantage of this approach is that it can handle new, unlabeled anomalies because it does not require the labeling of the anomalies. However, it may misrepresent

some atypical normal behaviors as anomalies. Typical techniques used by unsupervised learning methods in HPC anomaly detection include clustering, isolated forests, and self-encoders (Autoencoders) in deep learning. Unsupervised learning is characterized by its independence from existing labeled data for anomaly identification. In large-scale computing systems, abnormal events are usually rare. This means it's difficult to obtain sufficient labeled data to train a supervised learning model, where unsupervised learning plays an important role.

First, research by Morrow et al. explored the application of unsupervised clustering in HPC sensor data[14]. They focused on a DBSCAN algorithm to filter out normal behavior in large-scale data and sort out anomaly points based on the distance from the nearest normal cluster. Their results showed that DBSCAN outperforms other clustering algorithms, such as k-means and Gaussian kernel density estimation.

Molan et al. proposed a neural network-based framework RUAD, bringing a new perspective to anomaly detection in HPC [13]. Their neural network framework includes Long Short Term Memory (LSTM) units to explicitly consider the time series of the data.

In the anomaly detection of HPC, privacy protection is another important consideration. To address this issue, Ghiasvand proposed a method called uPAD, an unsupervised and privacy-preserving approach[5]. This method generates models representing normal system behavior by processing anonymous syslog entries using neural networks. These models are used to monitor and evaluate system behavior to detect anomalies at an early stage.

In addition to anomaly detection, the interpretability of the algorithms is also an important research direction. In this regard, Carletti et al. introduced a method called DIFFI, providing interpretability for the Isolation Forest (IF) algorithm[1]. The DIFFI method provides a quantitative measure of feature importance for anomaly detection tasks, which better helps understand the underlying data generation process and perform root cause analysis.

Finally, Wang et al. researched online anomaly detection methods for web applications under the impact of dynamic workloads [20]. This research introduces a workload-aware anomaly detection framework that combines incremental clustering, Local Outlier Factor (LOF), and the t-test method.

# Chapter 3

# Data Characterization and Integration for Node Data and Job Data

This chapter aims to address the first research question (RQ1): "What are the characteristics of HPC Node data and Job Data?" By exploring the characteristics of both Node and Job data, we seek to provide a foundational understanding that will inform subsequent analyses and modeling efforts. We will delve into the structure, distribution, and inherent patterns of the data.

## 1 PROMETHEUS Node-Level Dataset

In this section, we are going to characterize the PROMETHEUS Node-Level Dataset.

### 1.1 Dataset Introduction

The first dataset used in this study was collected from the LISA National Computing Cluster in the Netherlands, hosted at SURFsara [**?** ]. Table 3.1 showcases the diverse hardware information of selected nodes currently operational in the data center.This dataset consists of software and hardware trace data, which are real-time data generated during the operation of the nodes [10]. The data was collected by the PROMETHEUS time-series monitoring system [10].

PROMETHEUS is an open-source monitoring system and time series database that serves as a robust tool for collecting metrics in real-time. Its capabilities for scraping metrics at regular intervals make it particularly suitable for monitoring large-scale, dynamic systems like HPC clusters [16].

The dataset under consideration is extensive, both in terms of time span and granularity. It covers a period of nearly 145 days, providing a rich temporal context that is invaluable

| CPU Quantity | Memory | GPU Model | GPU Quantity |
|---|---|---|---|
| 24 | 191,488 | Titan RTX | 4 |
| 12 | 257,024 | Titan V | 4 |
| 40 | 1,481,024 | Titan RTX | 2 |
| 16 | 93,184 | N/A | N/A |
| 48 | 2,049,024 | N/A | N/A |
| 52 | 300,544 | N/A | N/A |

**Table 3.1:** Hardware information of LISA nodes.

for understanding long-term trends and patterns. The data is collected every 30 seconds, offering high granularity that allows for detailed analysis of system behavior at any given moment.

The dataset's size of 43 GB and its structure—comprising 129,546,553 rows and 63 columns—indicate the complexity and high dimensionality of the data. This makes it a rich source for extracting meaningful features but also poses challenges in terms of computational resources required for analysis.

The inclusion of data from 343 nodes provides a comprehensive view of the system, allowing for node-specific as well as system-wide analyses. This is particularly important for anomaly detection, where understanding the behavior of individual nodes can be as crucial as understanding the overall system dynamics.

The PROMETHEUS dataset encompasses a wide array of metrics that capture the operational state and performance of each node in the HPC system. These metrics can be broadly classified into several categories:

- **System Metrics**: These provide information about the system's operational status, boot time, load averages, and the number of processes in different states.
- **Disk Metrics**: These offer insights into the disk activity of the node, including the number of I/O operations and the total number of bytes read and written.
- **Filesystem Metrics**: These provide information about the filesystem, including available and total space, the number of files, and any device errors.
- **Network Metrics**: These capture the network activity of the node, including the total number of bytes and packets received and transmitted, and the total number of dropped and multicast packets received.

- **Memory Metrics**: These provide information about the memory usage of the node, including active and dirty memory, free memory, and per-CPU memory.
- **GPU Metrics**: These provide information about the GPU status and performance of the node, including duty cycle, fan speed, used memory, power usage, and temperature.
- **Other Metrics**: These include various other metrics such as time, temperature, power usage, and various network statistics.

The metrics presented in the Table 3.2 have been specifically selected for their significant influence on determining the anomaly status of a node in the HPC system.

**Table 3.2:** Description of selected metrics in PROMETHEUS dataset.

| **Metrics** | **Description** |
|---|---|
| id | Measurement ID |
| timestamp | Time of the measurement |
| node | The number of rack and node |
| node_load1 | CPU utilization load average (1 minute) |
| node_load15 | CPU utilization load average (15 minutes) |
| node_load5 | CPU utilization load average (5 minutes) |
| node_memory_Active_bytes | Memory |
| nvidia_gpu_power_usage_milliwatts | Power usage of the GPU device in milliwatts |
| nvidia_gpu_temperature_celsius | Temperature of the GPU device in Celsius |

## 1.2 Statistical Analysis

The Table 3.3 shows the overall statistics.

**Table 3.3:** Summary statistics.

| Metric | Mean | Std. Dev. | 25% | 50% | 75% | Max |
|---|---|---|---|---|---|---|
| node_load15 | 11.01 | 50.60 | 0.05 | 4.74 | 16.00 | 4905.28 |
| node_load5 | 11.03 | 50.83 | 0.06 | 4.28 | 16.02 | 4908.67 |
| node_load1 | 11.04 | 50.96 | 0.07 | 4.09 | 16.00 | 4910.27 |
| node_memory Active_bytes | $1.13 \times 10^{10}$ | $1.68 \times 10^{10}$ | $1.59 \times 10^{9}$ | $3.87 \times 10^{9}$ | $1.43 \times 10^{10}$ | $2.10 \times 10^{12}$ |

### 1.2.1 Node Memory Active

The `node_memory_Active_bytes` metric, representing active memory bytes, averaged around 11.25 TB. However, the considerable standard deviation of 16.81 TB indicates a wide spread in memory usage, ranging from a minimum of 0 bytes to a substantial 2.10 PB.

**Analysis:** The high standard deviation suggests that memory usage is highly variable across different nodes or time periods. This could indicate inconsistent workload distribution or varying job requirements, which may require further investigation for optimization.ad in memory usage, ranging from a minimum of 0 bytes to a substantial 2.10 PB.

### 1.2.2 Node Load Over Time

Table 3.3 presents the node load statistics over different intervals. The analysis of the node load over varying time intervals reveals interesting patterns:

**15 minutes (node_load15)** : The mean load across all nodes over this interval was 11.01. However, a substantial standard deviation of 50.60 indicates significant variability in the load across nodes. The load values spanned a broad range, from a negligible 0.00 to a peak of 4905.28.

**5 minutes (node_load5)** : For this shorter interval, the average load was marginally higher at 11.03, accompanied by a similar standard deviation of 50.83. The load ranged between 0.00 and 4908.67.

**1 minute (node_load1)** : The most immediate load average, taken over the last minute, was 11.04, with the highest observed value being 4910.27.

**Analysis:** The high standard deviation suggests that some nodes might be under heavy load while others are underutilized. This could point to inefficient resource allocation strategies.

## 1.3 Anomaly Analysis

In this stage of the study, the datasets were annotated to identify anomalies. The criteria for determining whether a data point is an anomaly were based on a meeting interview with Duncan Kampert, a staff member at SURFsara. The thresholds used for annotation are presented in the Table 3.4:

**Table 3.4:** Thresholds for PROMETHEUS metrics.

| Metrics | Threshold |
|---|---|
| node_load1 | 2 x Total CPU Amounts |
| node_load15 | 2 x Total CPU Amounts |
| node_load5 | 2 x Total CPU Amounts |
| node_memory_Active_bytes | Memory of the node |
| nvidia_gpu_power_usage_milliwatts | Standard power usage of the GPU device |
| nvidia_gpu_temperature_celsius | Standard Temperature of the GPU device |

After applying these thresholds to the datasets, a total of 3558579 anomalies were identified. This represents approximately 2.88% of the total data points in the datasets.The standard deviation of 0.17 suggests that these anomalies are not uniformly distributed, either across nodes or temporally.

Figure 3.1 presented encapsulates all the time series data from the PROMETHEUS dataset, with the temporal scope aggregated on a daily basis. The x-axis represents the date, while the y-axis denotes the names of the nodes, specifically the days. In the heatmap, the color of each cell corresponds to the level of anomalies detected for a particular node on a given day: blue indicates a low level of anomalies, yellow signifies a medium level, and red represents a high level of anomalies. A deeper, more intense red color signifies a higher number of anomalies, indicating a day with more irregularities in the node's operation.

It's important to note that the process of anomaly detection is not always straightforward. The thresholds used in this study were determined based on expert knowledge from Surfsara and may not capture all possible anomalies. However, they provide a starting point for identifying patterns that deviate from the norm and can be refined in future work.

## 1.4 Metric Correlation

In this section, we delve into understanding the relationships between the various features in our dataset. Specifically, we examine how each feature correlates with others and aim to decipher patterns that can offer insights into the nature of the data, especially in the context of anomaly detection.

From Figure 3.2, a few noteworthy observations surface immediately:

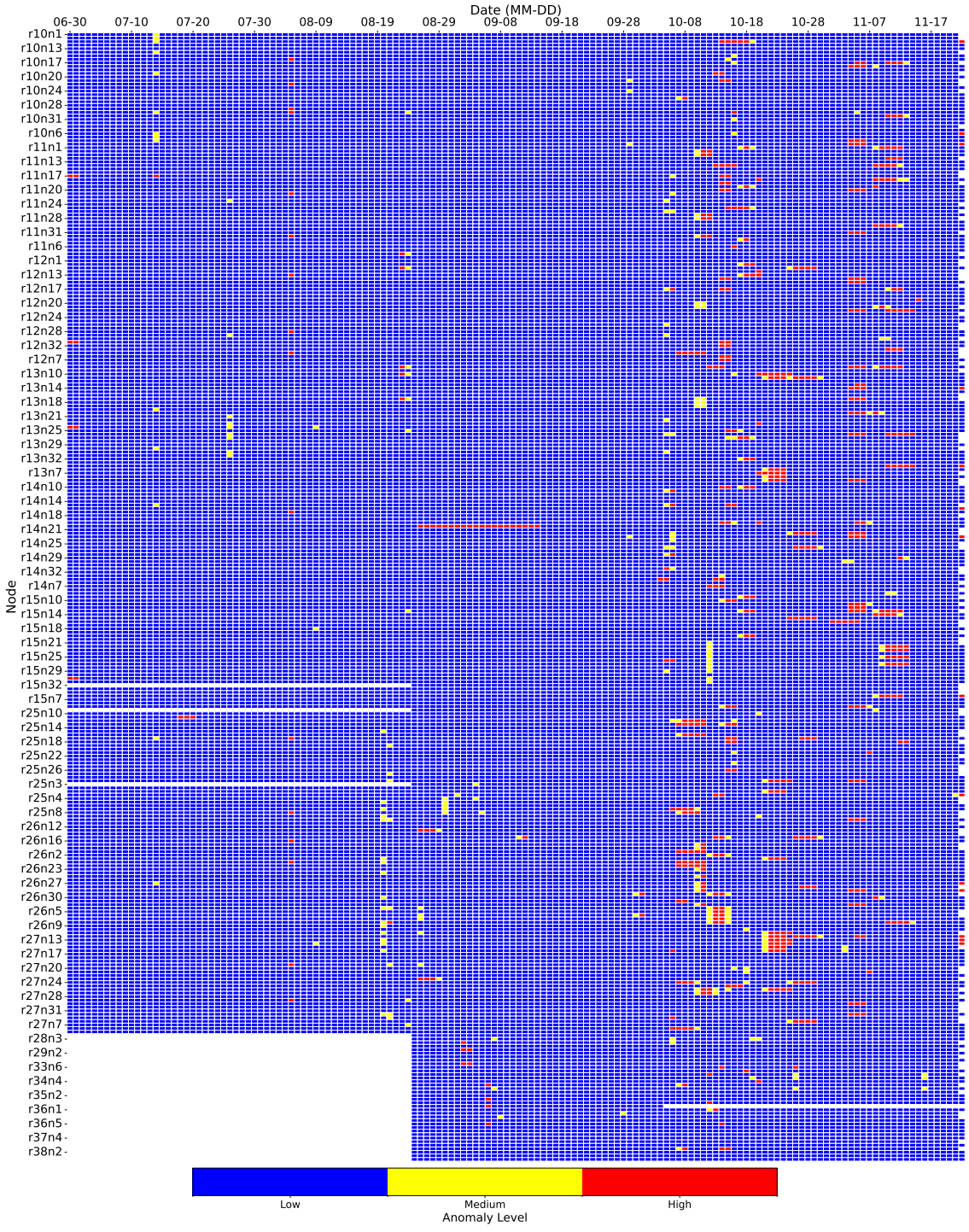


**Figure 3.1:** Time-series heatmap with anomalies for PROMETHEUS dataset, blue indicates low anomaly level, yellow indicates medium and red indicates high.

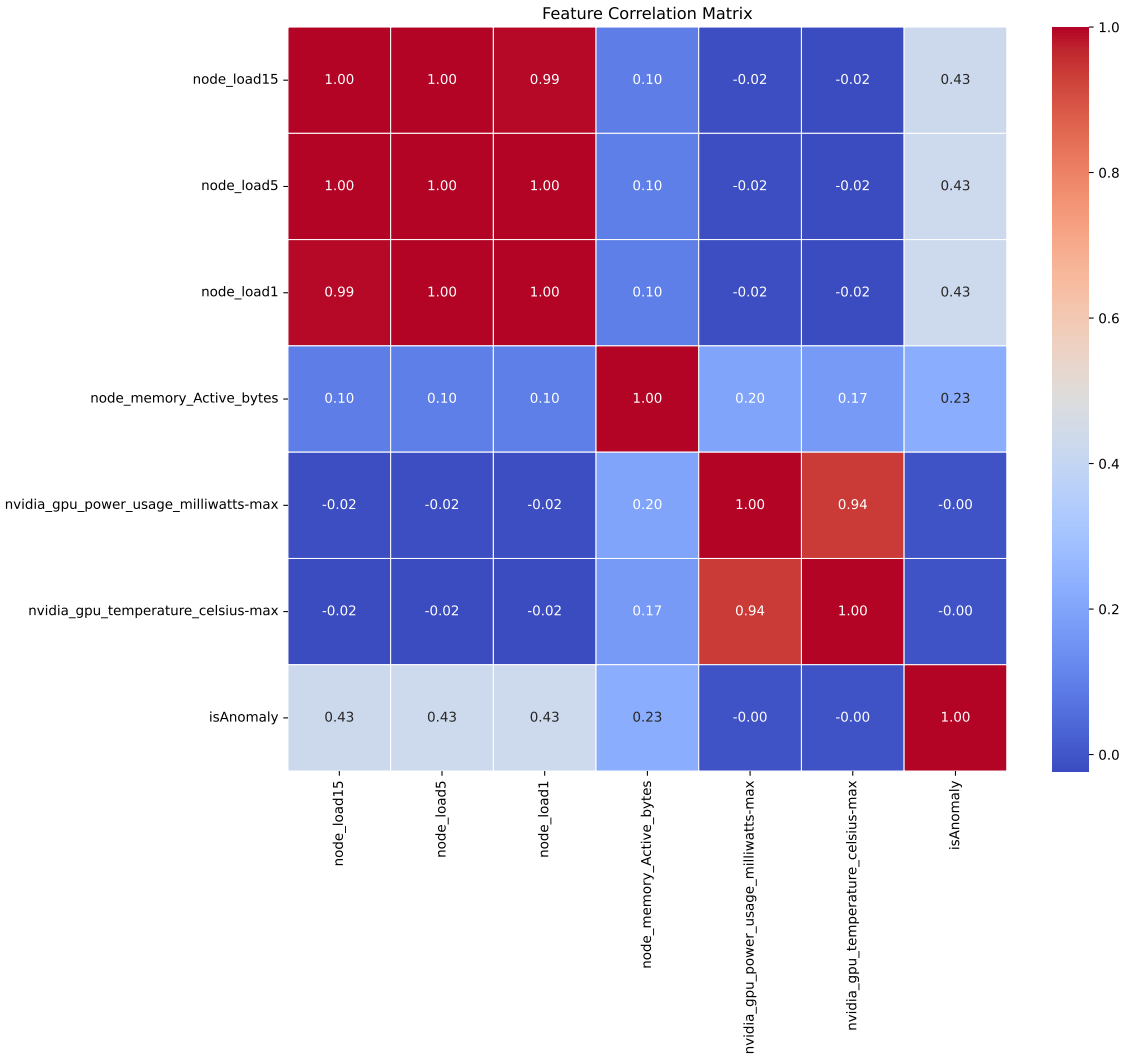


**Figure 3.2:** Feature correlation matrix.

### 1.4.1 Load Metrics Correlation

`node_load15`, `node_load5`, and `node_load1` are highly correlated with each other, with correlation values close to 1. This suggests that they carry similar information. Intuitively, this makes sense as they represent the load average of the system over different time intervals, and are expected to be closely aligned in behavior.

### 1.4.2 GPU Metrics Correlation

Both `nvidia_gpu_power_usage_milliwatts-max` and `nvidia_gpu_temperature_celsius-max` have a high correlation of 0.94. This indicates that as power usage in the GPU rises, there's a corresponding increase in GPU temperature. This is an expected relationship as higher workloads would generate more heat.

### 1.4.3 Memory and GPU Metrics

`node_memory_Active_bytes` has a modest correlation with `nvidia_gpu_power_usage_milliwatts-max` (0.20) and `nvidia_gpu`
`temperature_celsius-max` (0.17). This implies that while active memory usage might have some relation to GPU activities, it's not as tightly bound as the GPU metrics themselves.

### 1.4.4 Anomaly Correlations

:The `isAnomaly` column, which represents whether a data point is anomalous (with 1) or normal (with 0), has strong positive correlations with the load metrics (`node_load15`, `node_load5`, `node_load1`) with a correlation value of 0.43. However, it's interesting to note that it has negligible correlations with GPU related metrics and a moderate correlation with `node_memory_Active_bytes` (0.23).

Having drawn initial observations from the correlation matrix, it is crucial to further analyze these relationships to comprehend their implications on the dataset and how they could influence potential modeling decisions.

- **Load Metrics**: Given the high degree of correlation between the load metrics, it may be redundant to use all three for predictive modeling or anomaly detection. Reducing the dimensionality by selecting one of these could simplify models without sacrificing significant information.

- **GPU Metrics and Memory**: The relationship between GPU metrics and active memory bytes isn't robust, suggesting that memory usage and GPU activities might be somewhat independent. In a practical scenario, it's possible that high GPU activity (like deep learning model training) might not necessarily coincide with memory-intensive operations.
- **Anomaly Correlations**: The strong positive correlation of the `isAnomaly` column with the load metrics suggests that anomalies are heavily influenced by system load. This could be indicative of scenarios where the system becomes overloaded, leading to anomalous behavior. On the other hand, GPU metrics, despite their importance in the dataset, do not strongly indicate anomalies on their own.

## 1.5 K-Means Cluster Distribution

The analysis was underpinned by the KMeans clustering algorithm, a widely-used method for partitioning a dataset into a set of distinct [21], non-overlapping subgroups or clusters. KMeans aims to minimize the within-cluster sum of squares, ensuring that data points in the same cluster are close to each other while being far from points in other clusters [21]

The clustering of data points into distinct groups was achieved using the KMeans clustering algorithm. KMeans works by partitioning the dataset into a predefined number of clusters. It does this by minimizing the sum of squared distances between data points and their respective cluster centroids. The centroids are recalculated iteratively until the algorithm converges to an optimal solution [21]. In this analysis, the majority of the data points, approximately 99.76 million, were categorized under Cluster 0. Cluster 3 followed with around 20.12 million data points, Cluster 1 had 3.62 million, and Cluster 2 was the smallest with just over 60,000.This Table3.5 details the key characteristics of each clustering center.

### 1.5.1 Cluster Characteristics

**Table 3.5:** Cluster data.

| Cluster | Count | node_load15 | node_load5 | node_load1 | node_memory | Anomalies |
|---|---|---|---|---|---|---|
| 0 | $9.98 \times 10^7$ | 6.64 | 6.61 | 6.60 | $4.93 \times 10^9$ | 0.00 |
| 1 | $3.62 \times 10^6$ | 7.99 | 8.02 | 8.03 | $3.16 \times 10^{10}$ | 0.03 |
| 2 | $6.05 \times 10^4$ | 1,946.39 | 1,950.55 | 1,951.67 | $2.23 \times 10^9$ | 1.00 |
| 3 | $2.01 \times 10^7$ | 27.42 | 27.69 | 27.75 | $3.89 \times 10^{10}$ | 0.17 |

**Cluster 0** : This cluster predominantly consists of nodes with lower loads and memory activities. The nodes here exhibited a 15-minute load average of 6.64, a 5-minute load average of 6.61, and a 1-minute load average of 6.60. The average active memory stood at about 4.93 GB.

**Analysis:** This cluster seems to represent the "baseline" or "normal" behavior of the system, with low loads and memory usage. The absence of anomalies suggests that these nodes are operating within expected parameters. However, the low memory usage might indicate underutilization, which could be investigated for resource optimization.

**Cluster 1** : Nodes in this cluster demonstrated slightly elevated loads compared to Cluster 0. The average loads over 15 minutes, 5 minutes, and 1 minute were 7.99, 8.02, and 8.03, respectively. Notably, the active memory was significantly higher, averaging around 31.59 TB.

**Analysis:** The elevated load and significantly higher memory usage suggest that these nodes are handling more complex or numerous tasks. The slight presence of anomalies could indicate that this cluster is more susceptible to issues and might benefit from closer monitoring.

**Cluster 2** : This cluster is particularly intriguing. It showcased a very high average load over 15 minutes (1946.39), 5 minutes (1950.55), and 1 minute (1951.67). Despite these high loads, the memory activity was relatively low, averaging around 2.23 TB. Furthermore, this cluster had a 100% anomaly rate, pinpointing a potential area of concern.

**Analysis:** The extremely high load averages and 100% anomaly rate make this cluster a critical focus for further investigation. The low memory usage despite high loads could indicate bottlenecks or misconfigurations that are leading to inefficiencies and anomalies.

**Cluster 3** : Nodes in this cluster displayed moderate loads with averages of 27.42, 27.69, and 27.75 over 15 minutes, 5 minutes, and 1 minute, respectively. The active memory was substantial, averaging around 38.94 TB.

**Analysis:** This cluster seems to represent a "middle ground" between the low-load Cluster 0 and the high-load Cluster 2. The moderate but substantial memory usage suggests that these nodes are well-utilized but not overburdened. The presence of some anomalies could indicate that there are occasional issues that might require attention.

### 1.5.2 K-means Clustering Insights

From the provided analysis, it's evident that while a majority of nodes operate under standard conditions (as seen in Clusters 0 and 1), there are specific nodes, especially within Cluster 2, that appear to be under significant stress. Cluster 3 represents nodes with moderate loads but heightened memory activity. It is imperative to identify and monitor the nodes that frequently fall into Cluster 2, as they could be crucial for preventive maintenance or to ensure they don't evolve into performance bottlenecks.

## 1.6 Analysis and Comparations of GPU Nodes and non-GPU Nodes

PROMETHEUS data provides insights into the performance and utilization of nodes, both with and without GPUs. This section delves into the descriptive statistics, extreme values, and clustering results of these nodes. The Table 1.6 shows the statistics of nodes with and without GPU

**Table 3.6:** Statistics for nodes with and without GPU.

| | mean | std | 25% | 50% | 75% | max |
|---|---|---|---|---|---|---|
| | | | Nodes with GPU | | | |
| Node Load15 | 5.05 | 10.94 | 0.02 | 2.20 | 4.98 | 482.62 |
| Node Load5 | 5.07 | 11.11 | 0.04 | 2.16 | 4.94 | 803.62 |
| Node Load1 | 5.07 | 11.29 | 0.02 | 2.12 | 4.87 | 1967.07 |
| Memory Active | $1.89 \times 10^{10}$ | $2.55 \times 10^{10}$ | $2.47 \times 10^{9}$ | $1.17 \times 10^{10}$ | $2.47 \times 10^{10}$ | $2.42 \times 10^{11}$ |
| GPU Power Max | $1.34 \times 10^{5}$ | $1.10 \times 10^{5}$ | $3.18 \times 10^{4}$ | $7.97 \times 10^{4}$ | $2.67 \times 10^{5}$ | $4.22 \times 10^{5}$ |
| GPU Temp Max | 57.56 | 22.39 | 36.00 | 50.00 | 84.00 | 93.00 |
| IsAnomaly | 0.02 | 0.15 | 0.00 | 0.00 | 0.00 | 1.00 |
| | | | Nodes without GPU | | | |
| Node Load15 | 11.40 | 52.09 | 0.06 | 6.01 | 16.01 | 4905.28 |
| Node Load5 | 11.41 | 52.32 | 0.07 | 5.41 | 16.03 | 4908.67 |
| Node Load1 | 11.42 | 52.46 | 0.08 | 5.02 | 16.00 | 4910.27 |
| Memory Active | $1.08 \times 10^{10}$ | $1.60 \times 10^{10}$ | $1.57 \times 10^{9}$ | $3.71 \times 10^{9}$ | $1.36 \times 10^{10}$ | $2.10 \times 10^{12}$ |
| IsAnomaly | 0.03 | 0.17 | 0.00 | 0.00 | 0.00 | 1.00 |

### 1.6.1 Descriptive Statistics

**Nodes with GPU:** The load averages, namely `node_load15`, `node_load5`, and `node_load1`, provide insights into the system load over the last 15, 5, and 1 minutes respectively. The mean values for these loads are approximately 5, with some extreme outliers. Active memory on these nodes averages around 18 GB, with a large variance. GPU power usage and temperature also exhibit significant variances. Only about 2% of the data points are labeled as anomalies.

**Nodes without GPU:** These nodes have significantly higher load averages, averaging around 11.4. Memory utilization is approximately half of that in nodes with GPUs. As expected, GPU metrics are absent. Anomalies constitute a slightly higher percentage of the data at 3%.

### 1.6.2 Extreme Values

**Nodes with GPU:** Node `r38n4` consistently shows high values for load and memory, suggesting it's heavily utilized. Conversely, nodes like `r28n5` and `r35n3` indicate lesser utilization.

**Nodes without GPU:** Node `r11n29` consistently exhibits high values for load metrics. Several metrics have uniform values across these nodes, such as GPU metrics, which are expectedly zero.

### 1.6.3 K-means Clustering

Using the k-means clustering algorithm, as previously mentioned, nodes were segmented based on their resource utilization patterns. The Table 3.7 shows cluster statistics.

**Nodes with GPU:** Cluster 1 is the largest, representing nodes with low resource utilization. Cluster 0 and Cluster 3 represent nodes with moderate resource usage, while Cluster 2 represents heavily burdened nodes, often flagged as anomalies.

**Nodes without GPU:** Cluster 0 dominates, representing nodes with moderate resource utilization. Cluster 3 and Cluster 2 represent nodes with increasing levels of resource usage, while Cluster 1 represents the most heavily loaded nodes, often flagged as anomalies.

**Summary:** Nodes equipped with GPUs exhibit varied utilization patterns, with some nodes being heavily loaded and others showing moderate to low loads. Memory and GPU usage also vary significantly. Nodes without GPUs tend to have higher load averages. The

**Table 3.7:** Cluster statistics for nodes.

| Cluster | Count | Load15 | Load5 | Load1 | Memory | Power Max | Temp Max |
|---|---|---|---|---|---|---|---|
| Nodes with GPU | | | | | | | |
| 0 | $2.69 \times 10^6$ | 5.00 | 4.99 | 4.98 | $2.36 \times 10^{10}$ | $2.52 \times 10^5$ | 81.57 |
| 1 | $3.94 \times 10^6$ | 1.26 | 1.25 | 1.24 | $6.03 \times 10^9$ | $4.07 \times 10^4$ | 38.09 |
| 2 | $1.50 \times 10^5$ | 67.90 | 68.74 | 69.11 | $3.81 \times 10^{10}$ | $1.59 \times 10^5$ | 67.91 |
| 3 | $6.25 \times 10^5$ | 14.07 | 14.20 | 14.23 | $7.50 \times 10^{10}$ | $2.15 \times 10^5$ | 74.40 |
| Nodes without GPU | | | | | | | |
| 0 | $9.62 \times 10^7$ | 6.85 | 6.82 | 6.81 | $4.96 \times 10^9$ | 0.00 | 0.00 |
| 1 | $5.68 \times 10^4$ | 2006.42 | 2010.72 | 2011.86 | $2.27 \times 10^9$ | 0.00 | 0.00 |
| 2 | $3.34 \times 10^6$ | 107.98 | 109.39 | 109.64 | $3.36 \times 10^{10}$ | 0.00 | 0.00 |
| 3 | $1.66 \times 10^7$ | 11.45 | 11.48 | 11.51 | $3.99 \times 10^{10}$ | 0.00 | 0.00 |

clusters have effectively segmented the nodes based on their resource utilization patterns. Monitoring and managing the heavily loaded nodes, especially those flagged as anomalies, should be prioritized to ensure stable system performance.

## 1.7 Time Block Analysis

In the context of our dataset, a time block refers to a continuous segment of time during which the PROMETHEUS system was operational and collecting data. These time blocks are separated by periods of interruption, where no data was collected. These interruptions pose a unique challenge in the analysis of our dataset.

The distribution of time blocks in the dataset is not uniform. Figure 3.1, which was previously introduced to illustrate the distribution of anomalies, also provides insights into the continuity of the data. In this heatmap, In this heatmap, the presence of blue color indicates that the majority of the data for that day is continuous. Conversely, white areas signify days where the majority of data collection was interrupted. This visualization provides a clear overview of the distribution of time blocks and interruptions in the dataset.

Figure 3.1 revealed that the highest number of interruptions occurred in July 2022, with 5,479 interruptions affecting 280 nodes. The average interruption duration was 6,149 seconds, with the longest interruption lasting 85,140 seconds and the shortest lasting 60 seconds. The node with the longest interruption time was r13n3 (449,580 seconds), while the node with the shortest interruption time was r35n5 (65,190 seconds).

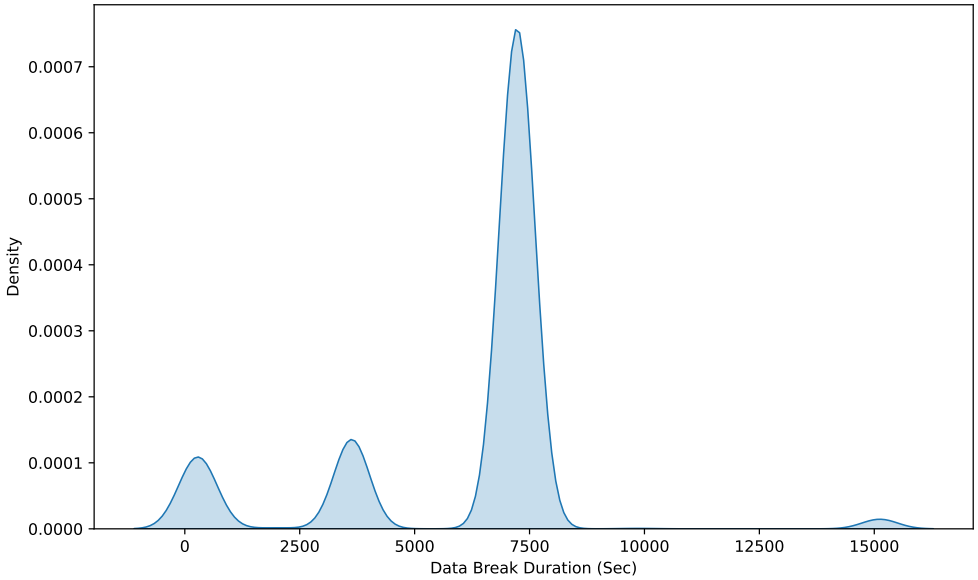


**Figure 3.3:** Density of data break duration.

A box plot was also created to visualize the distribution of all time block interruptions. The plot revealed that the majority of interruptions (approximately 73.64%) lasted 7,230 seconds. The remaining data points were sparsely distributed between 50,000 and 80,000 seconds. These interruptions in data collection can potentially impact the performance of unsupervised anomaly detection models, especially those that are sensitive to the temporal sequence of the data, such as LSTM autoencoders. For such models, it is crucial to ensure that the input data is from a continuous time block.

Given the frequency of data collection (every 30 seconds), most interruptions are too long to be filled using interpolation methods. For models that are not sensitive to the temporal sequence of the data, such as the Isolation Forest model, these interruptions can be ignored. However, for models that are sensitive to the temporal sequence of the data, it is important to ensure that the input data is from a continuous time block.

In the future, it may be beneficial to explore deep learning-based anomaly detection methods for time-series data, as these methods are capable of learning representations of large-scale sequences in an unsupervised manner and identifying anomalies from the data. However, these methods often require domain knowledge for appropriate deployment and may not be suitable for all use cases.

# 2 Slurm Job Dataset

In this section, we are going to characterize the SLURM dataset.

## 2.1 Dataset Node-Level Introduction

The second dataset used in this study also originates from the LISA National Computing Cluster, hosted at SURFsara [10]. Unlike the first dataset, this one focuses on job-related information.

The data was collected by the SLURM Workload Manager [19], a free and open-source job scheduler for Linux and Unix-like kernels, used by many of the world's supercomputers and computer clusters. The dataset various metrics that provide insights into the job-related information in the HPC system. These metrics can be broadly categorized into the following groups:

- **Job Metrics**: These include metrics such as 'id', 'start_date', 'end_date', 'submit', 'start', 'end', 'state', and 'exitcode'. These metrics provide information about the job's ID, start and end dates, submission time, start and end times, state, and exit code.
- **Node Metrics**: These include 'node', 'nodetypes', 'numnodes', 'numcores', and 'sharednode'. These metrics provide information about the node or nodes on which the job was run, the type of nodes, the number of nodes and cores used, and whether the node was shared with other jobs.
- **Reservation Metrics**: These include 'researvation' and 'partprepaid'. These metrics provide information about any reservations associated with the job and whether a part of the job was prepaid.

Collectively, these metrics provide a detailed picture of the job-related information in the HPC system, making them valuable for understanding the job execution and performance.

Stored in the Parquet format, the dataset is significantly smaller than the first one, with a size of 52 MB. It comprises 1,596,965 rows and 15 columns, each row representing a unique job and each column representing a different attribute of the job.

Table 3.8 displays the description of important metrics in this dataset.

**Table 3.8:** Description of selected metrics in SLURM dataset.

| Metrics | Description |
|---|---|
| id | Job ID |
| start_date | Start date of job |
| end_date | End date of job |
| node | Rack and node of the job |
| nodetypes | Type of node |
| numnodes | Number of nodes used for the job |
| numcores | Number of cores used for the job |
| state | 'TIMEOUT', 'COMPLETED', 'CANCELLED', 'FAILED', 'OUT_OF_MEMORY', 'NODE_FAIL' |
| exitcode | Exit code of job |

## 2.2 Overall Analysis

Figure 3.4 provides a visualization of the overall job distribution across different nodes and dates. Each cell in the heatmap corresponds to a specific combination of a date (X-axis) and a node name (Y-axis), and the color intensity of each cell indicates the number of jobs executed on that particular node on that specific date.

The color mapping ranges from light green to dark green, where light green represents a relatively low number of jobs and dark green represents a high number of jobs. In other words, the darker the color, the higher the job count for that node on the corresponding date. This logarithmic representation enhances the color contrast and hence makes it easier to observe patterns and anomalies.

On average, the system managed by Slurm handles 5,134.93 jobs daily. Intriguingly, the busiest day was on June 14, 2022, with a staggering 162,004 jobs. In contrast, the least activity was observed on December 25, 2021, with only 2 jobs, possibly due to the Christmas holiday leading to a significant reduction in computational tasks.

## 2.3 Hourly Job Submissions

Figure 3.5 showcases a notable trend in job submissions throughout the day. From midnight until the early morning (0:00 to 3:00), we observe a relatively low number of job submissions, with the trough at 1.0% around 3:00. However, from 6:00 in the morning, there's a steep ascent, reaching a zenith at 15:00 with a whopping 190,682 submissions,

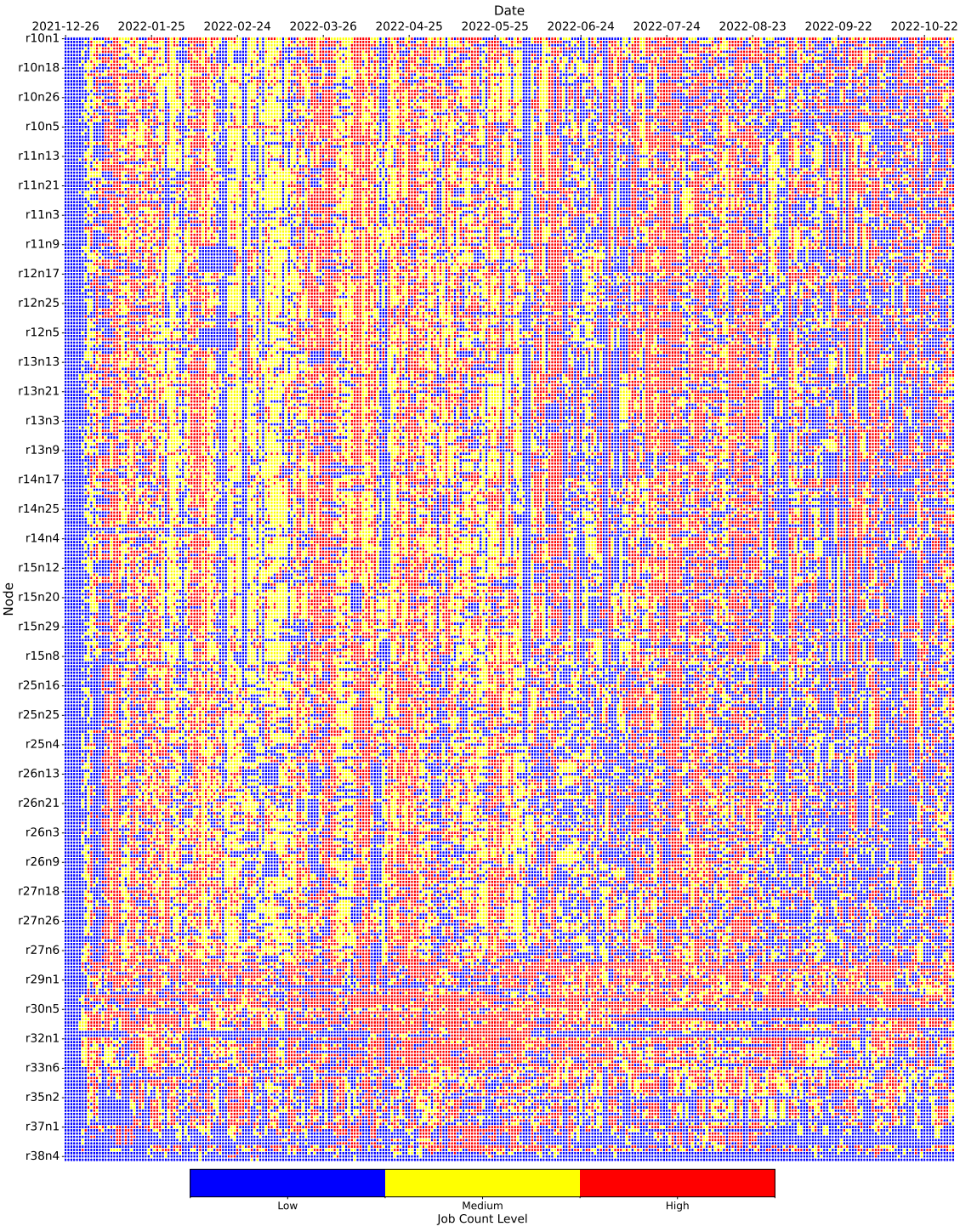


**Figure 3.4:** Node usage heatmap, blue indicates low workload level, yellow indicates medium, red indicates high.

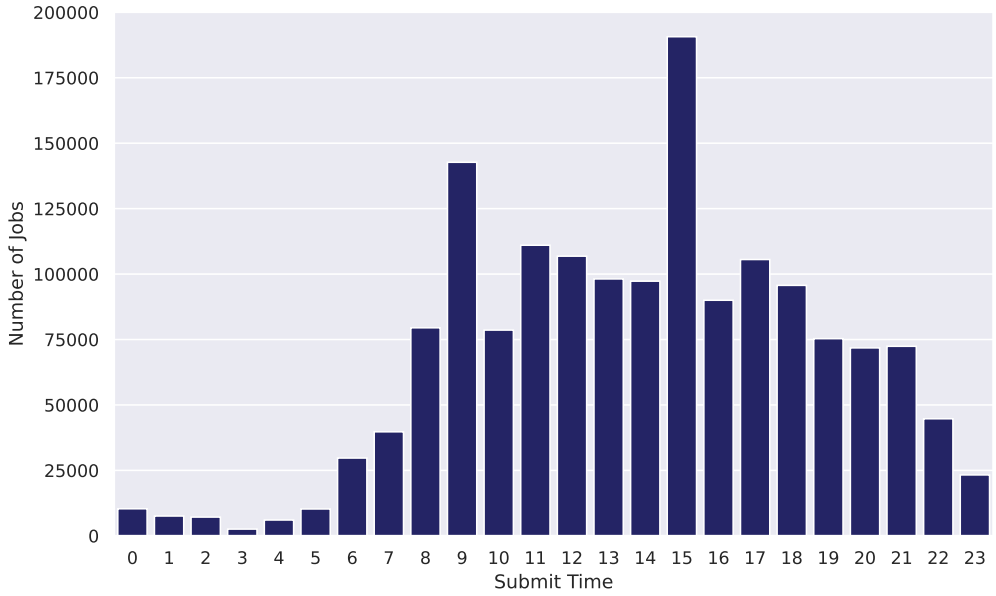


**Figure 3.5:** Job hourly submission.

accounting for 11.94% of the day's total. Subsequently, the numbers start to wane in the evening, hitting the trough again in the early morning.

These patterns could be influenced by the user behaviors, such as initiation of computational tasks in the morning and a reduced activity during nighttime. Such insights are critical for system administrators to optimize resource allocation during peak times.

## 2.4 Job State Distribution

Examining the state distribution of the jobs in Figure 3.6, the majority of them were found to be completed successfully, accounting for approximately 74.96% of the total jobs. This translates to 1,197,035 successful job completions out of the total 1,596,965 tasks. However, failed jobs aren't rare, with a significant 15.36% experiencing failure. Other states like cancelled and timeout constituted 6.13% and 2.88% respectively. Jobs that ran out of memory were comparatively fewer, approximately 0.57%, and node failures were the least common, with a minimal 0.11%.

This information underscores the efficiency and reliability of the system managed by Slurm, but also pinpoints areas of improvement, especially in mitigating the number of failed jobs.

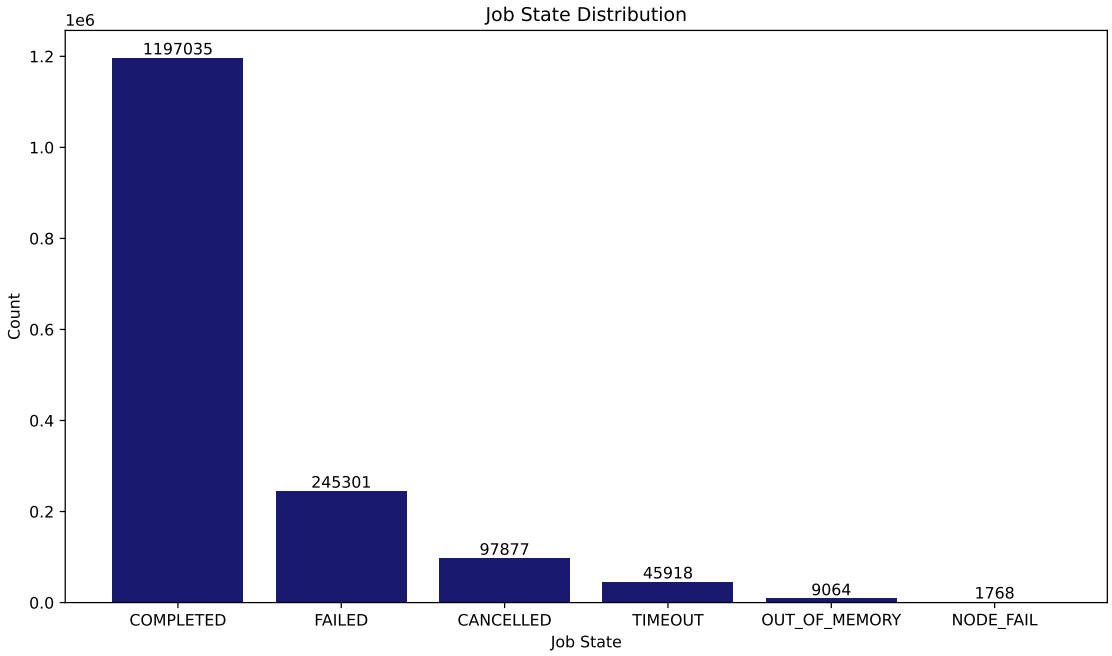


**Figure 3.6:** Job state distribution.

## 2.5 Node Analysis

The node analysis in Figure 3.7 provides insight into the performance of specific nodes. Among all, nodes ‘r37n1‘, ‘r37n2‘, ‘r37n3‘, ‘r37n4‘, and ‘r10n30‘ received the most job counts, ranging from 16,221 to 30,502. Their peak times mostly fell in May and August 2022, suggesting heightened computational activities during these periods.

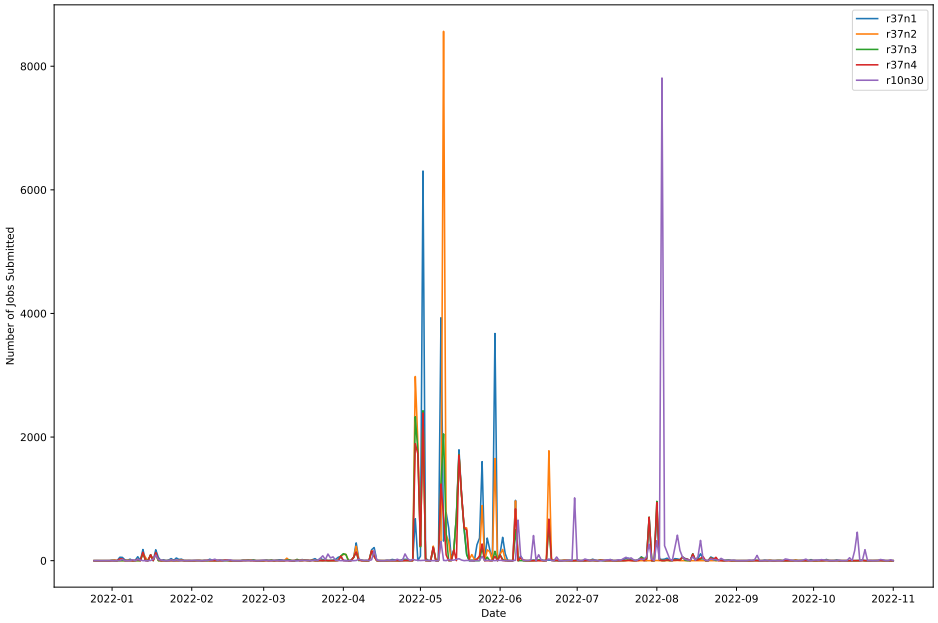


**Figure 3.7:** The number of submitted jobs on the top 5 nodes by date.

On the other hand, Figure 3.8 shows that nodes ‘r30n5‘, ‘r15n32‘, ‘r38n3‘, ‘r38n4‘, and ‘r38n5‘ experienced the least job counts, ranging from just 18 to 1,041. Their success rates were unfortunately found to be zero, indicating potential issues with these nodes that demand further investigation by the system administrators.

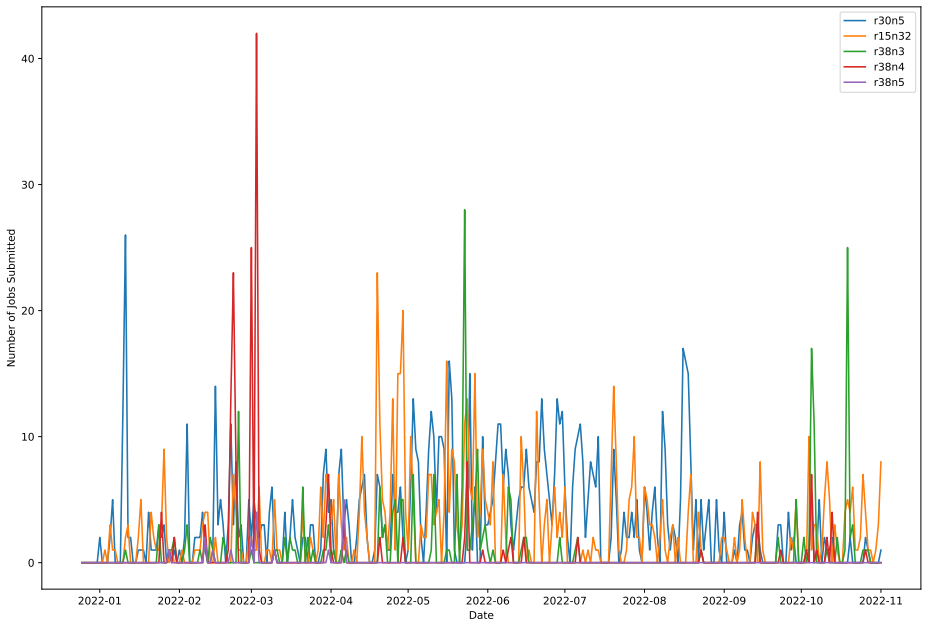


**Figure 3.8:** Bottom 5 node submission.

In conclusion, this Slurm job data analysis provides invaluable insights into the job submission patterns, job states, daily job counts, and performance on a node-specific level. These findings can guide optimizations, resource allocation, and troubleshooting efforts, ensuring the seamless operation of the computing system managed by Slurm.

# 3 Additional PROMETHEUS Dataset

In this section, we turn our attention to characterizing an additional PROMETHEUS dataset.

## 3.1 Dataset Introduction

The third dataset employed in this study is also sourced from the LISA National Computing Cluster in the Netherlands, similar to the first dataset. This dataset, however, has some distinct characteristics that set it apart from the initial one.

The dataset encompasses time-series data from 341 nodes. The data collection started on December 31, 2019, at 23:00:00 and concluded on April 2, 2020, at 21:59:45, covering

a span of 92 days, 22 hours, and 59 minutes. In total, the dataset comprises 180,110,880 rows of data.

Unlike the first dataset, where all metrics are stored in a single Parquet file, the metrics in this dataset are stored in individual Parquet files. This structure necessitates additional preprocessing steps to transform the data into a format consistent with the first dataset.

However, a notable challenge arises when dealing with GPU-related metrics. The format of these metrics in this dataset differs from that in the first dataset. Moreover, the exact format of these GPU metrics remains unknown, making it infeasible to convert them to match the format of the first dataset. Given this limitation, two GPU-related columns were excluded from characterization and the Experiment 4.

The inclusion of this additional dataset provides an opportunity to validate the robustness and generalizability of the models trained on the first dataset. By testing on a dataset with different structural characteristics and a slightly different time span, we aim to ascertain the adaptability of the models to diverse data scenarios. This is particularly crucial in real-world settings where data might come from varied sources and in different formats.

## 3.2 Statistical Analysis

The Table 3.9 shows the overall statistics.

**Table 3.9:** Summary statistics.

| Metric | Mean | Std. Dev. | 25% | 50% | 75% | Max |
|---|---|---|---|---|---|---|
| node_load15 | 14.06 | 46.44 | 2.00 | 12.85 | 16.00 | 24,182.98 |
| node_load5 | 14.10 | 51.03 | 2.00 | 13.06 | 16.02 | 27,593.61 |
| node_load1 | 14.11 | 53.44 | 1.95 | 13.43 | 16.00 | 27,885.24 |
| node_memory Active_bytes | $1.25 \times 10^{10}$ | $2.97 \times 10^{10}$ | $1.56 \times 10^{9}$ | $4.80 \times 10^{9}$ | $1.32 \times 10^{10}$ | $2.15 \times 10^{12}$ |

### 3.2.1 Node Memory Active

The `node_memory_Active_bytes` metric, representing active memory bytes, averaged around 12.51 TB. The standard deviation of 29.68 TB indicates a wide spread in memory usage, ranging from a minimum of 23.95 MB to a substantial 2.15 PB.

**Analysis:** The wide range and high standard deviation in active memory usage suggest that there are nodes with vastly different memory requirements. This could indicate a

heterogeneous workload or possibly point to specific nodes that are either underutilized or overburdened. The extreme values may warrant further investigation for anomalies or optimization opportunities.

### 3.2.2 Node Load Over Time

- **15 minutes (node_load15)**: The mean load was 14.06, with a standard deviation of 46.44. The load values ranged from a minimum of 0.01 to a maximum of 24182.98.
- **5 minutes (node_load5)**: The average load over this interval was 14.10, with a standard deviation of 51.03. The observed load values spanned from 0.01 to 27593.61.
- **1 minute (node_load1)**: The immediate load average was 14.11, with a standard deviation of 53.44. The highest observed value was 27885.24.

  **Analysis:** The high standard deviation and wide range suggest significant variability among nodes. Nodes with extremely high or low values could be experiencing issues or may be candidates for resource reallocation.

## 3.3 Anomaly Analysis

After applying the previously mentioned thresholds to the Additional PROMETHEUS dataset, a total of 3,814,028 anomalies were identified. This represents approximately 2.91% of the total data points in this dataset. The standard deviation of 0.17 suggests that these anomalies are not uniformly distributed, either across nodes or temporally.

Figure 3.9 presents a heatmap encapsulating all the time series data from the Additional PROMETHEUS dataset, aggregated on a daily basis. The x-axis represents the date, while the y-axis denotes the names of the nodes, specifically the days. In the heatmap, the color of each cell represents the level of anomalies detected for a particular node on a given day. Blue signifies a low level of anomalies, yellow indicates a medium level, and a deep, intense red represents a high number of anomalies, signaling a day with more irregularities in the node's operation.

## 3.4 Time Block Analysis

In the context of the Additional PROMETHEUS dataset, the continuity and distribution of time blocks remain a significant aspect of our analysis. As visualized in Figure 3.9, the presence of blue color signifies that most of the data for that day is continuous. On the

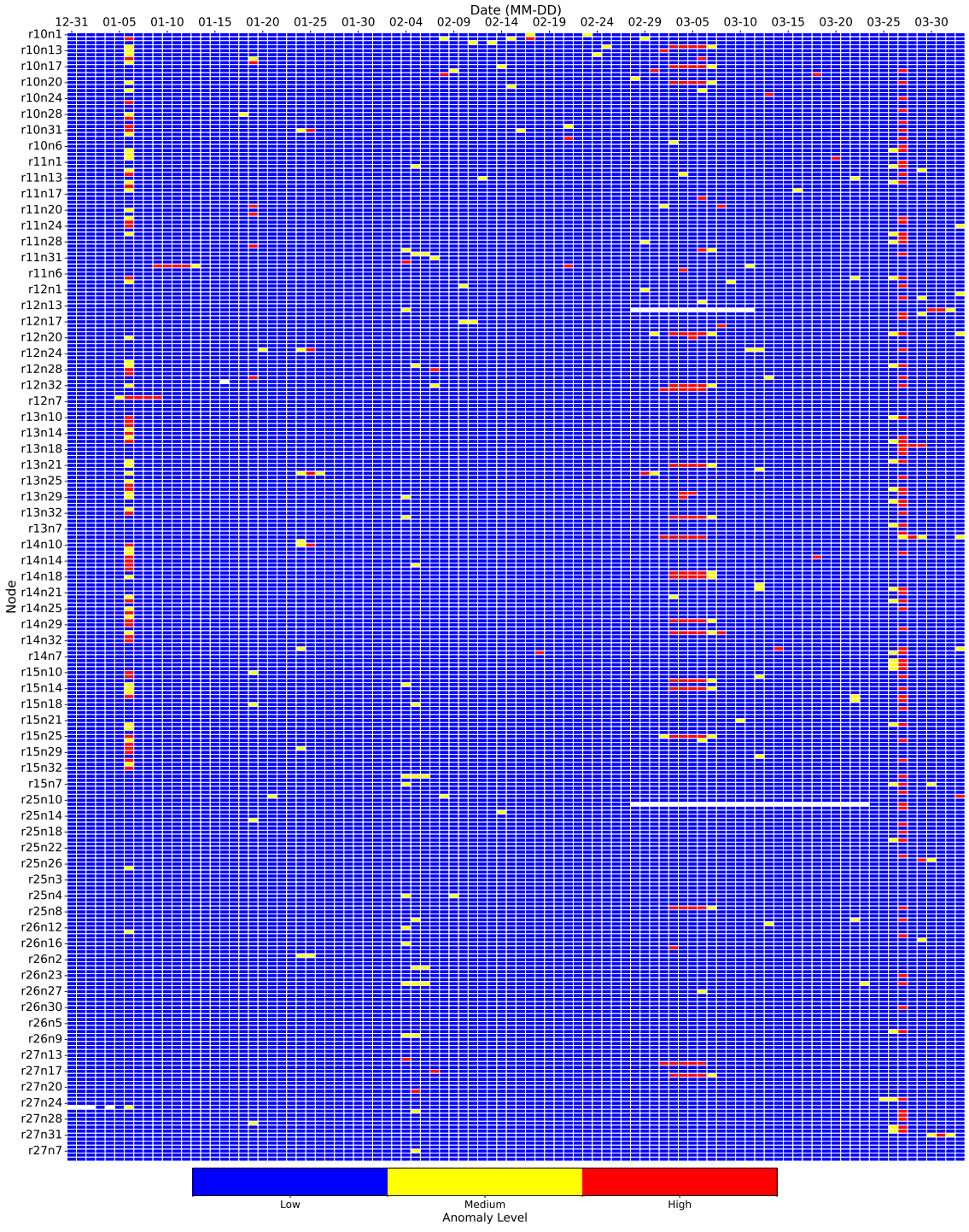


**Figure 3.9:** Time-series heatmap with anomalies for the additional PROMETHEUS dataset, blue indicates low anomaly level, yellow indicates medium and red indicates high.

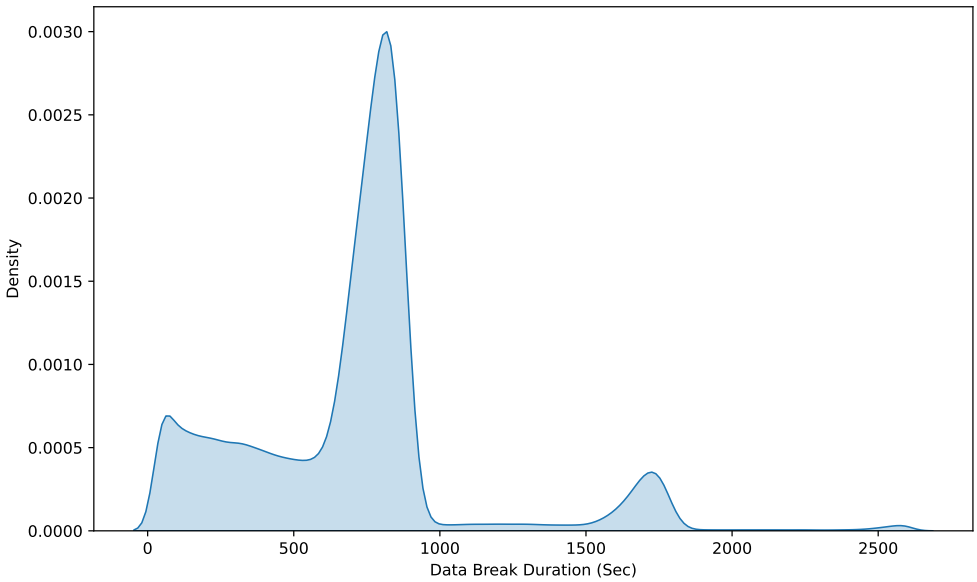


**Figure 3.10:** Densityplot of data break duration for the 2020 PROMETHEUS dataset.

other hand, white areas denote days where the majority of the data collection experienced interruptions.

The month-wise breakdown of interruptions, as shown in Table 3.10, revealed that January 2020 experienced the highest number of interruptions, affecting 283 nodes. The average interruption duration was 6272.3 seconds, with the longest interruption lasting 2249625 seconds and the shortest lasting 45 seconds. The node with the longest interruption time was r12n30, while the node with the shortest interruption time was r10n4.

**Table 3.10:** Combined breakdown and anomaly data.

| Month | Breakdown Count | Affected Node Count | Anomaly Count |
|---|---|---|---|
| 2022-06 | 287 | 280 | 25,637 |
| 2022-07 | 5479 | 280 | 437,003 |
| 2022-08 | 1513 | 319 | 617,376 |
| 2022-09 | 2763 | 319 | 146,964 |
| 2022-10 | 2053 | 319 | 1,449,874 |
| 2022-11 | 1922 | 318 | 881,725 |

The majority of interruptions, approximately 51.43%, lasted between 465 and 825 sec-

onds. These interruptions can potentially impact the performance of unsupervised anomaly detection models, especially those sensitive to the temporal sequence of the data, such as LSTM autoencoders. Given the frequency of data collection in this dataset, most interruptions are too long to be filled using interpolation methods. For models that are not sensitive to the temporal sequence of the data, these interruptions can be ignored. However, for models that are sensitive to the temporal sequence of the data, it is crucial to ensure that the input data is from a continuous time block.

# 4 Comparison First and Additional Prometheus Datasets

Upon closely examining both datasets, several key differences and similarities emerge:

## 4.1 Duration and Size

The first dataset spans a longer duration of 144 days, 19 hours, and 20 minutes, while the additional dataset covers 92 days, 22 hours, and 59 minutes. This difference in duration might influence the total number of anomalies detected in each dataset. Additionally, there is some slight difference in the number of nodes (343 in the first dataset and 341 in the additional dataset).

## 4.2 Node Memory Active

The additional dataset exhibits a higher average active memory bytes (12.51 TB) compared to the first dataset (11.25 TB). This represents an increase of approximately 11.2%. This could indicate more intensive memory usage patterns in the period covered by the additional dataset.

## 4.3 Anomaly Rate

Both datasets have a similar anomaly rate, with the first dataset at 2.88% and the additional dataset slightly higher at 2.91%. This consistency suggests that the anomaly detection thresholds applied were effective across different time periods and data densities.

## 4.4 Average Interruption Duration

The additional dataset had a slightly longer average interruption duration (6272.3 seconds) compared to the first dataset (6149 seconds). This could suggest more prolonged periods of system downtime or data collection issues during the additional dataset's timeframe.

Both datasets, while sourced from the same computing cluster and collected using the PROMETHEUS system, exhibit unique characteristics. These differences underscore the dynamic nature of HPC systems and the challenges in ensuring consistent data collection and analysis.

# 5 Summary

In this chapter, we embarked on a comprehensive journey into the realm of data characterization and integration. The essence of our exploration revolved around understanding the intricacies of our dataset, its nuances, and the inherent patterns that could be leveraged for effective anomaly detection.

The process of data characterization was not merely about understanding the statistical properties of the data but delving deeper into its structure, relationships, and potential significance in the broader context of our research. We meticulously analyzed various features, understanding their distributions, correlations, and potential impact on the outcomes of our anomaly detection models.

Integration, on the other hand, was about ensuring that our dataset was cohesive, consistent, and ready for the subsequent stages of our research. One of the pivotal steps in this phase was the marking of anomalies based on certain thresholds. This step, although seemingly straightforward, was crucial in setting the stage for our detection algorithms, ensuring that they had a clear benchmark against which to measure deviations.

In summary, this chapter laid the groundwork for our research, ensuring that we had a robust, well-understood, and effectively integrated dataset to work with.

# Chapter 4

# Data Integration

This chapter is dedicated to addressing Research Question 2: "How can node data be integrated with job data for the purpose of anomaly detection?" We will explore the methodologies and techniques employed for integrating SLURM job dataset with the PROMETHEUS time-series dataset.

## 1 Integration Background

The integration of the SLURM job dataset and the PROMETHEUS time-series dataset is a key step in this study. This integration aims to leverage the job-related information in the SLURM dataset to enhance the anomaly detection performance of the machine learning models. This is one of the innovative points of this thesis, as most existing studies do not incorporate job information into anomaly detection.

The motivation for integrating these two datasets is twofold. First, the SLURM dataset contains valuable information about the jobs running on the nodes, including whether a job was executed successfully or not. This information can potentially help the machine learning models to better identify anomalies. Second, the integration of these two datasets allows us to create a more comprehensive view of the system's operation, combining both the time-series data from the PROMETHEUS dataset and the job-related data from the SLURM dataset. Figure 4.1 shows an example of data integration. The integration process involves aligning the time-series data from the PROMETHEUS dataset with the job data from the SLURM dataset. For each job in the SLURM dataset, the corresponding time-series data from the PROMETHEUS dataset within the job's start and end times are selected. The node names in the two datasets are also matched to ensure that the data from the same node are integrated.

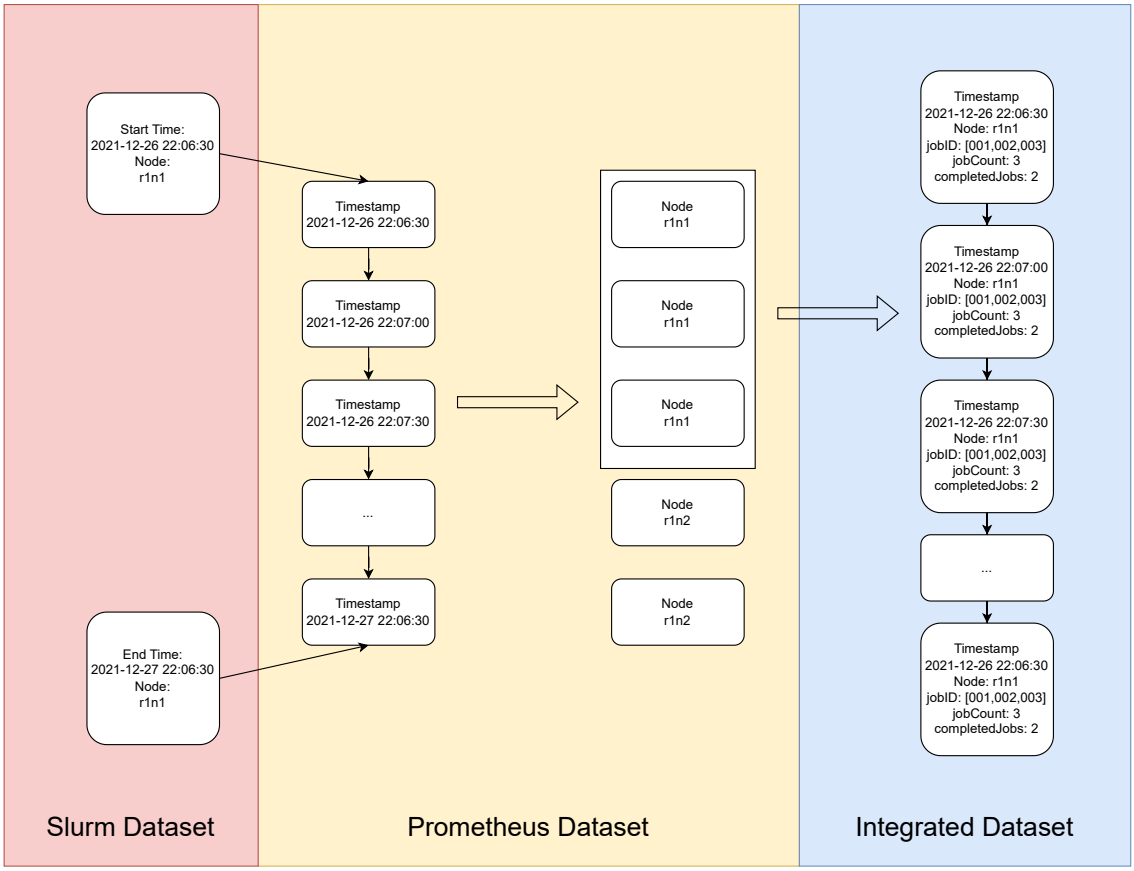


**Figure 4.1:** Example of the process of data integration.

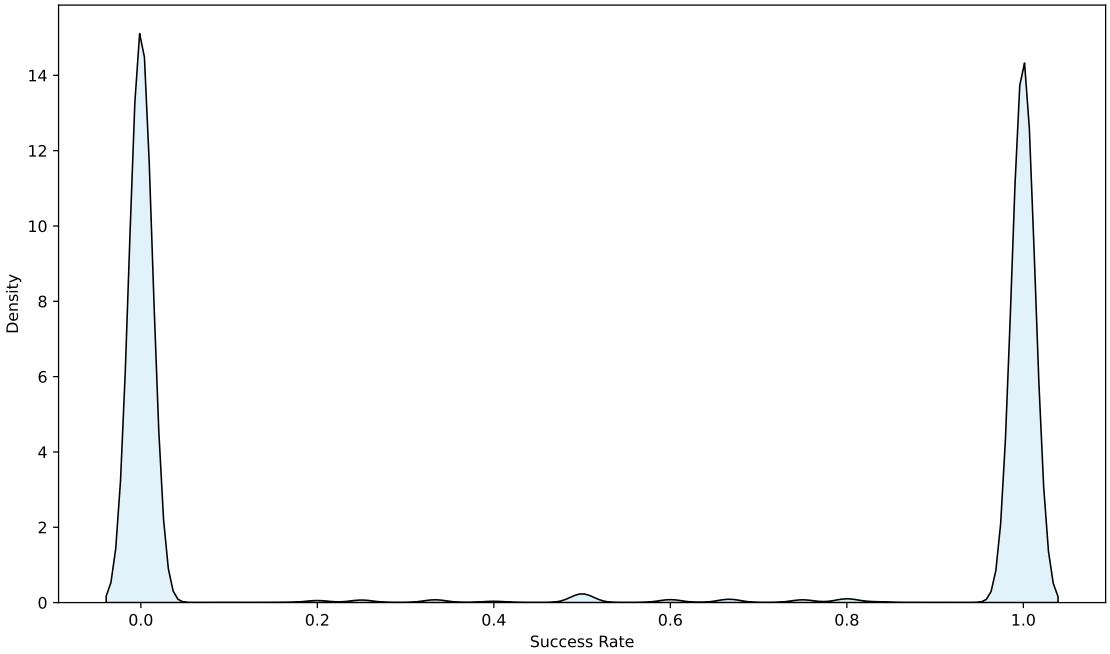


**Figure 4.2:** Success rate density plot.

One of the challenges encountered during the integration process is the complex structure of the node information in the SLURM dataset. The node information can appear in several formats, such as single letters and numbers (e.g., r10n19), letters and numbers with brackets and numbers (e.g., r12n[3,9]), and letters and numbers with brackets and range numbers (e.g., r15n[18-24,26,28,31]). To handle this complexity, a Python function was written to parse the node information and standardize it into a consistent format.

The integration process also involves the addition of new features to the original PROMETHEUS dataset. Three new features were added, representing the number of jobs running, the number of jobs completed, and the success rate of the running nodes in the last 30 seconds. These features provide additional context about the system's operation and can potentially help the machine learning models to better identify anomalies.

The integrated dataset will be used in the subsequent machine learning model training and evaluation. The aim is to verify whether the integration of job data can enhance the performance of the machine learning models in anomaly detection.

## 2 Integrated Data Analysis

The integrated dataset offers a unique perspective by combining both time-series metrics and job-related information. One of the key insights derived from the integrated data pertains to nodes that run multiple jobs simultaneously.

For nodes that are executing multiple tasks concurrently, the average success rate is observed to be 0.49. This indicates that, on average, about half of the jobs on these nodes are successful. However, this average figure might be misleading when we delve deeper into the distribution of success rates.

Figure 4.2 was constructed to visualize the distribution of success rates for these nodes. The y-axis represents the number of nodes, while the x-axis spans from 0 to 100, indicating the success rate in percentage terms. Intriguingly, the density plot reveals a bimodal distribution. The data predominantly clusters around the two extremes of the success rate spectrum. This means that nodes tend to either have a very high success rate (close to 100%) or a very low success rate (close to 0%).

Such a distribution suggests that nodes often operate in a binary mode: they either execute most of their jobs successfully or face consistent failures across the board. This insight is crucial as it underscores the importance of understanding the underlying reasons for such consistent successes or failures. It could be attributed to hardware reliability, software configurations, or even the nature of the jobs being executed.

This bimodal distribution also has implications for anomaly detection. Nodes that consistently fail might be easier to flag as anomalous. However, discerning anomalies in nodes that have a mixed bag of successes and failures might be more challenging and would require models to consider more nuanced patterns and dependencies.

In summary, the integrated data analysis sheds light on the operational patterns of nodes, emphasizing the need for a more granular understanding of success rates and their implications for anomaly detection.

## 3 Summary

In this chapter, we have explored the critical role of data integration in enhancing the performance of machine learning models for anomaly detection in HPC systems. The integration of the SLURM job dataset with the PROMETHEUS time-series dataset serves as an innovative approach, offering a more comprehensive view of the system's operation.

This integration not only enriches the feature set but also provides valuable context that can aid in more accurate anomaly detection.

We also delved into the complexities and challenges of the integration process, including the need for feature engineering and the standardization of node information. The addition of new features, such as the number of running jobs, completed jobs, and success rates, further enriches the dataset and provides additional dimensions for analysis.

Our integrated data analysis revealed intriguing patterns, particularly the bimodal distribution of success rates among nodes running multiple jobs. Overall, the integrated dataset serves as a robust foundation for the subsequent stages of machine learning model training and evaluation, aiming to verify the effectiveness of incorporating job-related information into anomaly detection.

# 4 Unsupervised Machine Learning Models for Anomaly Detection

This section aims to provide an in-depth discussion of the unsupervised machine learning models used for anomaly detection in HPC clusters. Specifically, we will delve into the intricacies of four key models: Isolation Forest (iForest), Autoencoder, Long Short-Term Memory Autoencoder (LSTM-Autoencoder), and K-Means clustering. Each model will be evaluated based on its performance in detecting anomalies.

## 4.1 Isolation Forest (iForest)

Isolation Forest, commonly abbreviated as iForest, is an anomaly detection algorithm that operates on the principle of isolating anomalies rather than profiling normal data points. The core idea behind this approach is that anomalies are few and different, which should make them easier to isolate compared to normal points [11].

Given a dataset $D$ of size $N$, the algorithm works by constructing an ensemble of isolation trees (iTrees). Each tree is constructed as follows:

1. If the number of samples in $D$ is less than a threshold $e$, or the tree depth reaches a limit $l$, the tree construction ends.

2. Randomly select a feature $q$ and a split value $p$ between the minimum and maximum values of $q$.

3. Partition the dataset into two subsets $D_1$ and $D_2$ such that $D_1$ contains all samples with values below $p$ for feature $q$, and $D_2$ contains the rest.

4. Recursively repeat the above steps for $D_1$ and $D_2$.

The anomaly score $s(x, n)$ for a data point $x$ in a tree of size $n$ is defined as [11]:

$$s(x, n) = 2^{-\frac{E(h(x))}{c(n)}}$$

Where $E(h(x))$ is the average path length of unsuccessful searches in a Binary Search Tree, and $c(n)$ is the average path length of unsuccessful search in a BST. The score is a value between 0 and 1, with larger scores indicating that $x$ is more likely to be an anomaly.

The key advantages of iForest include:

- **Efficiency:** iForest has a linear time complexity with a low constant and a low memory requirement.
- **Scalability:** The algorithm is particularly suited for high-dimensional datasets.
- **Interpretability:** The anomaly score produced by iForest can be traced back to the structure of the iTrees, providing insights into the reasons for a particular score.

### 4.2 LSTM Autoencoder

Long Short-Term Memory (LSTM) networks, a subtype of Recurrent Neural Network (RNN), are particularly adept at handling sequences due to their internal memory structures. An autoencoder, on the other hand, is designed for unsupervised learning of efficient codings. The primary objective of an autoencoder is to learn a representation (encoding) for a set of data, typically for dimensionality reduction. When combined, an LSTM Autoencoder becomes a powerful tool for time series anomaly detection.

The LSTM cell is defined by the following equations [7]:

$$\begin{aligned}
f_t &= \sigma(W_f \cdot [h_{t-1}, x_t] + b_f) && \text{(Forget Gate)} \\
i_t &= \sigma(W_i \cdot [h_{t-1}, x_t] + b_i) && \text{(Input Gate)} \\
\tilde{C}_t &= \tanh(W_C \cdot [h_{t-1}, x_t] + b_C) && \text{(New Cell State)} \\
C_t &= f_t \times C_{t-1} + i_t \times \tilde{C}_t && \text{(Final Cell State)} \\
o_t &= \sigma(W_o \cdot [h_{t-1}, x_t] + b_o) && \text{(Output Gate)} \\
h_t &= o_t \times \tanh(C_t) && \text{(Hidden State)}
\end{aligned}$$

Where: - $f_t$, $i_t$, $o_t$ are the forget, input, and output gates respectively. - $h_t$ is the hidden state. - $C_t$ is the cell state. - $W$ and $b$ are the weights and biases for each gate. - $\sigma$ is the sigmoid activation function. - $\times$ denotes element-wise multiplication.

The LSTM Autoencoder structure can be summarized as:

1. **Encoding:** The LSTM layers capture the temporal patterns in the data. This is achieved by the LSTM's ability to remember long-term dependencies, which allows it to recognize patterns over extended sequences.

2. **Bottleneck:** A dense layer that provides a compressed representation of the input data, encapsulating its most crucial features.

3. **Decoding:** The LSTM layers in the decoder aim to reconstruct the original data from the compressed representation.

The reconstruction error, typically the Mean Squared Error (MSE) between the original and the reconstructed sequences, serves as a metric for anomaly detection:

$$\text{MSE} = \frac{1}{N}\sum_{i=1}^{N}(x_i - \hat{x}_i)^2$$

Where: - $x_i$ is the original data point. - $\hat{x}_i$ is the reconstructed data point. - $N$ is the total number of data points.

A significantly higher reconstruction error for a new sequence, compared to the training sequences, indicates a potential anomaly.

### 4.3 Autoencoder

Autoencoders are a type of artificial neural network used primarily for unsupervised learning tasks. Their main objective is to learn a compressed, distributed representation (encoding) of the input data, and then use this representation to reconstruct the original data as closely as possible. The architecture of an autoencoder comprises two main parts: an encoder and a decoder. The encoder compresses the input into a latent-space representation, and the decoder reconstructs the original data from this representation [6].

Mathematically, given an input $x$, the encoder maps it to a latent representation $y$ through a function $f$:

$$y = f(x)$$

The decoder then maps $y$ back to the reconstructed input $\hat{x}$ through a function $g$:

$$\hat{x} = g(y)$$

The primary goal of an autoencoder is to minimize the difference between the original input $x$ and its reconstruction $\hat{x}$. This difference is often measured using a loss function like the mean squared error.

Autoencoders have found applications in various domains, including dimensionality reduction, feature learning, and anomaly detection. Their ability to learn efficient representations from data without the need for explicit labels makes them particularly useful in unsupervised learning scenarios.

### 4.3.1 KMeans Clustering

KMeans is a popular partitioning method that divides a dataset into $K$ distinct, non-overlapping subsets (or clusters). The goal of the algorithm is to partition the data points into clusters such that the sum of the squared distances between the data points and the centroid of their respective clusters is minimized. Mathematically, the objective function $J$ to be minimized is [22]:

$$J = \sum_{i=1}^{n} \sum_{j=1}^{K} w_{ij} \|x_i - c_j\|^2 \tag{4.1}$$

Where:

- $x_i$ is the $i^{th}$ data point.
- $c_j$ is the centroid of the $j^{th}$ cluster.
- $w_{ij}$ is an indicator variable that is 1 if $x_i$ is in cluster $j$ and 0 otherwise.
- $\|\cdot\|$ denotes the Euclidean distance.

The algorithm iteratively refines the position of the centroids. The steps are as follows:

1. Initialize $K$ centroids randomly.
2. Assign each data point to the nearest centroid.
3. Recalculate the centroid of each cluster as the mean of all the data points assigned to that cluster.

4. Repeat steps 2 and 3 until the centroids no longer change significantly or a stopping criterion is met.

One of the challenges with KMeans is the selection of the appropriate number of clusters, $K$. Various methods, such as the elbow method, can be used to determine an optimal value for $K$. Another challenge is that KMeans can be sensitive to the initial placement of centroids. To mitigate this, the algorithm can be run multiple times with different initializations and the best result in terms of within-cluster sum of squares can be chosen.

KMeans clustering has been widely used in various domains, including image segmentation, customer segmentation, and anomaly detection. Its simplicity and efficiency make it a popular choice for clustering tasks, especially when the data distribution is well-defined and the number of clusters is known a priority [22].

# 5 Evaluation Metrics

This section will discuss the evaluation metrics used in our study, including Accuracy, Precision, Recall, F1-Score, and ROC AUC.

## 5.1 Accuracy

Accuracy is a commonly used metric for evaluating classification models but has its limitations in the context of unsupervised anomaly detection, where the goal is to identify rare and previously unseen anomalous patterns in data.

Mathematically, accuracy $A$ can be defined as [2]:

$$A = \frac{TP + TN}{TP + TN + FP + FN} \tag{4.2}$$

Where:

- $TP$ is the number of True Positives
- $TN$ is the number of True Negatives
- $FP$ is the number of False Positives
- $FN$ is the number of False Negatives

In anomaly detection, $TP$ and $TN$ refer to the number of anomalies correctly identified and the number of normal data points correctly identified, respectively. The denominator represents the total number of predictions made or the size of the dataset.

While accuracy $A$ provides a general sense of the model's performance, it may not always be the ideal metric, especially in datasets with imbalanced class distributions. Therefore, it's crucial to evaluate it alongside other metrics such as precision, recall, and the F1-score, to understand the true capabilities of the model.

In our experiments, we use accuracy as a baseline metric but also consider other metrics due to the challenges posed by imbalanced datasets and the nature of unsupervised anomaly detection.

## 5.2 Precision

Precision, also known as the positive predictive value, is a measure of the accuracy of the positive predictions made by a model. In the context of anomaly detection, it quantifies how many of the data points that were predicted as anomalies were actual anomalies. Given the unsupervised nature of anomaly detection, where labels are often not available, a high precision indicates that the model is reliable in its anomaly predictions and minimizes the risk of false alarms.

Mathematically, precision is defined as [15]:

$$\text{Precision}(P) = \frac{TP}{TP + FP} \tag{4.3}$$

Where:

- **True Positives (TP):** The number of actual anomalies correctly identified by the model.
- **False Positives (FP):** The number of normal data points incorrectly identified as anomalies by the model.

In the realm of unsupervised anomaly detection, precision becomes particularly significant. Given that anomalies are rare, a model that raises too many false alarms (low precision) can lead to unnecessary investigations, wasted resources, and reduced trust in the system. Therefore, while it's essential to detect genuine anomalies, it's equally crucial to ensure that the number of false positives is kept to a minimum. This balance is often a challenging aspect of anomaly detection, making precision an invaluable metric for evaluating the performance of detection models [15].

## 5.3 Recall

Recall, often termed as Sensitivity, measures the proportion of actual anomalies that the model correctly identifies. In the context of anomaly detection, it quantifies the model's ability to capture all potential anomalies, ensuring that genuine threats or issues are not overlooked.

Mathematically, recall is defined as [15]:

$$\text{Recall}(R) = \frac{TP}{TP + FN} \tag{4.4}$$

Where:

- **True Positives (TP):** The number of actual anomalies correctly identified by the model.
- **False Negatives (FN):** The number of actual anomalies that the model failed to identify.

In unsupervised anomaly detection, achieving a high recall is of paramount importance. Given the rarity of anomalies and the potential severity of their implications, it's crucial that the model detects as many of them as possible. A model with low recall might overlook genuine anomalies, leading to potential undetected threats or issues. However, it's essential to note that there's often a trade-off between precision and recall. A model that is too aggressive in flagging anomalies might achieve a high recall but at the cost of a reduced precision, leading to many false positives [15].

## 5.4 F1-Score

The F1-Score is a metric that combines both precision and recall into a single value, providing a more holistic view of a model's performance, especially when the distribution of classes is imbalanced, as is often the case in anomaly detection.

Mathematically, the F1-Score is defined as the harmonic mean of precision and recall [17]:

$$\text{F1-Score}(F1) = 2 \times \frac{P \times R}{P + R} \tag{4.5}$$

The F1-Score ranges between 0 and 1, where a score of 1 indicates perfect precision and recall, and a score of 0 indicates that either the precision or the recall is zero.

In the context of unsupervised anomaly detection, where anomalies are rare and both false positives and false negatives can have significant implications, the F1-Score becomes

particularly valuable. It ensures that the model is not biased towards just precision or recall but strikes a balance between the two. A high F1-Score indicates that the model has achieved a good trade-off between identifying genuine anomalies and not raising too many false alarms [**?** ].

### 5.5 ROC AUC

The Receiver Operating Characteristic (ROC) curve is a graphical representation that illustrates the diagnostic ability of a binary classifier as its discrimination threshold varies. The ROC curve plots the True Positive Rate (TPR) against the False Positive Rate (FPR) at various threshold settings.

Mathematically, the TPR and FPR are defined as [4]:

$$\text{TPR} = \frac{TP}{TP + FN}$$
$$\text{FPR} = \frac{FP}{FP + TN}$$

The Area Under the ROC Curve (AUC) provides a scalar value that quantifies the overall ability of the model to discriminate between the positive and negative classes. An AUC of 1.0 indicates perfect classification, while an AUC of 0.5 suggests that the model's performance is no better than random guessing.

In the context of unsupervised anomaly detection, the ROC AUC becomes particularly crucial. Given that anomalies are rare events, the class distribution is typically highly imbalanced. In such scenarios, traditional accuracy can be misleading, and the ROC AUC offers a more informative metric. A high AUC indicates that the model can effectively distinguish between normal and anomalous data points, irrespective of the specific threshold set for classification.

## 6 Experiment 1: Model Performance on the Continuous Time-Series Dataset

The first experiment aims to investigate the impact of using continuous versus non-continuous time-series data on the performance of the machine learning models.The primary objective of the first experiment was to assess the performance of various anomaly detection models on a specific node, ‘r11n14‘, within a defined time range: from ‘2022-10-04 19:20:30‘ to

‘2022-10-17 16:23:00‘. This time range was selected due to its continuous nature, substantial data volume (comprising 29,451 rows), and a moderate anomaly proportion of 0.33.

## 6.1 Parameter Setting

- **iForest:** The Isolation Forest model was initialized with a contamination parameter set to 0.33. This value was determined based on the known proportion of anomalies in the dataset. The intention was to ensure that the model had an appropriate sensitivity level to detect outliers without generating an excessive number of false positives.

- **KMeans:** The KMeans clustering algorithm was employed with the number of clusters set to 2. This binary clustering approach was chosen under the assumption that the data primarily consists of two groups: normal and anomalous. Anomalies were identified by calculating the distance of data points to the cluster centers. Points farther from the centroids than the average distance were classified as anomalies.

- **Autoencoder:** A standard autoencoder was utilized with an encoding dimension of 14. This dimensionality was selected to capture a significant portion of the data variance while enabling the model to identify anomalies based on reconstruction error. The model was trained for 30 epochs to ensure convergence. Anomalies were detected by comparing the reconstruction error, specifically the mean squared error (MSE), between the original and reconstructed data. The threshold for anomaly detection was set at the 67% quantile of the MSE, aiming to identify the top 33% of data points with the highest reconstruction errors as anomalies.

- **LSTM Autoencoder:** The LSTM Autoencoder was designed to handle time-series data, with a sequence length of 10 time steps. The model architecture consisted of two LSTM layers for encoding, followed by two LSTM layers for decoding. The model was trained using the mean squared error (MSE) as the loss function. Anomalies were identified based on the reconstruction error between the original and predicted sequences. The threshold for anomaly detection was set at the 67th percentile of the MSE, targeting the top 33% of sequences with the highest reconstruction errors.

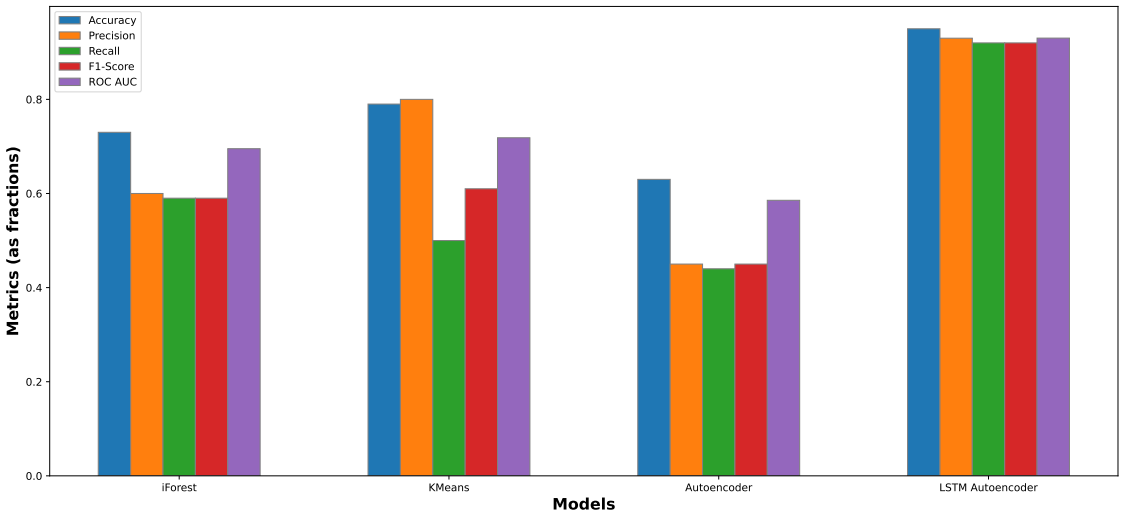


**Figure 4.3:** Experiment 1 results.

## 6.2 Results and Analysis

The results of the anomaly detection models applied to the dataset are summarized in Table 4.1 and Figure 4.3. Each model's performance metrics, including accuracy, precision, recall, F1-score, true positives, and ROC AUC score, are presented.

**Table 4.1:** Experiment 1 results.

| Model | Accuracy | Precision | Recall | F1-Score | True Positives | ROC AUC |
|---|---|---|---|---|---|---|
| iForest | 73% | 60% | 59% | 59% | 7252 | 0.6952 |
| KMeans | 79% | 80% | 50% | 61% | 6110 | 0.7184 |
| Autoencoder | 63% | 45% | 44% | 45% | 5454 | 0.5855 |
| LSTM Autoencoder | 95% | 93% | 92% | 92% | 11264 | 0.9302 |

From the results, it can be observed that the LSTM Autoencoder outperforms the other models in terms of accuracy, precision, recall, and F1-score. The iForest model, while achieving a reasonable accuracy, has a lower precision and recall compared to the LSTM Autoencoder. The KMeans model shows a high precision but a relatively lower recall, indicating that it might be missing out on detecting some anomalies. The Autoencoder has the lowest performance metrics among the models, suggesting that it might not be the best fit for this particular dataset.

The ROC AUC score, which measures the model's ability to distinguish between the nor-

mal and anomalous data points, is highest for the LSTM Autoencoder, further emphasizing its superior performance.

In terms of true positives, the LSTM Autoencoder successfully detects 11,264 anomalies, which is significantly higher than the other models. This indicates that the LSTM Autoencoder is more sensitive to anomalies in the dataset and can detect them with higher accuracy.

Overall, the results suggest that for this specific dataset and the features used, the LSTM Autoencoder is the most suitable model for anomaly detection. Future work could explore optimizing the other models or introducing new features to improve their performance.

# 7 Experiment 2: Model Performance on the Entire PROMETHEUS Dataset

The second experiment was designed to evaluate the performance of anomaly detection models on the entire dataset that encompasses all nodes. This approach contrasts with the first experiment, where a specific node and time range were selected. Due to the discontinuity in the dataset's timestamps, the LSTM Autoencoder was not tested in this experiment. This experiment is designed to provide a comprehensive assessment of the models' ability to detect anomalies in a large, complex dataset.

## 7.1 Parameter Setting

- **iForest:** The Isolation Forest model was initialized with a contamination parameter set to 0.028. This value was chosen based on the expected proportion of anomalies in the dataset, ensuring that the model is sensitive enough to detect outliers while minimizing false positives.

- **KMeans:** The KMeans clustering algorithm was employed with the number of clusters set to 2. This setting was chosen because the data is expected to have two primary groups: normal and anomalous. Anomalies were identified based on the distance of data points to the cluster centers, with the assumption that anomalies would be farther from the cluster centroids than regular data points.

- **Autoencoder:** A standard autoencoder was utilized with an encoding dimension of 14. This dimensionality was selected to capture the majority of the data variance while still enabling the model to detect anomalies based on reconstruction error. The model was trained for 10 epochs to ensure convergence without overfitting. Anomalies

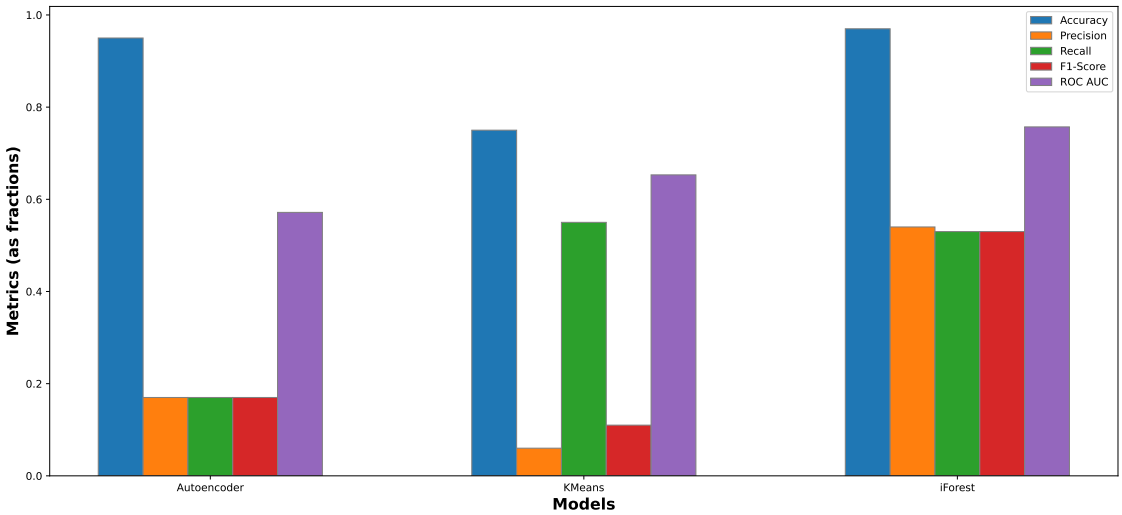


**Figure 4.4:** Experiment 2 results.

were detected based on the reconstruction error, specifically the mean squared error (MSE), between the original and reconstructed data. The threshold for anomaly detection was set at the 97.2% quantile of the MSE, aiming to capture the top 2.8% of data points with the highest reconstruction errors as anomalies.

## 7.2 Results and Analysis

In the second experiment, we evaluated the performance of various anomaly detection models across the entire dataset and all nodes. The results are summarized in Table 4.2 and Figure 4.4.

**Table 4.2:** Experiment 2 results.

| Model | Accuracy | Precision | Recall | F1-Score | True Positives | ROC AUC |
|---|---|---|---|---|---|---|
| Autoencoder | 95% | 17% | 17% | 17% | 594,225 | 0.5716 |
| KMeans | 75% | 6% | 55% | 11% | 1,972,033 | 0.6530 |
| iForest | 97% | 54% | 53% | 53% | 1,876,920 | 0.7571 |

The results from Experiment 2 offer a diverse range of performances across the tested models. The Autoencoder, despite its high accuracy, showed moderate ability in distinguishing between normal and anomalous data points with an ROC AUC score of 0.5716. KMeans, although less precise, demonstrated a higher recall and a slightly better ROC

AUC score. The iForest model emerged as the most effective, with strong precision, recall, and an ROC AUC score of 0.7571.

This leads us to an important consideration regarding the role of training iterations, particularly for deep learning models like the Autoencoder. Due to the computational constraints and the sheer volume of data, the Autoencoder was trained for only 5 iterations. This limited training likely hindered the model's ability to fully converge and capture the underlying patterns in the data. In contrast, in Experiment 1, the Autoencoder's performance peaked around 20 iterations, suggesting that more extensive training could potentially yield better results.

The limited training iterations for the Autoencoder highlight the challenges posed by the dataset's size and computational constraints. It raises questions about the model's ability to generalize well when not adequately trained, which could be a focus for future work.

In summary, Experiment 2 revealed that while some models like iForest performed exceptionally well, others like the Autoencoder could benefit from more extensive training. The varying performances across models underscore the complexity of anomaly detection and the need for careful consideration of model training and computational resources

# 8 Experiment 3: Model Performance on the Integrated Dataset

The third experiment aims to evaluate and compare the performance of the selected machine learning models on the integrated PROMETHEUS and SLURM dataset. This experiment is designed to assess whether integrating job-related information from the SLURM dataset with the time-series data from the PROMETHEUS dataset can enhance the models' ability to detect anomalies.

## 8.1 Parameter Setting

In this experiment, the parameters from the first experiment are retained. However, the dataset is augmented with the following job-related metrics from the SLURM dataset:

- Number of tasks running in the past 30 seconds.
- Number of tasks that successfully completed in the past 30 seconds.
- Success rate of tasks in the past 30 seconds.

The hypothesis is that these additional metrics might provide a richer context for the models, potentially leading to improved anomaly detection performance. By incorporating

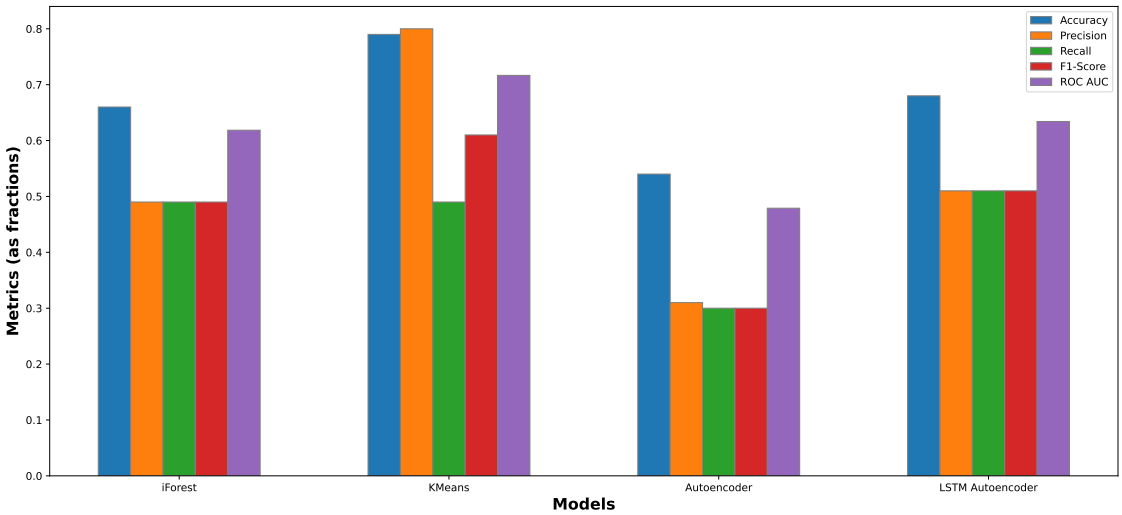


**Figure 4.5:** Experiment 3 results.

these job-related metrics, this experiment seeks to understand if such auxiliary information can offer more insights into the system's behavior and enhance the anomaly detection process.

## 8.2 Results and Analysis

The third experiment aimed to integrate job-related information from the SLURM dataset to enhance anomaly detection capabilities. The performance metrics of various models on this augmented dataset are presented in Table 4.3 and Figure 4.5. These metrics are also compared with the models' performance in Experiment 1 to assess the impact of the additional metrics.

**Table 4.3:** Experiment 3 results.

| Model | Accuracy | Precision | Recall | F1-Score | True Positives | ROC AUC |
|---|---|---|---|---|---|---|
| iForest | 66% | 49% | 49% | 49% | 4,795 | 0.6185 |
| KMeans | 79% | 80% | 49% | 61% | 4,854 | 0.7166 |
| Autoencoder | 54% | 31% | 30% | 30% | 2,966 | 0.4789 |
| LSTM Autoencoder | 68% | 51% | 51% | 51% | 4,999 | 0.6341 |

The results of Experiment 3 reveal a nuanced picture. While some models like KMeans and LSTM Autoencoder maintained their performance levels, others like iForest and the standard Autoencoder experienced declines in various metrics. This suggests that the

additional job-related metrics did not universally improve the models' anomaly detection capabilities.

Delving deeper into the complexities of data integration, especially in the context of anomaly detection, we find that the SLURM dataset introduces a unique set of challenges. It provides a macroscopic view of task statuses across nodes, affecting the time series of all nodes involved in a task when a failure occurs. However, anomalies at the node level are often transient and localized, which may not necessarily impact the entire task or all nodes involved. This discrepancy introduces noise into the model, potentially affecting its performance.

The integration of SLURM data, therefore, appears to be a double-edged sword. On one hand, it provides additional context that could be valuable for anomaly detection. On the other hand, the granularity and nature of the data might introduce noise, thereby affecting the model's performance adversely.

In summary, while the integration of job-related metrics from the SLURM dataset adds another layer of complexity, it does not consistently enhance the performance of the models tested. Future work could explore alternative methods of integrating these metrics or consider other auxiliary information sources to improve model performance.

# 9 Experiment 4: Model Generalization Across Different Datasets

The fourth experiment is designed to evaluate the generalization capability of the models trained on the data from Experiment 1. The primary objective is to assess how well the models, which were trained on the 'r11n14' node data from the PROMETHEUS dataset, perform on a different node ('r14n17') from the Additional PROMETHEUS Dataset. This experiment will provide insights into the models' ability to detect anomalies in datasets they haven't been trained on, which is crucial for real-world applications where the model might be deployed on unseen data.

## 9.1 Experiment Design

The data for this experiment is sourced from the 'r14n17' node of the Additional PROMETHEUS Dataset, spanning from 'Mar 02 2020 13:10:30' to 'Mar 17 2020 20:51:15'. This dataset was chosen due to its continuous nature, substantial data volume, and an anomaly proportion similar to that of Experiment 1.

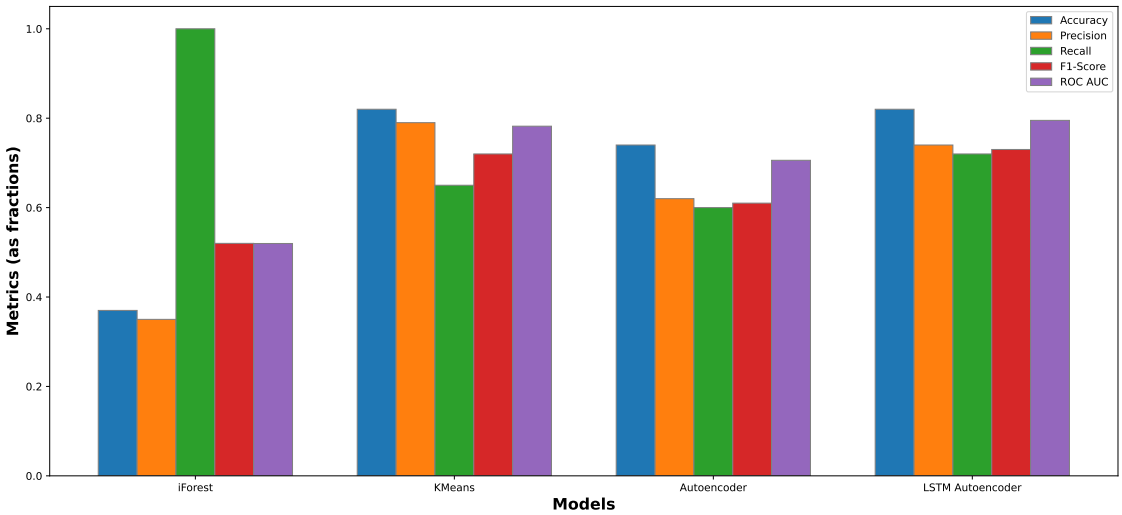


**Figure 4.6:** Experiment 4 results.

Instead of training the models from scratch, this experiment leverages the models that were previously trained on the 'r11n14' node data from Experiment 1. This approach allows for a direct assessment of the models' generalization capabilities.

This experiment is pivotal in understanding the robustness of the models and their adaptability to different datasets. It will shed light on the potential need for model retraining or fine-tuning when transitioning between datasets with similar characteristics.

## 9.2 Results and Analysis

The results of Experiment 4, where models trained on the 'r11n14' node data from Experiment 1 were tested on the 'r14n17' node data from the Additional PROMETHEUS Dataset, are presented in Table 4.4 and Figure 4.6

**Table 4.4:** Experiment 4 results.

| Model | Accuracy | Precision | Recall | F1-Score | True Positives | ROC AUC |
|---|---|---|---|---|---|---|
| Autoencoder | 74% | 62% | 60% | 61% | 18,114 | 0.7058 |
| iForest | 37% | 35% | 100% | 52% | 30,136 | 0.5197 |
| KMeans | 82% | 79% | 65% | 72% | 19,714 | 0.7821 |
| LSTM Autoencoder | 82% | 74% | 72% | 73% | 21,653 | 0.7951 |

Upon comparing the results from Experiment 4 with those of Experiment 1, several observations can be made:

- The **Autoencoder** in Experiment 4 achieved an accuracy of 0.74, which is an improvement from the 0.63 in Experiment 1. The precision, recall, and F1-Score also saw improvements, suggesting that the model trained on the ‘r11n14‘ node data has a good generalization capability when tested on the ‘r14n17‘ node data.

- The **iForest** model's performance dropped significantly in Experiment 4, with an accuracy of only 0.37, compared to 0.73 in Experiment 1. This indicates that the model might be overfitting to the specific characteristics of the ‘r11n14‘ node data and struggles to generalize to the ‘r14n17‘ node data.

- The **KMeans** model's accuracy in Experiment 4 is 0.82, a slight improvement from the 0.79 in Experiment 1. The precision and recall also saw improvements, suggesting that KMeans has a stable clustering mechanism that works similarly across different datasets with comparable characteristics.

- The **LSTM Autoencoder** achieved an accuracy of 0.82 in Experiment 4, which is a decrease from the 0.95 in Experiment 1. However, the precision and recall in Experiment 4 are slightly lower than in Experiment 1. This indicates that while the model still generalizes well, there might be specific characteristics in the ‘r14n17‘ node data that the model finds challenging.

In conclusion, while some models like the Autoencoder and KMeans showed promising generalization capabilities, others like iForest might require fine-tuning or retraining when transitioning to different datasets.

## 10 Summary

Throughout this chapter, we systematically explored the performance of various anomaly detection models across different scenarios and data characteristics within the PROMETHEUS dataset.

To discern the most robust model from each experiment, the Receiver Operating Characteristic Area Under the Curve (ROC AUC) metric was principally considered. ROC AUC provides an effective measure that encapsulates both the true positive rate and the false positive rate, thereby offering a holistic understanding of model performance regardless of the threshold setting. From each experiment, the model with the highest ROC AUC was deemed the best performing. Table 4.5 and Figure 4.7 shows the performance metrics of the best models chosen from each experiment.

**Table 4.5:** Best model comparison across experiments.

| Exp. | Model | Accuracy | Precision | Recall | F1-Score | True Positives | ROC AUC |
|---|---|---|---|---|---|---|---|
| 1 | LSTM Autoencoder | 95% | 93% | 92% | 92% | 11,264 | 0.9302 |
| 2 | iForest | 97% | 54% | 53% | 53% | 1,876,920 | 0.7571 |
| 3 | KMeans | 79% | 80% | 49% | 61% | 4,854 | 0.7166 |
| 4 | LSTM Autoencoder | 82% | 74% | 72% | 73% | 21,653 | 0.7951 |

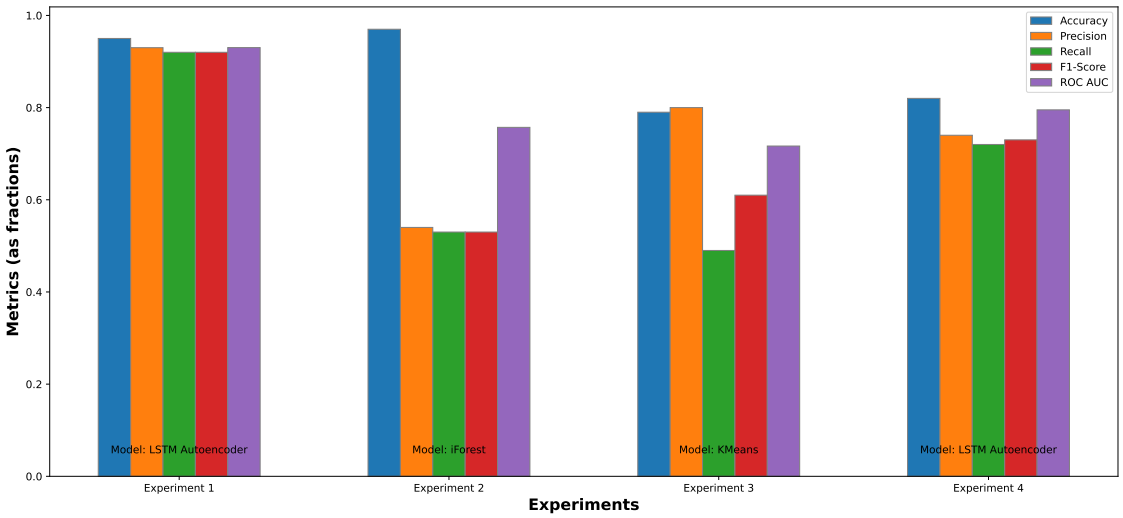


**Figure 4.7:** Experiment results comparison.

In the first experiment, centered around a specific node with continuous time-series data, comprising 29,451 rows and an anomaly proportion of 33%, the LSTM Autoencoder emerged as the most proficient model, demonstrating its capability in handling time-series data with a high proportion of anomalies.

The second experiment expanded the dataset to encompass all nodes, resulting in a massive data volume of 129,546,553 rows. However, the anomaly proportion significantly decreased to 2.8%. Despite the absence of LSTM in this experiment, the iForest model showcased commendable performance, especially considering the vastness of the dataset and the stark contrast in anomaly proportion compared to the first experiment.

The third experiment aimed to understand the impact of integrating job-related information from the SLURM dataset with the PROMETHEUS time-series data. The results were mixed; while some models maintained their performance, others saw declines, suggesting that the integration of job-related metrics might introduce complexities that not

all models can handle effectively.

In the fourth experiment, we applied models trained on the data from the first experiment to a different dataset: the Additional PROMETHEUS Dataset's r14n17 node data from Mar 02 2020 to Mar 17 2020. Among the models tested, the LSTM Autoencoder once again stood out, achieving commendable performance metrics. This suggests that the LSTM Autoencoder not only excels in handling time-series data with high anomaly proportions but also showcases a significant degree of generalizability across datasets. However, when comparing the performance of other models like iForest and KMeans, it's evident that while some models maintain their efficacy across datasets, others might face challenges, emphasizing the importance of model adaptability.

Across all experiments, the scale of the dataset and the proportion of anomalies play pivotal roles in influencing model performance. The LSTM Autoencoder consistently showcased its strength, especially when dealing with time-series data with high anomaly proportions. On the other hand, the iForest model demonstrated resilience in the second experiment with a vast dataset but faced challenges in the fourth experiment.

The integration of job-related metrics in the third experiment presented mixed outcomes. It underscores the notion that while auxiliary data can provide additional context, it also introduces complexities that can influence model behavior. The fourth experiment further emphasized the importance of model generalizability. In essence, while the LSTM Autoencoder emerged as the most consistent performer across diverse datasets, the experiments highlight the multifaceted nature of anomaly detection. Model performance is deeply intertwined with data characteristics, scale, and the nature of anomalies present.

# Chapter 5

# Conclusion and Future Work

Throughout this research, we have delved into the complex realm of anomaly detection within high-performance computing clusters. This journey, although challenging, has been enlightening, offering invaluable insights into the intricacies of handling large datasets and detecting anomalies within them.

## 1 Conclusion

### 1.1 RQ1: What are the characteristics of HPC Node data and Job Data?

Our in-depth data characterization has provided valuable insights into the structure, relationships, and inherent patterns within the HPC Node and Job Data.

### 1.2 RQ2: How can node data be integrated with job data for the purpose of anomaly detection?

The integration of HPC node data from the PROMETHEUS Dataset with job data from the SLURM Dataset has been a pivotal aspect of our research. We developed a method to seamlessly integrate these two types of operational data, thereby providing a more comprehensive view of the system's state.

### 1.3 RQ3: How do various unsupervised machine learning models perform in detecting anomalies in large-scale HPC clusters?

Our methodologies, from the use of autoencoders to the application of clustering algorithms, have demonstrated the versatility and depth of our approach. The models were rigorously evaluated using established metrics, attesting to their effectiveness in anomaly detection within large-scale HPC clusters.

# 2 Future Work

## 2.1 Future Work for RQ1

For RQ1, future work could delve into more granular aspects of HPC Node and Job Data by examining the impact of different types of jobs on node performance. Additionally, advanced statistical methods could be employed to better understand the underlying distributions and correlations within the data.

## 2.2 Future Work for RQ2

For RQ2, which focused on the integration of node and job data, future studies could explore different methods of data integration, such as using machine learning techniques to automatically annotate anomalies based on a wider range of metrics. Additionally, optimizing the integration process to handle real-time data streams could allow for more timely anomaly detection.

One potential avenue for improvement lies in the granularity of task logs. Specifically, recording the exact timestamp when a task starts to fail could help pinpoint the onset of an anomaly, and identifying the specific node or nodes responsible for the task failure could offer insights into localized anomalies. By maintaining such detailed logs, we could train our model to recognize correlations between node-specific anomalies and task failures more effectively. This aligns with our initial vision of leveraging both node and task information to enhance anomaly detection capabilities.

## 2.3 Future Work for RQ3

For RQ3, which examined the performance of various unsupervised machine learning models in anomaly detection, future work could involve re-evaluating the performance of the autoencoder on the dataset from Experiment 2 with an increased number of iterations. Conducting this on a more powerful computing cluster could mitigate challenges posed by the dataset's size and potentially improve the model's accuracy and reliability. Additionally, exploring other unsupervised learning models, such as Generative Adversarial Networks (GANs), could offer new avenues for effective anomaly detection.

In conclusion, this research has laid a robust foundation for anomaly detection within HPC clusters. The methodologies developed, insights gained, and challenges overcome all attest to the potential of our approach. As we look forward to the future, we are optimistic about the advancements and innovations that await in this ever-evolving field.

# Bibliography


[1] Carletti, M., Terzi, M., and Susto, G. A. Interpretable anomaly detection with diffi: Depth-based feature importance of isolation forest. *Engineering Applications of Artificial Intelligence 119* (2023), 105730. 15

[2] Chicco, D., and Jurman, G. The advantages of the matthews correlation coefficient (mcc) over f1 score and accuracy in binary classification evaluation. *BMC Genomics* (2020). 51

[3] Chu, X., Talluri, S., Versluis, L., and Iosup, A. How do ml jobs fail in datacenters? analysis of a long-term dataset from an hpc cluster. In *Companion of the 2023 ACM/SPEC International Conference on Performance Engineering* (New York, NY, USA, 2023), ICPE '23 Companion, Association for Computing Machinery, p. 263–268. 13, 14

[4] Fawcett, T. An introduction to roc analysis. *Pattern recognition letters 27*, 8 (2006), 861–874. 54

[5] Ghiasvand., S. upad: Unsupervised privacy-aware anomaly detection in high performance computing systems. In *Proceedings of the 8th International Conference on Pattern Recognition Applications and Methods - ICPRAM* (2019), INSTICC, SciTePress, pp. 852–859. 15

[6] Hinton, G. E., and Salakhutdinov, R. R. Reducing the dimensionality of data with neural networks. *science 313*, 5786 (2006), 504–507. 49

[7] Hochreiter, S., and Schmidhuber, J. Long short-term memory. *Neural computation 9*, 8 (1997), 1735–1780. 48

[8] Iosup, A., Kuipers, F., Varbanescu, A. L., Grosso, P., Trivedi, A., Rellermeyer, J., Wang, L., Uta, A., and Regazzoni, F. Future computer systems and networking research in the netherlands: A manifesto, 2022. 8

[9] Iosup, A., Uta, A., Versluis, L., Andreadis, G., van Eyk, E., Hegeman, T., Talluri, S., van Beek, V., and Toader, L. Massivizing computer systems: A vision to understand, design, and engineer computer ecosystems through and beyond modern distributed systems. In *2018 IEEE 38th International Conference on Distributed Computing Systems (ICDCS)* (2018), pp. 1224–1237. 8

[10] Kusuma, G. Y., and Oktiawati, U. Y. Application performance monitoring system design using opentelemetry and grafana stack. *Jurnal Ilmu Sistem dan Teknologi Informasi 3*, 1 (2022), 1–10. 16, 30

[11] Liu, F. T., Ting, K. M., and Zhou, Z.-H. Isolation forest. *2008 Eighth IEEE International Conference on Data Mining* (2008), 413–422. 47, 48

[12] Liu, P., and Guitart, J. Performance characterization of containerization for hpc workloads on infiniband clusters: an empirical study. *Cluster Computing 25*, 2 (4 2022), 847–868. 8

[13] Molan, M., Borghesi, A., Cesarini, D., Benini, L., and Bartolini, A. Ruad: Unsupervised anomaly detection in hpc systems. *Future Generation Computer Systems 141* (2023), 542–554. 15

[14] Morrow, A., Baseman, E., and Blanchard, S. Ranking anomalous high performance computing sensor data using unsupervised clustering. In *2016 International Conference on Computational Science and Computational Intelligence (CSCI)* (2016), pp. 629–632. 15

[15] Powers, D. M. Evaluation: from precision, recall and f-measure to roc, informedness, markedness and correlation. *arXiv preprint arXiv:2010.16061* (2011). 52, 53

[16] Reuther, A., Byun, C., Arcand, W., Bestor, D., Bergeron, B., Gadepally, V., Houle, M., Hubbell, M., Jones, M., Klein, A., et al. Interactive supercomputing on 40,000 cores for machine learning and data analysis. *arXiv preprint arXiv:1807.07814* (2018). 16

[17] Sokolova, M., and Lapalme, G. A systematic analysis of performance measures for classification tasks. *Information Processing  Management 45*, 4 (2009), 427–437. 53

[18] Suman, S. Literature study on operational data analytics frameworks in large-scale computing infrastructures. 9

[19] Tanash, M., Yang, H., Andresen, D., and Hsu, W. Ensemble prediction of job resources to improve system performance for slurm-based hpc systems. In *Proceedings of the 2021 ACM SIGMETRICS International Conference on Measurement and Modeling of Computer Systems* (2021), pp. 1–2. 30

[20] Wang, T., Wei, J., Zhang, W., Zhong, H., and Huang, T. Workload-aware anomaly detection for web applications. *Journal of Systems and Software 89* (2014), 19–32. 15

[21] Wu, C., Yan, B., Yu, R., Yu, B., Zhou, X., Yu, Y., and Chen, N. k-means clustering algorithm and its simulation based on distributed computing platform. *Hindawi* (2021). 24

[22] Zahra, S., Ghazanfar, M., Khalid, A., Azam, M. A., Naeem, U., and Prügel-Bennett, A. Novel centroid selection approaches for kmeans-clustering based recommender systems. *Information Sciences 324* (2015), 20–36. 50, 51