# Correlating Job Logs with Hardware Failures in High-Performance Computing

**Manoj Dattatreya Myneni**

Graduation: Spring 2025

UIN: 659696543

mmyne@uic.edu

Advisor: **Zhiling Lan**

College of Engineering
Department of Computer Science
University of Illinois Chicago
zlan@uic.edu

Committee Member: **Anrin Chakraborti**

College of Engineering
Department of Computer Science
University of Illinois Chicago
anrin@uic.edu

Department of Computer Science
University of Illinois Chicago
Chicago, IL, USA

# Abstract

High-performance computing (HPC) systems, such as the *Theta* supercomputer at Argonne National Laboratory, provide vital computational resources for cutting-edge scientific research across a diverse range of domains. As these systems scale in complexity, ensuring their reliability and minimising downtime becomes increasingly challenging. One approach to enhancing HPC reliability involves examining the rich sources of data produced during system operation—specifically, correlating job logs with hardware-error events. By systematically integrating and analysing datasets that record both job execution details and hardware malfunctions, it is possible to uncover patterns linking low-level component faults to high-level job failures.

This study focuses on the Theta system, which features a hierarchical architecture of cabinets, chassis, blades, and nodes interconnected via a DragonFly topology. Over 91,000 job records and 136,000 hardware-error events from 2019 form the basis of the analysis. After careful data preparation—including timestamp standardisation, missing-value handling, and component-ID parsing—direct, time-based, and spatial correlation methods are applied. The results show that while direct correlations—where hardware logs explicitly record the job ID—are rare, they offer strong evidence of causality. Time-based correlations reveal hardware errors occurring during a job's execution window even without explicit job IDs. Further refining the analysis, spatial correlation maps component-level failures to specific nodes or node ranges used by the job, enhancing confidence in inferred relationships.

These correlations highlight hotspots of failing hardware components and inform the design of predictive-maintenance strategies. By identifying patterns that reliably precede failures, operators can schedule proactive repairs, minimise resource waste, and improve job throughput. The insights gained here provide a foundation for advanced predictive models, guiding maintenance scheduling, workload allocation, and the development of more resilient HPC infrastructures.

**Index Terms** – High-Performance Computing (HPC), job logs, hardware errors, Theta supercomputer, predictive maintenance, correlation analysis, reliability, system architecture, DragonFly topology, data integration, timestamp standardization, HPC operations, node mapping, failure criteria, proactive maintenance.

# Contents

# Chapter 1

# Background and Motivation

High-Performance Computing (HPC) systems stand at the forefront of modern scientific inquiry, providing the computational backbone for advanced research domains ranging from climate modelling and astrophysics to genomics and materials science. With simulations and analyses scaling to unprecedented complexity and size, HPC platforms have evolved into sprawling infrastructures featuring thousands of compute nodes, specialised processors, intricate network topologies, and high-speed storage systems. In this context, the reliability, resilience, and maintainability of HPC systems directly impact the quality and timeliness of scientific discoveries. Unexpected hardware errors, job failures, or system downtime can lead to delayed research outcomes, wasted computational resources, and inflated operational costs.

One representative example of an advanced HPC platform is the *Theta* supercomputer at Argonne National Laboratory. Theta is equipped with Intel Xeon Phi processors distributed across 4,608 nodes and delivers a peak performance of approximately 11.69 petaflops. Such a system is designed to handle intensive computational workloads for a diverse range of scientific applications. However, as systems like Theta grow more complex, the risk and frequency of component-level malfunctions—memory errors, CPU faults, interconnect issues—also increase. Even minor hardware glitches, if left undetected, can cascade into substantial disruptions, causing numerous job failures and affecting the productivity of countless researchers who rely on timely results.

The motivation for this research arises from the critical need to maintain system reliability and minimise downtime. A single job failure not only wastes the compute hours allocated to that job but can also invalidate partial results, disrupt workflow pipelines, and necessitate costly re-runs. From an operational standpoint, the ability to proactively detect, diagnose, and mitigate hardware-related issues before they lead to widespread job failures is invaluable. Historically, HPC system administrators have relied on reactive strategies: hardware-monitoring tools, environmental sensors, and Resource and Service Management (RSM) systems provide logs *after* failures occur. Operators then schedule maintenance and repairs. This approach, while functional, is not optimised for prevention.

An alternative perspective focuses on the wealth of data generated by HPC systems. Job logs, in particular, represent a rich source of information about how applications interact with the underlying hardware. These logs include details about start and end times, resource-usage patterns, performance metrics, error messages, and exit statuses. By correlating job logs with hardware-error logs—detailed records of hardware malfunctions, their timestamps, and affected components—we can potentially gain insights into subtle patterns that precede outright failures.

Such correlation analysis can serve multiple purposes:

- Illuminate causal relationships, identifying whether certain job-failure classes consistently coincide with particular hardware faults.
- Guide predictive maintenance: if specific error patterns in job logs often manifest hours before a hardware component fails, operators could proactively intervene.
- Inform workload-scheduling policies, where the scheduler avoids placing high-priority tasks on nodes exhibiting warning signs in recent logs.

In this project, the main goal is to analyse the correlation between job logs and hardware errors on Theta. By examining job-execution intervals, hardware-failure timestamps, and spatial mappings of nodes and components, we aim to identify how hardware issues influence job outcomes. Ultimately, establishing these correlations can help predict job failures and improve the resilience and throughput of large-scale HPC systems.

System reliability is the cornerstone of HPC productivity. When hardware errors remain undetected, they can degrade performance, reduce effective resource availability, and erode user trust. Properly correlating job failures and hardware issues addresses these challenges head-on, illuminating how, when, and where issues occur and paving the way for strategies that prevent failures rather than simply reacting to them.

# Chapter 2

# Primary Goal and Specific Objectives

The overarching goal of this study is to analyze the correlation between job logs and hardware errors of the Theta HPC system at Argonne National Laboratory, ultimately aiding in the prediction and mitigation of job failures. This high-level objective encompasses multiple subgoals and specific tasks designed to provide a comprehensive understanding of how low-level hardware faults influence high-level computational workflows.

## A. Primary Goal

Correlation and Prediction: Examine and quantify the relationships between HPC job logs and hardware error events, with a focus on understanding how and when job-level anomalies align with underlying hardware issues. The long-term vision is to leverage these correlations to predict imminent job failures and proactively take remediation steps, ranging from alerting system administrators to re-routing workloads to healthier nodes.

## B. Specific Objectives

1. Correlate Job Logs with Hardware Failure Logs: The first step is to integrate two disparate but complementary data sources: job logs and hardware error logs. Job logs provide insights into what jobs were running, where they ran, how long they took, and whether they succeeded or failed. Hardware error logs detail the nature, location, and timing of hardware related faults. By merging these datasets on common keys such as COBALT_JOBID or through time-based correlation windows, we seek to uncover hidden patterns that might not be apparent when analyzing these logs independently.
2. Define and Identify Job and Hardware Failures: To perform meaningful correlation, clear definitions of what constitutes a job failure or a hardware failure are essential. For jobs, a non-zero exit status is often the simplest indicator of failure, but other signals like abnormal termination, exceeded walltime, or specific error messages may also qualify. On the hardware side, failures can span a range of severities, from transient memory errors and single-event upsets to persistent CPU core faults and repeated interconnect glitches.

Establishing robust criteria ensures the analysis focuses on genuine failures that impact HPC operations.

3. Implement Time-Based and Spatial Correlation Methods: Direct correlations rely on explicit linkage of COBALT_JOBID fields. However, not all hardware errors are directly tagged with a job ID. In these cases, time-based correlation looks for hardware error events that occur during a job's execution window. Similarly, spatial correlation attempts to map hardware error events to specific nodes or sets of nodes, then cross-references those nodes to jobs that were actively running on them at the time of the error. These methods help identify subtle relationships, such as repeated node-level hardware issues that correlate with a cluster of job failures over a given period.
4. Analyze Affected Jobs on the Highest-Failure-Rate Node(s): HPC systems like Theta are composed of thousands of nodes, each subdivided into cabinets, chassis, blades, and nodes. Some nodes or node groups may exhibit higher failure rates due to manufacturing variances, aging components, or usage patterns. By focusing on the compute node with the highest failure rate, we can gain deeper insights into localized phenomena and potentially identify the underlying causes of repeated failures. This targeted approach may also serve as a template for a broader system-wide analysis.

## C. Rationale for the Chosen Objectives

The rationale behind these objectives is grounded in operational efficiency. HPC environments process an enormous volume of computations, and each job log represents a piece of evidence: how the system behaved under a particular workload. Hardware error logs, on the other hand, record the system's underlying health. By correlating these two sets of evidence, we aim to build a more complete narrative.

If a particular pattern of warning messages in job logs consistently precedes a memory DIMM failure by several hours, that pattern could serve as an early warning sign. Similarly, if certain node configurations or applications consistently trigger hardware errors, then system administrators can either reassign critical workloads away from those nodes or schedule maintenance interventions before catastrophic failures occur.

Moreover, these objectives support the shift towards predictive maintenance strategies. Traditionally, HPC centers adopt reactive methods: a node fails, administrators investigate logs, and then repair or replace components. By demonstrating that hardware failures can be predicted from job logs, we open the door to proactive measures—fixing or removing problematic components before they cause job failures, reducing downtime, and saving resources.

## D. Challenges and Considerations

The complexity of HPC systems and the variety of workloads running on them present challenges to correlation efforts. Jobs differ drastically in terms of computational patterns, resource requirements, and run times. Hardware failures can range from sporadic events to lingering, recurring problems. This diversity demands flexible correlation methods—direct, time-based, and spatial—and a careful consideration of outliers and noise in the data.

Another challenge is ensuring that correlations are not purely coincidental. Observing that a certain error message tends to appear before a hardware failure does not necessarily prove causality. Additional statistical analyses, machine learning approaches, or domain knowledge may be needed to confirm that the observed correlation truly indicates an underlying relationship.

Lastly, this endeavor must respect data integrity and confidentiality. Some job logs may contain sensitive code fragments or proprietary methods. The correlation analysis should be mindful of privacy policies and adhere to all regulations governing the handling of HPC usage data.

## E. Conclusion of Objectives

By clearly defining failures, merging datasets, and implementing correlation methods, we can move beyond isolated observations. The ultimate outcome is a set of actionable insights that guide HPC maintenance and operations towards greater efficiency and reliability. Each objective serves as a stepping stone in achieving the overarching goal: a predictive framework that harnesses the synergy of job logs and hardware error data to improve HPC system health.

# Chapter 3

# Theta System Architecture

The Theta supercomputer at Argonne National Laboratory exemplifies the state-of-the-art in high-performance computing design. Understanding its architecture is essential for interpreting the spatial and component-level references embedded in both the job logs and hardware error logs. Such an understanding provides the foundation for spatial correlation methods and contextualizes the complexity and scale of the system under study.

## 3.1 Physical Layout and Components

Theta's architecture can be conceptualized in a hierarchical manner, beginning with a top-level arrangement of cabinets, each containing multiple chassis, which in turn house numerous blades, and ultimately culminating in the nodes—each equipped with processing elements, memory, and interconnect interfaces. Specifically, Theta comprises 24 cabinets, each containing 3 chassis. Every chassis hosts 16 blades, and each blade holds 16 cards. On these cards reside the Intel Xeon Phi processors that provide the computational backbone of Theta, ensuring the system can reach its peak performance of 11.69 petaflops.

This nested hierarchy is essential for large-scale operations. The design allows for localized maintenance and containment of failures, as well as efficient provisioning of resources. By isolating errors to particular cabinets, chassis, blades, or nodes, system administrators can quickly zero in on problematic hardware while leaving the rest of the system fully operational.

## 3.2 DragonFly Network Topology

Beyond its physical hierarchy, Theta employs a sophisticated network topology known as DragonFly. DragonFly is designed to provide both high bandwidth and low latency, scaling gracefully as the system grows in size. It accomplishes this by combining high-radix switches with a hierarchical grouping of nodes into local groups, which are then interconnected by global links.

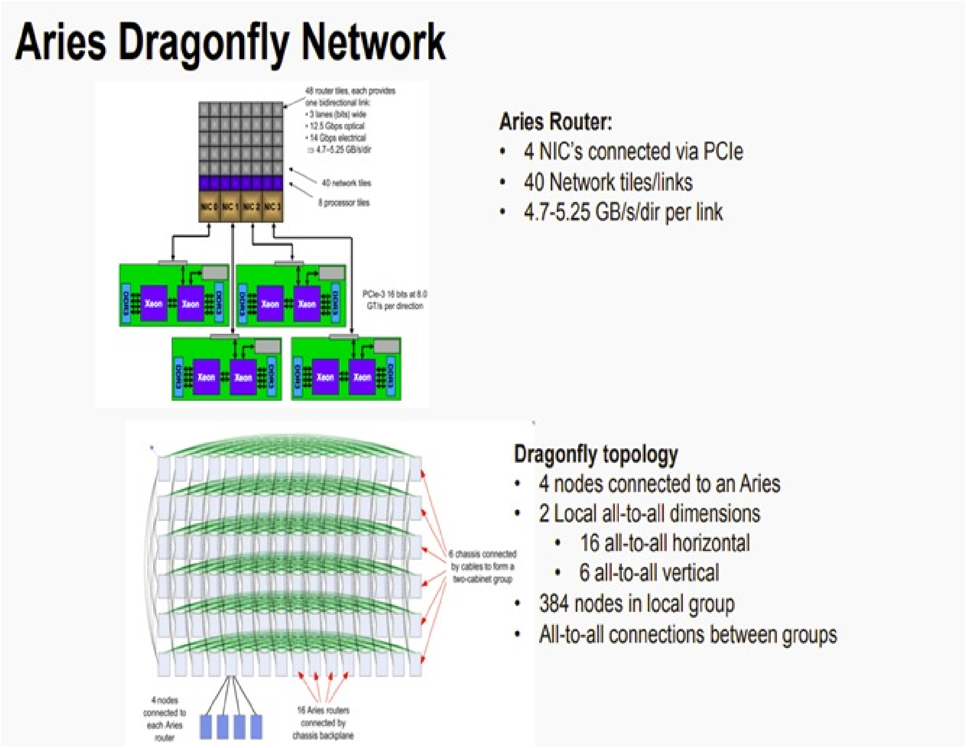
### 3.2.1 Local Interconnects

Within groups (often corresponding to local sets of cabinets or chassis), an all-to-all pattern ensures that nodes can communicate efficiently with minimal hops. This local neighborhood concept reduces contention and provides a robust baseline of intra-group bandwidth.

### 3.2.2 Global Interconnects

DragonFly's global links connect these local groups, enabling system-wide communication. Although global links are more expensive in terms of latency and complexity, careful topology design minimizes the frequency with which messages must travel outside of local neighborhoods.

DragonFly's ability to handle various traffic patterns is essential in an HPC environment running heterogeneous and dynamic workloads. Some applications are tightly coupled and require frequent synchronization among nodes, while others primarily perform local computations and only occasionally communicate globally. The flexibility and robustness of the DragonFly topology help maintain performance even when certain nodes or links are under stress due to hardware failures.


48 router tiles, each provides one bidirectional link:
• 3 lanes (bits) wide
• 12.5 Gbps optical
• 14 Gbps electrical
⇒ 4.7-5.25 GB/s/dir
40 network tiles
8 processor tiles
NIC 0
NIC 1
NIC 2
NIC 3
Xeon
DDR3
PCIe-3 16 bits at 8.0 GT/s per direction
6 chassis connected by cables to form a two-cabinet group
4 nodes connected to each Aries router
16 Aries routers connected by chassis backplane

Aries Dragonfly Network

**Aries Router:**
- 4 NIC's connected via PCIe
- 40 Network tiles/links
- 4.7-5.25 GB/s/dir per link

**Dragonfly topology**
- 4 nodes connected to an Aries
- 2 Local all-to-all dimensions
  - 16 all-to-all horizontal
  - 6 all-to-all vertical
- 384 nodes in local group
- All-to-all connections between groups

Figure 3.1: Aries Dragonfly Network

## 3.3 Mapping Components to Nodes

The detailed naming conventions and identifiers used in the hardware error logs rely on a consistent mapping from cabinets, chassis, blades, and node offsets to a global node numbering scheme. For instance, a component identifier like `c1-1c2s10n2` might correspond to a particular node within the entire system. A deterministic function can translate these component identifiers into a numerical node index that can be directly matched against job logs referencing nodes or node ranges.

This node mapping ensures that spatial correlation is possible. Consider that job logs often denote which nodes a job utilized, sometimes in a range format (e.g., nodes 110–119). On the other hand, hardware errors might be recorded using a composite identifier referencing a particular cabinet, chassis, blade, and node offset. By parsing and converting these identifiers into a consistent node numbering system, we can cross-reference the job logs and hardware logs, enabling analysis of whether a hardware fault on Node X coincided with a running job on that same node.

## 3.4 Locality and Failure Propagation

One reason why spatial correlation matters is that the location of a hardware error can influence how severely it impacts running jobs. A failure at a single node may only disrupt the jobs currently running there, while a failure in a more critical component—such as a chassis-level controller or a switch in the network—could potentially affect multiple nodes, triggering a cascade of job failures across a larger part of the system. Understanding Theta's hierarchical architecture allows us to differentiate between localized failures and broader systemic issues.

Localizing failures also aids in maintenance. If analysis reveals that a specific chassis or blade is producing a disproportionate number of hardware error events over time, administrators can target that hardware for inspection, replacement, or firmware updates. By focusing efforts on the most failure-prone segments of the hierarchy, operational efficiency improves and system downtime decreases.

## 3.5 DragonFly Topology and Error Distribution

The network topology itself can influence the distribution and impact of hardware errors. For example, if a node fails within a particular local group, the DragonFly topology might naturally reroute communication. While jobs running on that node may fail immediately, other jobs in the same group might experience a brief performance degradation or message timeouts before stabilizing. Over time, analyzing hardware error events across different parts of the network can reveal if certain groups are more error-prone or if certain network links are frequently degraded.

These insights can inform future HPC system designs and improvements. Understanding how topology interacts with hardware reliability and job outcomes can lead to architectural refinements that minimize the potential for localized failures to snowball into more extensive disruptions. It might also suggest software strategies, such as workload schedulers that preferentially allocate critical or long-running jobs to nodes known to have fewer hardware incidents, or routing algorithms that avoid unreliable links.

## 3.6 Challenges in Spatial Mapping and Data Consistency

While the architecture provides a structured framework, it also adds complexity to data analysis. Properly parsing component IDs, confirming that node numbering schemes remain consistent across all logs, and ensuring that changes in the system's configuration (upgrades, replacements, or re-cabling) are recorded and accounted for can be challenging. Additionally, different logs might adopt different naming conventions or abbreviations. The job logs might record nodes using a simplified numbering scheme, while hardware logs might rely on the more descriptive cX-1cYsZnM format. Standardizing this information is a critical step, as inaccurate or incomplete mappings would reduce the reliability of spatial correlation results.

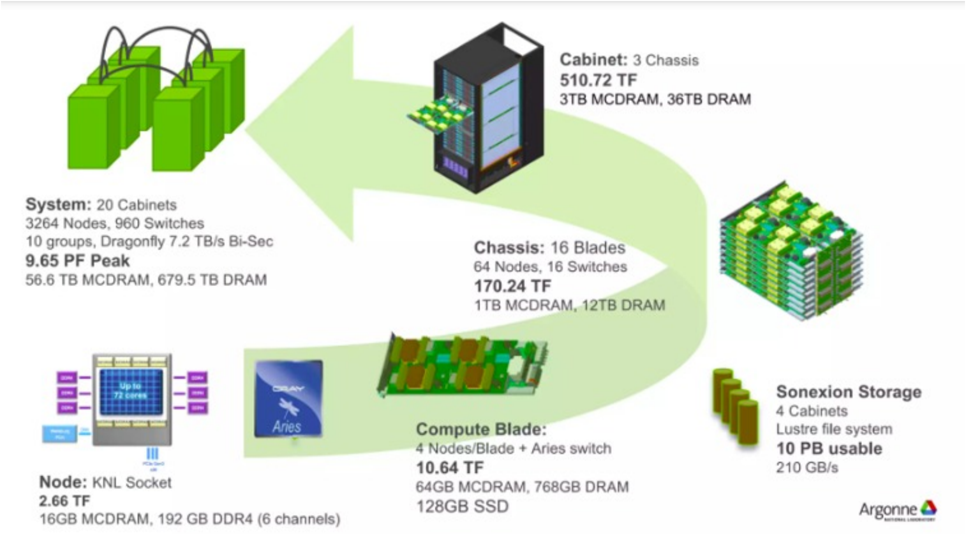


Figure 3.2: Theta System Architecture

## 3.7 Conclusion on Architecture

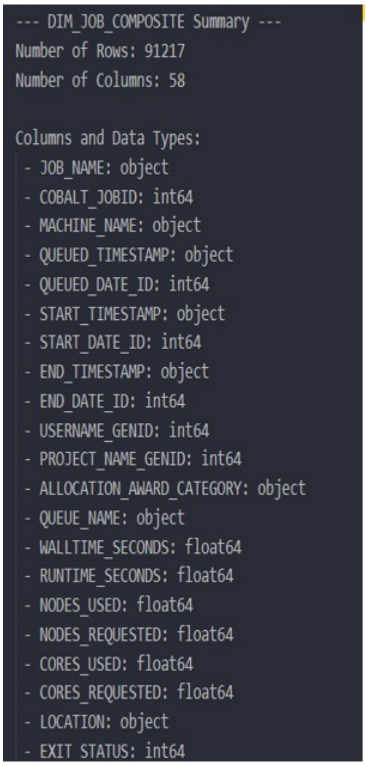
A thorough understanding of Theta's hierarchical hardware organization and its DragonFly network topology underpins the ability to correlate job and hardware failures effectively. It helps distinguish between localized node-level issues and broader infrastructure-level problems. Equipped with an accurate mapping of components to nodes, along with knowledge of how the DragonFly topology influences communication patterns, analysts can confidently link hardware failures to the jobs they affect.

This architectural context sets the stage for the ensuing sections, where we examine the datasets, describe the data preparation steps, and delve into the methodologies for correlating job and hardware logs. Armed with knowledge of Theta's structure, we can more precisely and confidently draw connections between the physical state of the system and the computational results observed in job logs.

# Chapter 4

# Datasets and Data Preparation

The correlation analysis between job logs and hardware failures relies heavily on the quality, completeness, and consistency of the underlying datasets. For this study, two primary datasets were used: the Job Logs (`DIM_JOB_COMPOSITE`) and the Hardware Error Logs (`THETA_HARDWARE_ERROR`). Both datasets were collected over the year 2019 from the Theta supercomputer. Before attempting correlation, significant data preparation steps—timestamp standardization, handling missing values, data type enforcement, and data cleaning—were undertaken to ensure the integrity and reliability of the analysis.



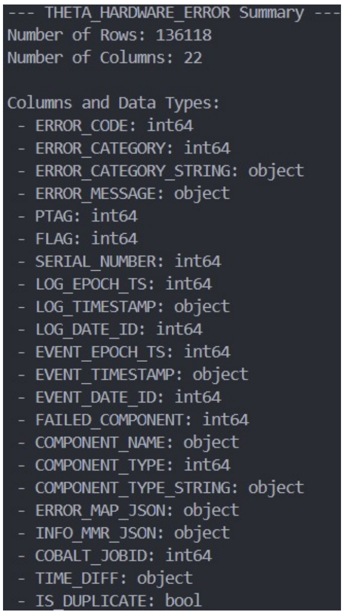
```
--- DIM_JOB_COMPOSITE Summary ---
Number of Rows: 91217
Number of Columns: 58

Columns and Data Types:
 - JOB_NAME: object
 - COBALT_JOBID: int64
 - MACHINE_NAME: object
 - QUEUED_TIMESTAMP: object
 - QUEUED_DATE_ID: int64
 - START_TIMESTAMP: object
 - START_DATE_ID: int64
 - END_TIMESTAMP: object
 - END_DATE_ID: int64
 - USERNAME_GENID: int64
 - PROJECT_NAME_GENID: int64
 - ALLOCATION_AWARD_CATEGORY: object
 - QUEUE_NAME: object
 - WALLTIME_SECONDS: float64
 - RUNTIME_SECONDS: float64
 - NODES_USED: float64
 - NODES_REQUESTED: float64
 - CORES_USED: float64
 - CORES_REQUESTED: float64
 - LOCATION: object
 - EXIT STATUS: int64
```

Figure 4.1: DIM JOB COMPOSITE dataset

## 4.1 Job Logs (DIM_JOB_COMPOSITE)

The Job Logs dataset contains approximately 91,216 job entries from the year 2019. Each record provides a wealth of details about a particular job's lifecycle:

- **COBALT_JOBID**: A unique identifier assigned by the Theta scheduling system. This key is crucial for linking job-related events to hardware error records when available.
- **START_TIMESTAMP** and **END_TIMESTAMP**: Indicate when the job began and concluded, enabling the calculation of execution intervals.
- **EXIT_STATUS**: Signifies whether the job completed successfully or failed. A value of zero usually indicates success, while non-zero values signal some form of error or abnormal termination.
- **LOCATION**: Provides information on the nodes allocated to the job. Often represented as a range (e.g., [110–119]), these node references must be translated into a node list to facilitate spatial correlation.



```
--- THETA_HARDWARE_ERROR Summary ---
Number of Rows: 136118
Number of Columns: 22

Columns and Data Types:
 - ERROR_CODE: int64
 - ERROR_CATEGORY: int64
 - ERROR_CATEGORY_STRING: object
 - ERROR_MESSAGE: object
 - PTAG: int64
 - FLAG: int64
 - SERIAL_NUMBER: int64
 - LOG_EPOCH_TS: int64
 - LOG_TIMESTAMP: object
 - LOG_DATE_ID: int64
 - EVENT_EPOCH_TS: int64
 - EVENT_TIMESTAMP: object
 - EVENT_DATE_ID: int64
 - FAILED_COMPONENT: int64
 - COMPONENT_NAME: object
 - COMPONENT_TYPE: int64
 - COMPONENT_TYPE_STRING: object
 - ERROR_MAP_JSON: object
 - INFO_MMR_JSON: object
 - COBALT_JOBID: int64
 - TIME_DIFF: object
 - IS_DUPLICATE: bool
```

Figure 4.2: Theta hardware error

## 4.2 Hardware Error Logs (THETA_HARDWARE_ERROR)

This dataset comprises about 136,118 hardware error events from the same time frame (2019). Each record describes an event where the system detected a malfunction:

- **EVENT_TIMESTAMP**: The exact time at which the hardware error was recorded.
- **ERROR_MESSAGE** and **COMPONENT_NAME**: Detail the nature and location of the error. `COMPONENT_NAME` follows a structured notation (e.g., `c1-1c2s10n2`) indicating cabinet, chassis, blade, and node offset.
- **COBALT_JOBID** (if present): Some errors can be matched up with a COBALT_JOBID, allowing a direct linkage to the specific job that was running at the time the hardware issue occurred. This direct correlation greatly simplifies the analysis since it removes

uncertainty. Without COBALT_JOBID, further time-based or spatial correlation methods are needed to infer whether a hardware error influenced a given job.

| Column Name | Data Type | Description |
|---|---|---|
| COBALT_JOBID | INTEGER | Some errors can be matched up with a cobalt_jobid. |

Table 4.1: Data Dictionary of COBALT_JOBID in Theta Hardware Error

## 4.3 Timestamp Standardization

One of the first data preparation steps involved standardizing timestamps. The logs were initially recorded in potentially differing time zones or formats. To ensure consistency, all timestamps—`START_TIMESTAMP`, `END_TIMESTAMP`, and `EVENT_TIMESTAMP`—were converted to a common baseline, such as Coordinated Universal Time (UTC). This standardization is crucial when comparing events across datasets: a hardware error and a job failure that appear simultaneous in UTC might be hours apart if one record is still in local time.

## 4.4 Handling Missing Values

Data integrity demanded a careful approach to miss ing values. Incomplete records—lacking critical fields like COBALT_JOBID, timestamps, or EXIT_STATUS—were re moved or marked for exclusion. While such deletions can reduce the sample size, ensuring that only reliable data points remain was paramount. Some fields may have contained null or NaN values. For EXIT_STATUS, for instance, missing values would undermine any attempt to categorize the job's outcome. Similarly, hardware error events without valid times tamps or node identifiers were deemed unusable. By removing or imputing missing data judiciously, the dataset became more trustworthy..

## 4.5 Data Type Enforcement

Proper interpretation of fields often required enforcing correct data types. Fields representing numeric values, such as EXIT_STATUS or numeric node IDs, were cast to integer types. Timestamps were converted to appropriate date-time objects to facilitate time-based comparisons and calculations. Ensuring that each column in the dataset had an appropriate and consistent data type helped prevent logical errors during analysis. For instance, treating COBALT_JOBID as a string rather than an integer could cause issues when performing joins or searching for specific job IDs.

## 4.6 Data Cleaning and Deduplication

Given the large scale of HPC logging systems, duplicates or redundant entries can sometimes occur. Duplicate records, if not removed, can skew the analysis by making it appear as if certain failures or jobs occurred more frequently than they actually did. The cleaning process involved identifying such duplicates using a combination of unique identifiers and timestamps and removing them. This step helps maintain the statistical integrity of correlation analyses and ensures that the relationships derived from the data reflect reality.

## 4.7 Node Mapping and Parsing `COMPONENT_NAME`

One of the more intricate data preparation tasks was parsing the COMPONENT_NAME field from the hardware error logs. Entries like c1-1c2s10n2 encode the node's physical location within the Theta supercomputer's hierarchical architecture. Each component ID typically encodes four hierarchical levels: cabinet, chassis, blade, and node offset.

For the Theta system:

- There are 24 cabinets in total.
- Each cabinet contains 3 chassis.
- Each chassis has 16 blades.
- Each blade hosts 4 nodes (cards).

To facilitate spatial correlation, these hierarchical identifiers must be translated into a consistent, globally unique node number. After extracting the cabinet, chassis, blade, and node offset values using pattern matching (e.g., regular expressions), a deterministic formula is applied:

$$\begin{aligned} \text{NODE\_NUMBER} = {} & (\text{Cabinet} \times 3 \times 16 \times 4) \\ & + (\text{Chassis} \times 16 \times 4) \\ & + (\text{Blade} \times 4) \\ & + \text{Node\_Offset}. \end{aligned} \tag{4.1}$$

For instance, consider the component ID c1-1c2s10n2:

- Cabinet: 1
- Chassis: 2
- Blade: 10
- Node_Offset: 2

Plugging these into the formula:

$$\begin{aligned} \text{NODE\_NUMBER} &= (1 \times 3 \times 16 \times 4) \\ &\quad + (2 \times 16 \times 4) \\ &\quad + (10 \times 4) \\ &\quad + 2 \\ &= 362. \end{aligned} \tag{4.2}$$

This standardized node numbering scheme ensures that the hardware error logs and job logs reference nodes consistently. Without a unified node numbering system, it would be impossible to determine if, for example, `c1-1c2s10n2` corresponds to node 362 or another arbitrary numbering scheme. By mapping each component ID in this manner, hardware errors recorded at a given node can be directly matched against job logs that utilize those node numbers. This alignment is crucial for accurate spatial correlation, as it links the hierarchical physical structure of the system with the more straightforward node references found in the job logs.

## 4.8 Expanding Job Location Ranges

Job logs frequently specify a location range rather than enumerating all nodes used. For example, a job might run on nodes 110 through 119. To properly correlate hardware errors, these ranges were expanded into an explicit list of nodes: [110, 111, 112, ..., 119]. Each node in this list was then matched to a uniform node numbering system. This step is essential because hardware error events are often tied to a single node or a small subset of nodes, and only by expanding the job's node allocation range can we identify if the faulty node was part of that job's resource pool.

## 4.9 Ensuring Temporal Alignment

Beyond just standardizing timestamps, temporal alignment is about ensuring that the intervals defined by job start and end times can be directly compared to the single-point timestamps of hardware errors. This involves creating time-based indexes, sorting events chronologically, and preparing efficient lookup structures (e.g., interval trees or binary indexed searches) that quickly determine whether an error event falls within a job's execution window.

## 4.10 Quality Assurance Checks

After these data preparation steps, a series of quality assurance checks were implemented. For example, random samples of the merged dataset were inspected to ensure that join operations worked as intended and that no obvious data anomalies remained. The distribution of jobs and errors over time was visualized to ensure that the data spanned the entire year and that there were no unexpected gaps or spikes that could indicate logging malfunctions.

## 4.11 Outcomes of Data Preparation

After completing these steps, the final prepared datasets were well-structured, consistent, and suitable for correlation analysis. Jobs and hardware errors were aligned in a common timeline (UTC), referenced the same nodes, and contained no critical missing values. These carefully prepared datasets provided a robust foundation for subsequent analyses—direct correlation, time-based correlation, and eventually spatial correlation—ensuring that any patterns found would be as accurate and meaningful as possible.

In essence, the data preparation stage transformed raw, somewhat messy HPC logs into a coherent and analyzable corpus. Without this rigorous groundwork, attempts at correlation would likely yield incomplete or misleading conclusions. By ensuring data integrity and clarity, we set the stage for effective correlation analysis that can truly inform predictive maintenance strategies and better HPC operational policies.

# Chapter 5

# Criteria for Job and Hardware Failures

Before correlating job logs and hardware errors, it is necessary to establish clear and consistent criteria for what constitutes a failure in each domain. Without proper definitions, the analysis could conflate normal behavior with problematic conditions, leading to ambiguous or misleading correlations. This section outlines the criteria for identifying failed jobs and hardware failures, ensuring that the dataset used for correlation is both relevant and representative of genuine reliability issues.

## 5.1 Criteria for Job Failure

### 5.1.1 Exit Status (`EXIT_STATUS`)

The simplest and most common indicator of job failure is a non-zero exit status. A well-behaved HPC job that completes its intended workload and exits normally will typically return a zero code, signaling success. Non-zero exit codes suggest that the job terminated prematurely, encountered an unhandled exception, ran out of allocated memory, or exceeded the allotted runtime without completing its tasks.

### 5.1.2 Abnormal Termination Signals

Beyond the `EXIT_STATUS` field, some jobs may fail due to external signals, such as `SIGKILL` or `SIGSEGV` (segmentation fault). If such signals are recorded, they offer additional evidence of a job failure. Although these signals often result in a non-zero exit code as well, including them explicitly ensures that no subtle failures are missed.

### 5.1.3 Exceeding Walltime

In HPC environments, each job is typically assigned a maximum walltime. If a job exceeds its allocated time, the scheduler may forcibly terminate it. While the job's exit code might reflect this condition, explicitly recognizing exceeded walltime as a failure criterion helps differentiate between natural job completion and forced termination due to resource constraints.

### 5.1.4 Error Messages and Logs

Some jobs generate their own error logs, compiler messages, or runtime warnings indicating failures at a higher level. While these messages alone do not always constitute a failure (the job may recover or handle the error gracefully), recurring patterns of such messages correlated with a non-zero exit code or abnormal termination can strengthen the evidence of a true job failure.

### 5.1.5 Categorizing Jobs Based on Criteria

- **Successful Jobs:** `EXIT_STATUS = 0`, no abnormal signals, and no critical error messages.
- **Failed Jobs:** `EXIT_STATUS` $\neq 0$, or the presence of abnormal termination signals, or a known error condition (like exceeded walltime) that prevented normal completion.

By consistently applying these criteria, we achieve a clean separation of failed jobs from successful ones, paving the way for meaningful correlation with hardware errors.

## 5.2 Criteria for Hardware Failure

On the hardware side, failures are more nuanced. HPC hardware includes a variety of components: processors, memory modules, network interfaces, storage drives, and voltage regulators, among others. These components can fail intermittently (transient errors) or repeatedly until replaced (persistent errors).

### 5.2.1 Error Codes (`ERROR_CODE`) or Messages (`ERROR_MESSAGE`)

Hardware logs often contain standardized error codes that correspond to particular fault conditions. For instance, repeated multi-bit memory errors or uncorrectable ECC errors typically signal serious hardware issues. Similarly, CPU core faults, persistent ECC errors, and failed network ports are recorded as distinct messages.

### 5.2.2 Transient vs. Persistent Errors

Some hardware errors may not immediately cause job failures. Transient errors can sometimes be corrected by ECC logic or driver retries without the user noticing any visible impact. However, if the same component reports errors repeatedly over a short period, this pattern might reflect a deteriorating piece of hardware. Persistent errors, those that reoccur or escalate over time, are more likely to lead to job failures.

### 5.2.3 Component Criticality

Another dimension involves the criticality of the component affected. A memory DIMM that generates multiple correctable errors may eventually fail entirely, causing one or more jobs running on that node to crash. In contrast, a single transient network error on a non-critical path might not affect any jobs. Thus, while all logged hardware errors are examined, repeated or severe errors are weighed more heavily.

### 5.2.4 Descriptive Failure Messages

Error messages like "Memory ECC Uncorrectable Error on Node X" or "CPU Core Fault Detected on Node Y" enable categorization of hardware failures into classes (memory errors, CPU faults, network issues). Classifying hardware errors allows targeted correlation: if memory-related errors frequently precede job failures, it guides maintenance priorities.

### 5.2.5 Categorizing Hardware Failures

- **Transient Hardware Errors:** Single or sporadic errors that might be corrected automatically and do not necessarily coincide with job failures.
- **Persistent Hardware Errors:** Repeated occurrences of similar errors in the same component over time, indicating escalating hardware issues.
- **Critical Hardware Failures:** Errors severe enough (based on error codes or messages) to immediately impact ongoing computations and likely cause job terminations.

## 5.3 Error Categories

In addition to defining what hardware failures are, we can classify them by type and severity. Understanding the categories of hardware errors recorded in the system logs provides insight into their seriousness, potential impact on system stability, and the urgency of intervention. Below are several common categories of hardware errors encountered in HPC environments like Theta:

### 5.3.1 GHAL_ARIES_INFORMATIONAL

- **Description:** Informational errors represent non-critical events or status updates logged by the system. They do not directly affect performance or stability.
- **Importance:** While not serious, these logs serve as alerts or advisories for system administrators. They can highlight conditions worth monitoring, even if no immediate action is required.

### 5.3.2 MCE_ERROR (Machine Check Exception)

- **Description:** MCEs signal serious hardware issues detected by the processor, potentially involving CPU, memory, or I/O devices.
- **Importance:** These are critical errors that may lead to system instability or crashes. Immediate investigation is recommended to prevent data loss or downtime.

### 5.3.3 GHAL_ARIES_CORRECTABLE_MEMORY_ERRORS

- **Description:** These errors indicate memory problems automatically corrected by the system, often using ECC (Error-Correcting Code) memory.
- **Importance:** Although not immediately harmful, such events may suggest underlying hardware degradation or environmental factors affecting memory reliability. Monitoring these can help preempt more serious failures.

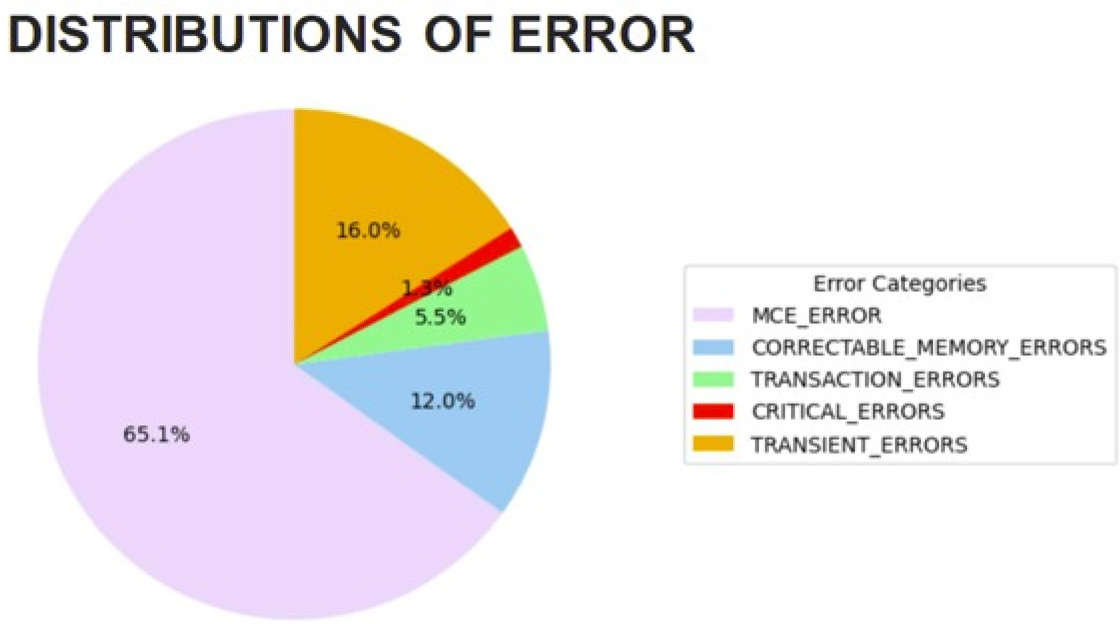


Figure 5.1: Error categories

### 5.3.4 GHAL_ARIES_TRANSIENT_ERRORS

- **Description:** Transient errors are temporary and may resolve themselves with retries or minimal intervention.
- **Importance:** They may not pose an immediate threat, but repeated occurrences could signal emerging hardware instability or intermittent connectivity issues.

### 5.3.5 GHAL_ARIES_TRANSACTION_ERRORS

- **Description:** Occurring during data transactions within the Aries interconnect network, these errors indicate issues in data transmission between nodes or components.
- **Importance:** Such errors can affect data integrity and throughput. They point to potential connectivity or hardware issues in the network fabric that might need focused diagnostics.

### 5.3.6 GHAL_ARIES_CRITICAL_ERRORS

- **Description:** Critical errors represent severe, system-level issues that can cause significant disruptions, such as crashes, data loss, or major performance degradation.
- **Importance:** These demand immediate intervention. Rapid identification and remediation are necessary to prevent extended downtime or further damage.

## 5.4 Aligning Failures for Correlation

Defining these criteria forms the basis for effective correlation. By first filtering and classifying failures accurately, we avoid conflating benign events with serious issues. This careful categorization ensures that when we align job and hardware failures in time and space, the relationships we discover are genuinely meaningful rather than coincidental.

## 5.5 Impact on Predictive Maintenance

With robust failure criteria, the correlation analysis can yield actionable insights. If repeated memory ECC failures predict a surge in job failures, system administrators can schedule proactive maintenance—offlining nodes, running diagnostics, or replacing components—before catastrophic failures occur. Understanding which hardware errors generally do not lead to job failures prevents overreactions and ensures that maintenance resources target truly problematic components.

## 5.6 Conclusion on Failure Criteria

Clear, consistent criteria for identifying job and hardware failures form the foundation of a meaningful correlation study. By rigorously categorizing failures, we create a stable platform on which to build predictive models and maintenance strategies. Subsequent analyses—direct correlation, time-based correlation, and spatial correlation—will rely on these well-defined standards, ensuring that discovered patterns accurately reflect the interplay between hardware health and HPC job outcomes.

# Chapter 6

# Methodology: Direct and Time-Based Correlation

With the datasets cleaned, criteria established, and architectural context in place, the next step is to outline the methodology for correlating job failures with hardware errors. Two fundamental approaches guide this analysis: direct correlation via shared identifiers and time-based correlation when no direct linkage is available. Each approach has its advantages and limitations, and together they provide a robust framework for uncovering meaningful relationships.

## 6.1 Direct Correlation Analysis

### 6.1.1 Rationale

Direct correlation leverages the fact that some hardware error logs include a `COBALT_JOBID` field. When a hardware fault is directly tied to the job that was running at the time, establishing a connection is straightforward. Such direct linkages provide the most reliable evidence that a specific hardware error impacted a particular job.

### 6.1.2 Approach

The simplest method is to perform an inner join of the Job Logs and Hardware Error Logs on the `COBALT_JOBID`. This operation returns pairs of records—one job record and one hardware failure event—both bearing the same job ID. By examining the timestamps and nature of the errors, we can determine how often hardware failures coincide with job failures.

### 6.1.3 Example

**Job Log:**

| | |
|---|---|
| **JobID:** | 12345 |
| **StartTime:** | 2024-12-04 10:00 |
| **EndTime:** | 2024-12-04 12:00 |
| **Nodes:** | NodeA, NodeB |

**Failure Log:**

| | |
|---|---|
| **Timestamp:** | 2024-12-04 10:15 |
| **Node:** | NodeA |
| **ErrorType:** | MemoryErr |
| **COBALT_JOBID:** | 12345 |

**Result:** Since both records share `COBALT_JOBID`=12345, and the hardware error occurred during the job's execution on NodeA, we can conclusively link the job failure to the MemoryErr event. This direct correlation confirms that the hardware fault impacted the job.

### 6.1.4 Advantages

- **Clarity:** Direct matches leave little room for doubt.
- **Ease of Implementation:** A simple database join suffices.

### 6.1.5 Limitations

- **Data Dependency:** Relies on the presence of `COBALT_JOBID` in hardware error logs.
- **Incomplete Coverage:** Many errors lack direct job IDs.

Confirmed direct correlations serve as "ground truth" and guide the interpretation of more complex correlation methods. They validate that the approach can indeed detect authentic relationships and help calibrate time-based or spatial correlation techniques introduced later.

## 6.2 Time-Based Correlation Method

### 6.2.1 Rationale

Many hardware errors are recorded without a `COBALT_JOBID`, making direct correlation impossible. However, these errors may still influence jobs running at the same time. If a job failed during the same time window that a hardware error occurred on its allocated node, we can infer a potential link.

### 6.2.2 Approach

For each failed job, we consider its execution interval (from `START_TIMESTAMP` to `END_TIMESTAMP`) and search for hardware errors that:

- Occurred on the same node(s) used by the job.
- Fell within the job's execution window.

Mathematically, if $t_{\text{start}}(J) \leq t(E) \leq t_{\text{end}}(J)$ for job $J$ and error $E$, and the node aligns, then $(J, E)$ is considered correlated under a time-based approach.

### 6.2.3 Advantages

- **Broader Coverage:** Uncovers relationships hidden to direct correlation.
- **Flexibility:** Can consider pre- or post-job intervals.

### 6.2.4 Limitations

- **Ambiguity:** Temporal overlap does not guarantee causation.
- **Complexity:** Multi-node and multi-job scenarios can obscure direct links.

Time-based correlation extends the analysis beyond explicit job IDs, uncovering subtle patterns.

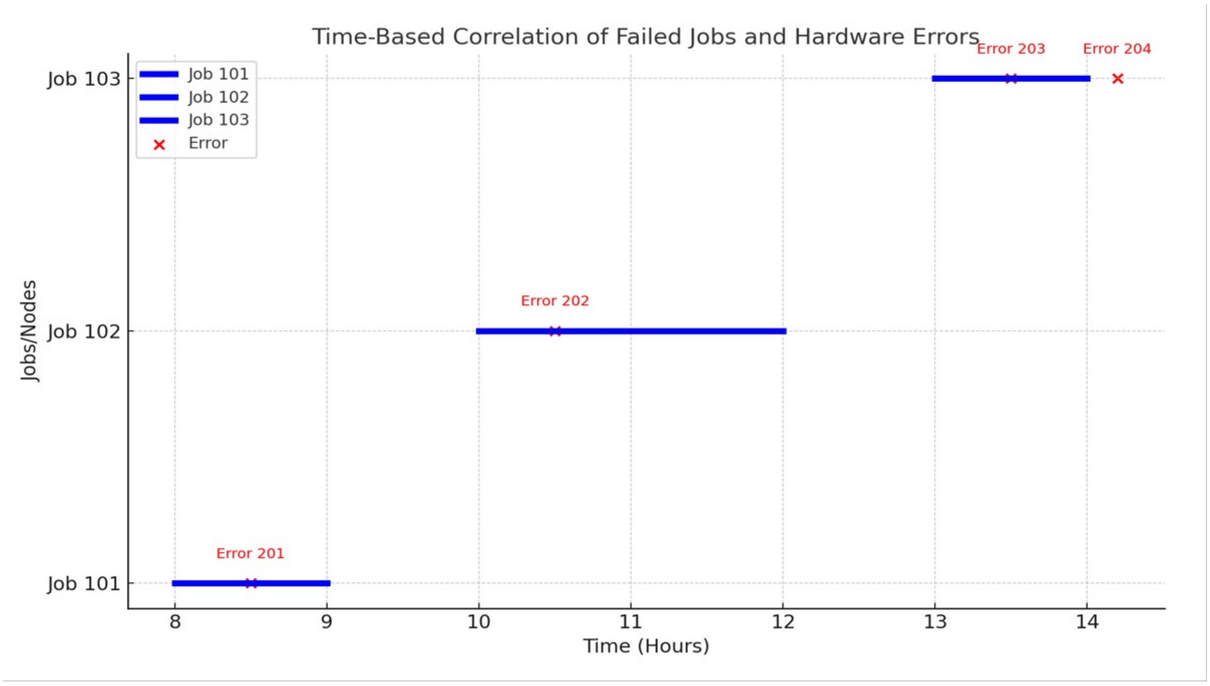


Figure 6.1: Time-based correlation analysis

## 6.3 Combining Approaches

Utilizing both direct and time-based correlation methods provides comprehensive coverage. Direct correlation offers reliable matches (gold standard cases), while time-based correlation fills in gaps. The methodology typically proceeds as follows:

1. **Direct Correlation:** Join datasets by `COBALT_JOBID`. Identify and record job-hardware error pairs with explicit IDs.
2. **Time-Based Correlation:** For the remaining failed jobs, compare their execution intervals with hardware error timestamps to find overlaps.

3. **Confidence Scoring:** Assign weights based on the proximity of timestamps and severity of errors.

4. **Pattern Analysis:** Aggregate correlations to identify recurring patterns, error types, and node groups prone to failures.

## 6.4 Addressing Ambiguities

Time-based correlation can produce false positives. To mitigate this, additional criteria can be used:

- **Severity of Error:** Severe, uncorrectable errors carry more weight.

- **Frequency of Errors:** Repeated similar errors before and during the job strengthen the correlation.

- **Cross-Referencing:** Use environmental sensors, RAS logs, or user reports for confirmation.

## 6.5 Conclusion on Methodology

The combination of direct and time-based correlation methods provides a solid foundation for linking hardware failures to job outcomes. While direct correlation leverages explicit data, time-based correlation infers relationships where explicit linkages are absent. With this dual strategy, the analysis can capture a broad spectrum of correlations, from clear-cut to subtle.

Equipped with these methodologies, subsequent analyses—such as spatial correlation—can yield deeper insights into how hardware reliability truly influences HPC job performance and stability.

# Chapter 7

# Time and Spatial Based Correlation Analysis

While direct and time-based correlations offer significant insights, they do not fully exploit the complex spatial topology of HPC systems like Theta. Given that the system's architecture is organized hierarchically from cabinets down to individual nodes, spatial correlation analysis seeks to refine our understanding of where hardware errors originate and how they affect running jobs. By aligning temporal data (when the errors occur) with spatial data (which components and nodes are involved), we can isolate problematic infrastructure areas and identify recurring patterns of failures.

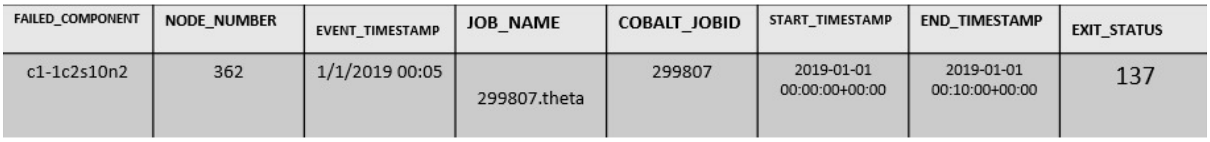
## 7.1 Methodology

### 7.1.1 Time-Based Correlation

Time-based correlation examines whether a hardware error event falls within the execution window of a job. For a given job $J$ starting at $t_{\text{start}}(J)$ and ending at $t_{\text{end}}(J)$, and a hardware error event $E$ occurring at time $t(E)$:

$$t_{\text{start}}(J) \;\leq\; t(E) \;\leq\; t_{\text{end}}(J).$$

If the hardware error timestamp lies within the job's runtime interval, it suggests that the job was active at the time of the hardware fault, potentially exposing the job to unstable conditions.



| FAILED_COMPONENT | NODE_NUMBER | EVENT_TIMESTAMP | JOB_NAME | COBALT_JOBID | START_TIMESTAMP | END_TIMESTAMP | EXIT_STATUS |
|---|---|---|---|---|---|---|---|
| c1-1c2s10n2 | 362 | 1/1/2019 00:05 | 299807.theta | 299807 | 2019-01-01 00:00:00+00:00 | 2019-01-01 00:10:00+00:00 | 137 |

Figure 7.1: Time and spatial based correlation analysis

### 7.1.2 Spatial Correlation

Temporal overlap alone does not confirm that the error impacted the job. Spatial correlation further refines the relationship by ensuring the hardware error occurred on a node actually used by the job. If the set of nodes allocated to job $J$ is $\mathrm{nodes}(J)$, and the hardware error occurred on $\mathrm{node}(E)$:

$$\mathrm{node}(E) \ \in \ \mathrm{nodes}(J).$$

Only when both temporal and spatial conditions are satisfied do we have a strong indication that the hardware failure affected the job in question.

## 7.2 Spatial Mapping of Hardware Errors

As discussed earlier, a key step involves parsing the hardware error `COMPONENT_NAME` fields and converting them into a normalized node numbering system. For instance, `c1-1c2s10n2` is parsed into Cabinet=1, Chassis=2, Blade=10, Node_Offset=2. Using the formula:

$$\begin{aligned}\text{NODE\_NUMBER} = {} & (\text{Cabinet} \times 3 \times 16 \times 4) \\ & + (\text{Chassis} \times 16 \times 4) \\ & + (\text{Blade} \times 4) \\ & + \text{Node\_Offset}. \end{aligned} \tag{7.1}$$

we map `c1-1c2s10n2` to a single node number (e.g., Node 362). Once all hardware errors and job allocations reference a consistent node numbering scheme, correlating them spatially becomes more straightforward.

## 7.3 Job Location Ranges and Node Lists

Job logs often record the nodes they use in a range format (e.g., [110–119]). To facilitate spatial correlation, these ranges are expanded into explicit node lists $\{110,111,112,\ldots,119\}$. If a hardware error is logged at node 113, and that node number corresponds to one in the job's expanded node list, spatial correlation confirms the node-level overlap.

## 7.4 Temporal Alignment and Node-Level Correlation

Time-based correlation determines if the error and the job overlap in time. Spatial correlation adds another condition: did the error occur on a node the job was actively using at that time?

$$t_{\text{start}}(J) \ \leq \ t(E) \ \leq \ t_{\text{end}}(J) \quad \text{and} \quad \mathrm{node}(E) \ \in \ \mathrm{nodes}(J).$$

If both conditions hold, time-and-spatial correlation strongly suggests that the hardware error could have directly impacted the job.

## 7.5 Example Record

Consider the following integrated record after data preparation and node mapping:

Table 7.1: Example of Correlated Record

| | |
|---|---|
| **FAILED_COMPONENT** | c1-1c2s10n2 |
| **NODE_NUMBER** | 362 |
| **EVENT_TIMESTAMP** | 2019-01-01 00:05:00+00:00 |
| **JOB_NAME** | 299807.theta |
| **COBALT_JOBID** | 299807 |
| **START_TIMESTAMP** | 2019-01-01 00:00:00+00:00 |
| **END_TIMESTAMP** | 2019-01-01 00:10:00+00:00 |
| **EXIT_STATUS** | 137 |

**Job Details:** The job with `COBALT_JOBID`=299807 ran from 00:00:00 UTC to 00:10:00 UTC on January 1, 2019, and included the node corresponding to NODE_NUMBER=362.
**Hardware Error Event:** A hardware error occurred at 2019-01-01 00:05:00 UTC on `c1-1c2s10n2` (mapped to Node 362).
**Time Correlation Check:** 00:05:00 UTC falls within the job's execution window (00:00:00 to 00:10:00 UTC).
**Spatial Correlation Check:** The job was using NODE_NUMBER=362, the same node where the hardware error occurred. Thus, both temporal and spatial conditions are met.
**Job Outcome:** The job's `EXIT_STATUS`=137, a non-zero value indicating that the job did not complete successfully. Given that a hardware error arose mid-execution on a node the job actively used, it is reasonable to suspect that the hardware anomaly contributed to the job's failure.

## 7.6 Handling Node Offset and Hierarchies

Theta's hierarchical organization (cabinets, chassis, blades, nodes) allows spatial correlation to occur at multiple levels. Blade-level or chassis-level aggregation can reveal broader hardware hotspots, providing systemic insights that go beyond individual nodes.

## 7.7 Addressing Multi-Node and Multi-Job Complexity

Multiple jobs run simultaneously, and some hardware errors may affect shared resources impacting multiple nodes at once. Clusters of simultaneous job failures may highlight systemic hardware issues at higher hierarchical levels. Conversely, recurring node-level failures suggest localized hardware problems requiring targeted intervention.

## 7.8 Integration with DragonFly Topology

Theta's DragonFly network topology introduces another spatial dimension. Correlating hardware errors with particular network segments can highlight areas of the interconnect fabric prone to failures. If certain global links degrade frequently, jobs communicating over those links may fail more often, guiding maintenance efforts to these critical network components.

## 7.9 Advantages of Time and Spatial Correlation

- **Increased Confidence:** Adding spatial criteria reduces coincidental matches, improving correlation reliability.
- **Identification of Hotspots:** Spatial aggregation helps pinpoint nodes, blades, or chassis that fail disproportionately.
- **Informed Scheduling:** Awareness of error-prone components allows schedulers to steer critical workloads away from them.

## 7.10 Challenges and Limitations

- **Parsing Complexity:** Extracting node indices from hierarchical identifiers is error-prone.
- **Resource Intensive:** Large datasets may require efficient indexing or parallel processing.
- **Transient Shared Failures:** Some failures do not map neatly to a single node, complicating correlation.

## 7.11 Conclusion on Spatial Correlation

Time and spatial correlation together form a powerful analytical lens. While time-based methods establish when events overlap, spatial correlation clarifies where they overlap. This dual perspective reveals patterns neither could fully capture alone. As these analyses mature, administrators can use insights to prioritize interventions, focusing on the most error-prone areas. Ultimately, these refined correlations support proactive maintenance, enabling a more stable and predictable HPC environment.

# Chapter 8

# Using Interval Trees for Efficient Temporal Correlation

One of the central challenges in correlating job logs with hardware failures in large-scale HPC environments is efficiently determining which jobs were running at the exact moment a hardware error occurred. As HPC systems scale into tens of thousands of nodes and run thousands of jobs simultaneously, the volume of log data grows proportionally. Consequently, a direct, brute-force approach to time-based correlation—checking each hardware error event's timestamp against every job's execution interval—becomes computationally prohibitive. To address this challenge, we introduce interval trees as a data structure that can significantly improve the efficiency of temporal correlation queries.

## 8.1 Motivation for Interval Trees

In the temporal correlation step, we aim to identify the set of jobs active at a given hardware error timestamp $t(E)$. Each job's runtime is represented as an interval $[t_{\text{start}}(J), t_{\text{end}}(J)]$. Without any specialized data structure, determining which jobs were active at time $t(E)$ involves checking whether $t_{\text{start}}(J) \leq t(E) \leq t_{\text{end}}(J)$ for every job $J$.

Consider the scale of the problem: If there are $N$ jobs and $M$ hardware errors, a naïve approach could involve up to $N \times M$ comparisons. With tens of thousands of jobs and hardware errors, this approach could require billions of comparisons, making it infeasible for responsive analysis and iterative exploration. Since HPC centers may run continuous analytics on logs to detect patterns or anomalies, a more efficient strategy is urgently needed.

Interval trees address this inefficiency directly. By preprocessing the job intervals into a specialized data structure, we can answer queries of the form "Which intervals contain this point in time?" far more efficiently, typically in $O(\log N)$ time per query rather than $O(N)$.

## 8.2 Understanding Interval Trees

An interval tree is a balanced binary search tree specifically designed to store intervals and answer overlap queries quickly. Each node in the interval tree represents an interval and maintains additional metadata that facilitates pruning large portions of the search space during queries.

The most commonly stored metadata is the maximum endpoint of all intervals in the node's subtree. With this information, when we search for intervals overlapping a particular point $t(E)$, we can skip entire subtrees if their maximum endpoint is less than $t(E)$, as no interval in that subtree can possibly overlap the desired point. This pruning reduces the number of intervals we check directly, improving query performance dramatically.

## 8.3 Construction of the Interval Tree

Extracting Job Intervals: From the job logs (DIM_JOB_COMPOSITE dataset), each job $J$ has a START_TIMESTAMP and END_TIMESTAMP. After ensuring consistent time zones and formats (such as converting all timestamps to UTC), each job is represented as an interval $[t_{\text{start}}(J), t_{\text{end}}(J)]$.

Building the Tree: The simplest approach to building an interval tree is:

1. Sort all intervals by their start times.
2. Insert them into a balanced binary search tree structure, using the start times as keys.
3. Compute and store the maximum endpoint for each node's subtree.

The construction cost is $O(N \log N)$ due to the need to sort intervals and maintain a balanced tree. Although this is more expensive than a single linear scan, it's a one-time cost. After construction, queries run much faster, and the amortized cost becomes beneficial when handling large datasets or multiple queries.

## 8.4 Querying with Interval Trees

Once the interval tree is built, querying it is straightforward:

Query Operation: For a given hardware error timestamp $t(E)$, we want to find all job intervals that contain $t(E)$.

The query algorithm typically works as follows:

1. Start at the root node.
2. If the root node's interval overlaps with $t(E)$, record this interval as a match.
3. Using the maximum endpoint metadata, decide which subtree(s) to explore:
    - If the left subtree's maximum endpoint is $\geq t(E)$, it may contain overlapping intervals, so we search it.
    - If $t(E)$ is greater than the start time of the root's interval, we also search the right subtree.
4. Continue recursively until all possible overlapping intervals are found.

This approach drastically reduces the number of intervals checked. Instead of scanning all $N$ intervals, the query prunes large parts of the tree and typically only examines $O(\log N)$ nodes, plus a few corresponding to actual matches.

## 8.5 Concrete Example

Scenario: Assume three jobs:

- Job A: [00:00, 00:10]
- Job B: [00:05, 00:20]
- Job C: [00:30, 01:00]

A hardware error occurs at $t(E) = 00 : 07$.
Without Interval Trees:

- Check Job A: 00:07 $\in$ [00:00, 00:10]? Yes.
- Check Job B: 00:07 $\in$ [00:05, 00:20]? Yes.
- Check Job C: 00:07 $\in$ [00:30, 01:00]? No.

We scanned all intervals.
With Interval Trees:

- Build the tree from intervals A, B, C.
- Query with 00:07:
- Find A quickly at the root.
- Check right subtree (because 00:07 >A's start): find B also matches.
- C starts at 00:30, which is beyond 00:07, so it's pruned due to max endpoint checks.

We identify A and B without scanning all intervals linearly.

## 8.6 Integration with Spatial Correlation and Direct Correlation

Interval trees focus solely on improving temporal correlation efficiency. After identifying which jobs were active at $t(E)$, spatial correlation checks if the hardware error's node matches one of these jobs' allocated nodes. This spatial step narrows down which jobs were truly impacted.

If a hardware error log includes a COBALT_JOBID directly, time-based correlation is unnecessary for that particular error. Such direct correlations serve as ground truth. Interval trees handle the more complex cases where the job ID is not available.

In essence, interval trees prevent time-based correlation from becoming a computational bottleneck, enabling large-scale, efficient analyses.

## 8.7 Performance Considerations

Complexity: Constructing the interval tree takes $O(N \log N)$, and each query runs in $O(\log N + K)$, where $K$ is the number of matched intervals, usually small. Thus, effectively $O(\log N)$ per query.

Memory Overhead: Additional metadata and pointers add memory overhead, but this is acceptable given HPC's capacity and the significant speed gains.

Static vs. Dynamic Data: Interval trees excel in static datasets where intervals do not change frequently. HPC job logs are historical and fixed, making interval trees an ideal choice.

Implementation Ease: Numerous libraries and code samples exist, reducing the complexity of adopting interval trees.

## 8.8 Conclusion

Interval trees offer a powerful solution to optimize the time-based correlation step in large-scale HPC reliability analyses. By reducing a potentially $O(N)$-per-query operation to $O(\log N)$, interval trees enable rapid identification of which jobs were running during hardware faults. This speed-up encourages frequent, iterative analyses, real-time anomaly detection, and improved predictive maintenance strategies.

As HPC systems grow in complexity and data volume, efficient data structures like interval trees become increasingly vital. Incorporating interval trees into the correlation workflow equips the HPC community with a robust tool to handle temporal complexity at scale, ultimately fostering more reliable HPC infrastructures and enhancing scientific productivity.

# Chapter 9

# Self-Reflection and Technical Insights

## 9.1 Self-Reflection

Throughout this exploratory process, I have continuously adapted my strategies as I gained a deeper understanding of both the datasets and the computational methods at my disposal. Initially, I began with the most straightforward approach: leveraging direct correlations via `COBALT_JOBID`. This method allowed me to confirm, beyond doubt, that certain hardware failures directly influenced specific jobs. However, as I delved deeper, it became apparent that many hardware errors were not explicitly linked to a `COBALT_JOBID`, thereby limiting the scope of direct correlation. This realization prompted me to seek alternative methods.

My next step was to explore time-based correlation. Here, the idea was to check if hardware errors occurred within the time interval of any running job. While this approach expanded the set of potentially correlated events, it soon became clear that time-based correlation alone was too broad. Multiple jobs could be running simultaneously on different nodes, and an error's occurrence during a job's runtime did not guarantee that it affected that particular job. Without a spatial dimension to pinpoint the exact node, time-based correlation risked generating many false positives and provided insufficient causal evidence.

This revelation led me to incorporate spatial correlation. By mapping hardware errors to specific nodes and verifying that a job was running on the same node at the same time, I significantly refined the correlation. Spatial correlation felt more promising, as it reduced ambiguity and improved the granularity of the analysis. Still, scaling spatial correlation to large datasets introduced performance challenges, especially when performing time-based lookups for many jobs and errors.

To address these performance issues, I considered using interval trees. Interval trees offered a theoretically more efficient way to handle temporal queries, transforming $O(N)$ searches into $O(\log N)$ lookups. I implemented interval trees in my codebase, hoping to handle the large dataset more gracefully. Unfortunately, my current hardware—an aging laptop—struggled to process the vast datasets, even with these optimizations. This technical limitation prevented me from fully realizing the interval tree's potential. In the future, upgrading my laptop or leveraging high-performance computing resources will be necessary for smoother analysis and experimentation.

Additionally, while I managed some data visualizations, my laptop's limited resources again posed a bottleneck. The system became unresponsive during complex visual analytics, underscoring the need for better hardware or more efficient data reduction techniques. Beyond the correlation methodologies themselves, I have a growing interest in applying machine learning models—such as LSTMs—to predict job failures based on critical hardware errors. Since critical errors are the most significant and disruptive, focusing on them can yield valuable predictive insights. This December, I plan to deeply explore different data columns and features to optimize correlation strategies further. Although spatial correlation currently seems the most accurate approach, it remains time-consuming and resource-intensive. Nonetheless, by refining these methods and integrating predictive modeling, I hope to develop a more robust and insightful framework for anticipating hardware-induced job failures in HPC environments.

## 9.2 Technical Insights and Future Work

The correlation analyses—employing direct, time-based, and spatial approaches—yield several technical insights that illuminate how hardware errors relate to job failures and how HPC reliability can be enhanced. These insights serve both as conclusions for the current study and as guideposts for future endeavors seeking to deepen predictive capabilities and inform system design improvements.

### 9.2.1 Key Technical Insights

Direct Linkages are Rare but Valuable: Direct correlation cases, where `COBALT_JOBID` is present in both job and hardware error logs, were relatively scarce. However, these cases provide high-quality ground truth data. They confirm unequivocally that certain hardware failures directly led to job terminations. Such confirmed correlations help calibrate other methodologies, guiding the refinement of time-based or spatial heuristics and validating that the analysis approach can indeed detect authentic relationships.

Time-Based Correlation Enhances Coverage: Incorporating time-based correlation significantly expanded the number of potential matches between hardware errors and failed jobs. While this approach introduced ambiguity, careful analysis and confidence scoring can reduce false positives. By capturing hardware errors not explicitly linked to a single job, time-based correlation surfaces subtle patterns that predict future failures. For example, repeated transient memory errors during a job's runtime may forecast an imminent memory DIMM failure.

Spatial Correlation Pinpoints Problematic Infrastructure: Adding a spatial dimension proved crucial for narrowing down correlations. By mapping errors to specific nodes (or aggregated sets of nodes), the analysis identified hardware hotspots. Certain nodes consistently reported more hardware errors and correlated with higher job failure rates. Identifying these hotspots supports targeted maintenance and can inform scheduling policies that avoid problematic nodes for critical workloads.

DragonFly Topology and Error Distribution: The hierarchical and networked nature of Theta's architecture revealed that errors were not uniformly distributed. Some cabinets, chassis, or network links contributed disproportionately to reliability issues. Understanding this distribution enables HPC operators to prioritize stabilizing the most error-prone segments, potentially prolonging hardware life and enhancing overall system performance.

Complex Interactions Beyond Simple Causation: Not all observed correlations imply direct causation. Some correlations may reflect complex downstream effects or coincide with purely software-induced job failures. Recognizing these nuances encourages integrating additional data sources (e.g., RAS logs, environmental sensors) to disambiguate correlations and form a more complete picture of HPC reliability.

### 9.2.2 Towards Predictive Maintenance

A central motivation of this work was shifting from reactive to predictive maintenance. The insights gained form a blueprint for future predictive frameworks:

Feature Engineering for Predictive Models: The correlations discovered can serve as input features for machine learning models that forecast hardware failures. Combining metrics like memory ECC error frequency and job failure counts per node can produce probability estimates of future node-level failures. Such predictions enable preemptive interventions, improving uptime and resource utilization.

Threshold-Based Alerts and Policies: Automated alerting mechanisms can trigger when correlation patterns meet certain thresholds. For example, multiple transient memory errors followed by minor job slowdowns might prompt draining the affected node of critical jobs and scheduling a diagnostic test. These thresholds translate correlation insights into actionable maintenance policies.

Enhanced Logging and Metadata: Improved logging practices could simplify correlation efforts. Consistently recording `COBALT_JOBID`, adopting standardized naming conventions, and including richer metadata (component serial numbers, firmware versions) would enhance the fidelity of correlation analyses, reduce parsing overhead, and streamline future investigations.

### 9.2.3 Broadening the Analysis Scope

While this study focused on CPU and memory-related failures, HPC systems encompass GPUs, storage arrays, and complex interconnect fabrics. Extending correlation techniques to these components provides a more holistic reliability profile. Similarly, analyzing data over longer timeframes or from multiple HPC platforms can reveal whether certain workloads or configurations predispose systems to specific hardware issues.

### 9.2.4 Integrating External Data Sources

Environmental conditions like temperature, humidity, or vibration can affect hardware reliability. Incorporating sensor data could expose correlations between environmental fluctuations and increased error rates, further refining predictive models. Maintenance logs, too, can anchor correlation analyses in ground truth, confirming whether predicted failures led to preemptive interventions that improved reliability.

### 9.2.5 User Feedback and Domain Expertise

Domain experts, HPC operators, and scientists can help interpret correlation results. Anomalies or unexpected patterns might correspond to known quirks in particular applications or highlight previously overlooked hardware vulnerabilities. Closing the loop with user feedback ensures that correlation analyses inform practical, domain-relevant strategies for maintaining system health and performance.

### 9.2.6 Conclusion and Future Directions

This study lays a foundation for understanding the multifaceted relationships between HPC job logs and hardware failures. By defining clear failure criteria, integrating datasets, and employing direct, time-based, and spatial correlation approaches, we derived actionable insights. We confirmed that certain hardware failures closely align with job outcomes, guiding predictive maintenance and improved HPC operations.

Immediate next steps involve building predictive models from these correlations and integrating them into HPC scheduling and monitoring frameworks. Long-term efforts might standardize logging interfaces, extend analyses to emerging hardware technologies, and leverage advanced machine learning methods—like anomaly detection or survival analysis—to refine predictive accuracy.

Ultimately, correlating job logs with hardware failures goes beyond saving compute hours or preventing downtime. It fosters an environment where scientists trust system stability and focus on scientific discoveries. As HPC systems evolve toward exascale, data-driven strategies connecting application-level behavior to underlying hardware conditions will become integral to HPC design, operations, and sustained performance.

# Chapter 10

# Combined Correlation and Component/Temporal Analysis

## 10.1 Combined Correlation Results

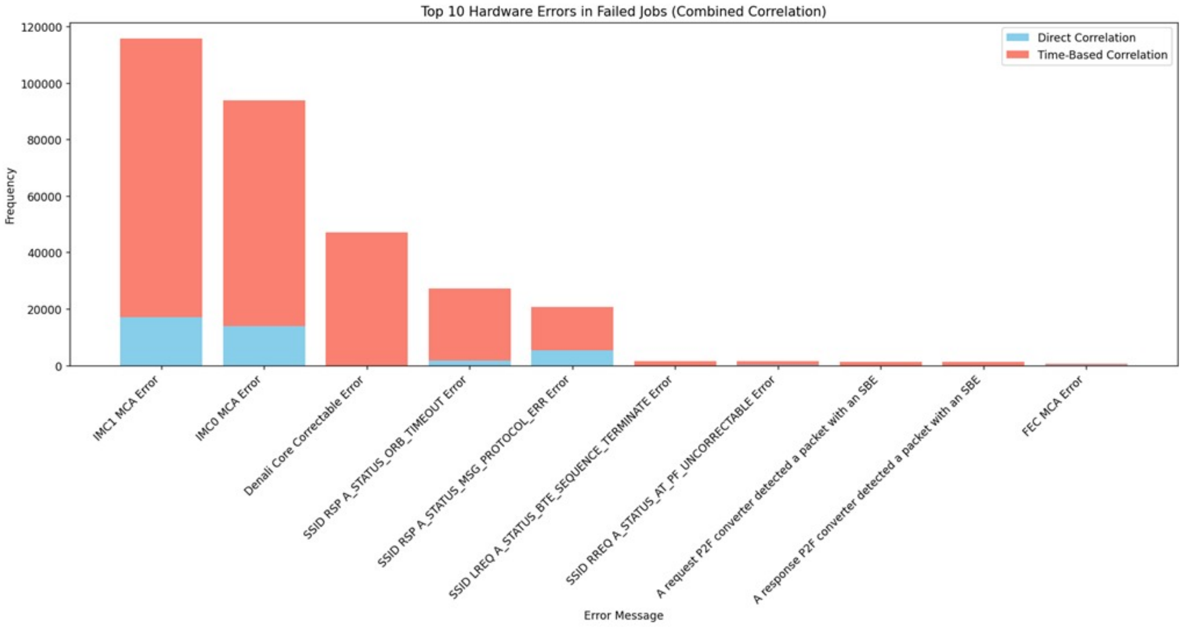


Figure 10.1: Top 10 Hardware Errors in Failed Jobs with Combined Correlation.

Total Failed Jobs with Hardware Errors: 27,507
Direct Correlation: 8,134 failed jobs
Combined Correlation: 19,373 failed jobs

$$\text{Percentage} = \left(\frac{27,507}{36,300}\right) \times 100 \approx 75.8\%$$

Approximately 75.8% of failed jobs are associated with hardware errors.

## 10.2 Component Analysis

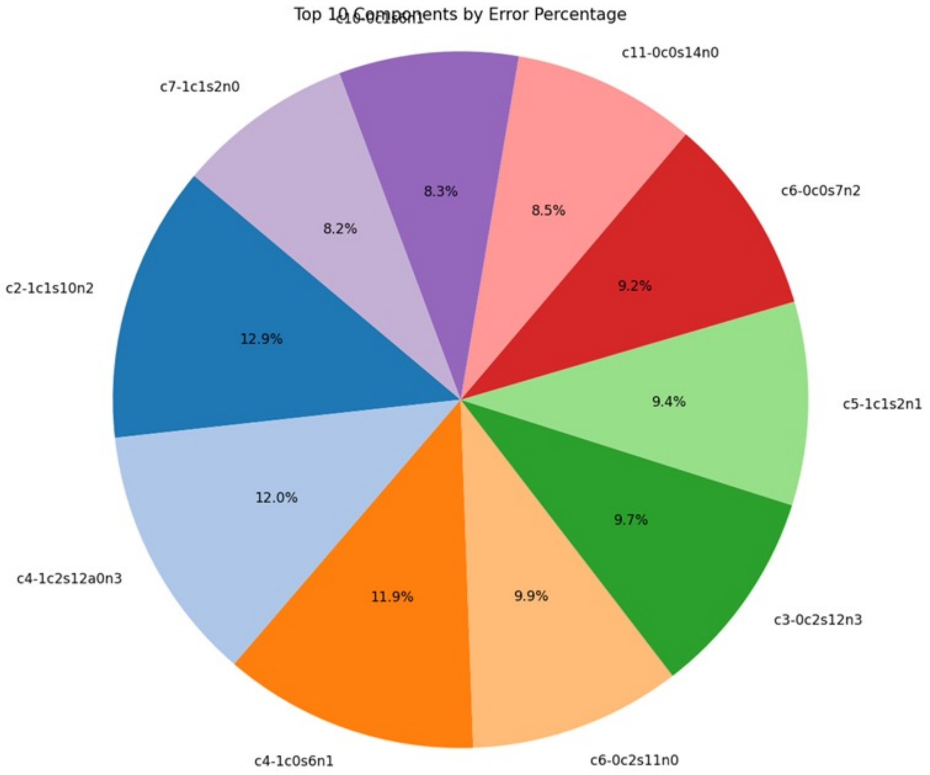


Figure 10.2: Top 10 Components by Error Percentage.

To identify the hardware components most affected by errors contributing to job failures, data was merged from both direct and time-based correlation analyses. Table 10.1 lists the top 10 components with the highest error counts.

| Rank | Component Name | Error Count |
|---|---|---:|
| 1 | c2-1c1s10n2 | 3,790 |
| 2 | c4-1c2s12a0n3 | 3,528 |
| 3 | c4-1c0s6n1 | 3,496 |
| 4 | c6-0c2s11n0 | 2,906 |
| 5 | c3-0c2s12n3 | 2,849 |

Table 10.1: Top 10 Components by Error Count.

## 10.3 Temporal Analysis

### 10.3.1 Daily Distribution of Errors and Failures

By aggregating data daily, we identified fluctuations in error occurrences and job failures over time. Specific periods with high error rates correlated with spikes in job failures.

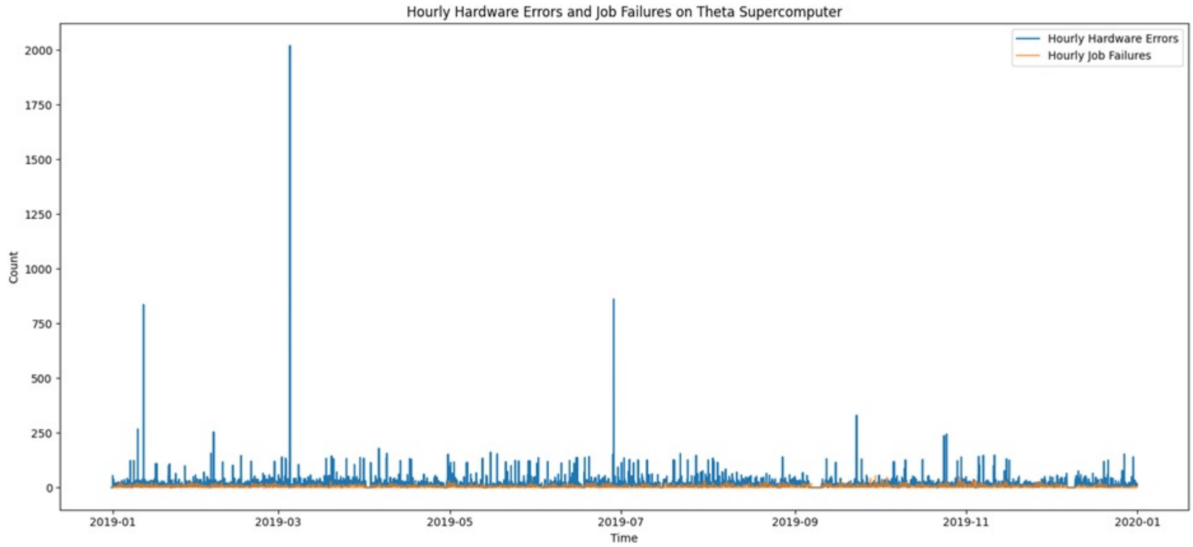


Figure 10.3: Daily Distribution of Hardware Errors and Job Failures.

### 10.3.2 Hypothetical Testing

Pearson Correlation Coefficient (r): 0.4526
Null Hypothesis ($H_0$): No correlation between hardware errors and job failures.
Alternative Hypothesis ($H_1$): Significant correlation exists.

Statistical Results:

- t-statistic: 9.6714
- Degrees of Freedom: 363
- p-value: $7.7592 \times 10^{-20}$

Since p-value $< 0.05$, we reject $H_0$. There is a statistically significant moderate positive correlation.

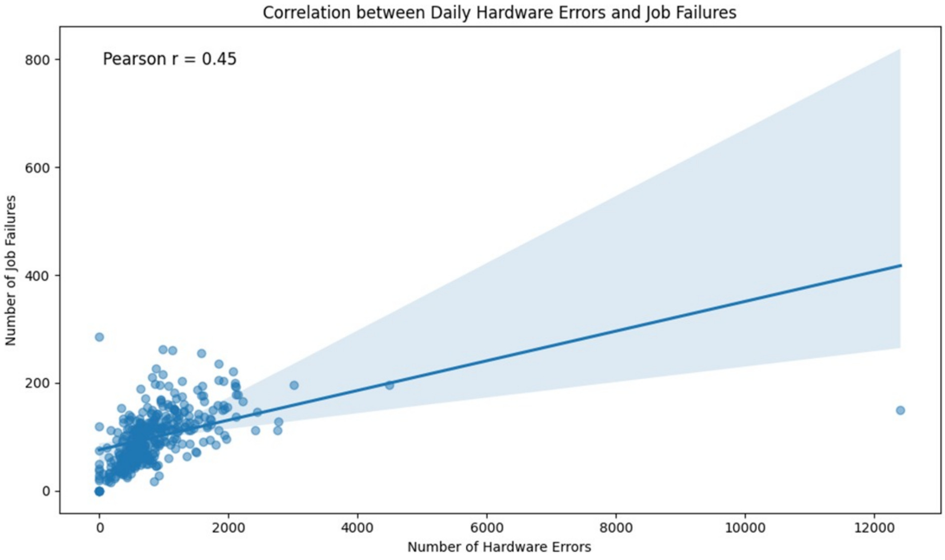


Figure 10.4: Pearson correlation Hypothetical testing

## 10.4 Monthly Trends

This analysis shows the monthly distribution of hardware errors and their trends over time, providing insights into long-term patterns.

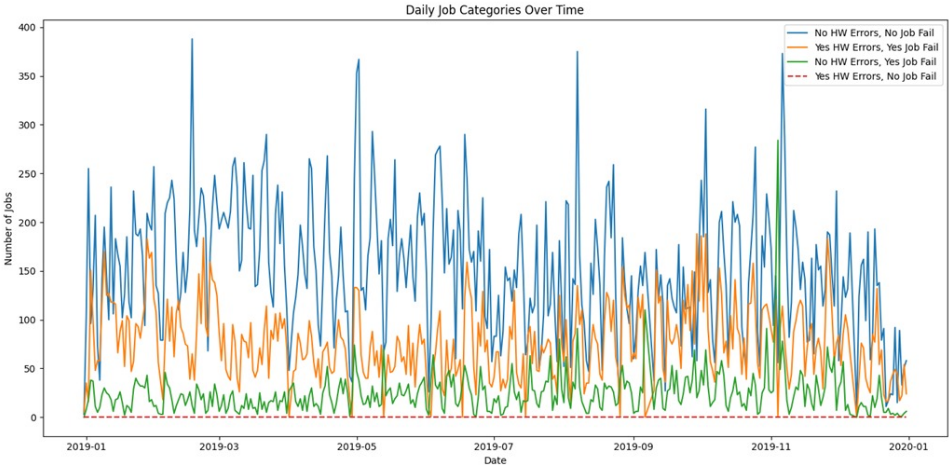


Figure 10.5: Correlation between Daily Hardware Errors and Job Failures.

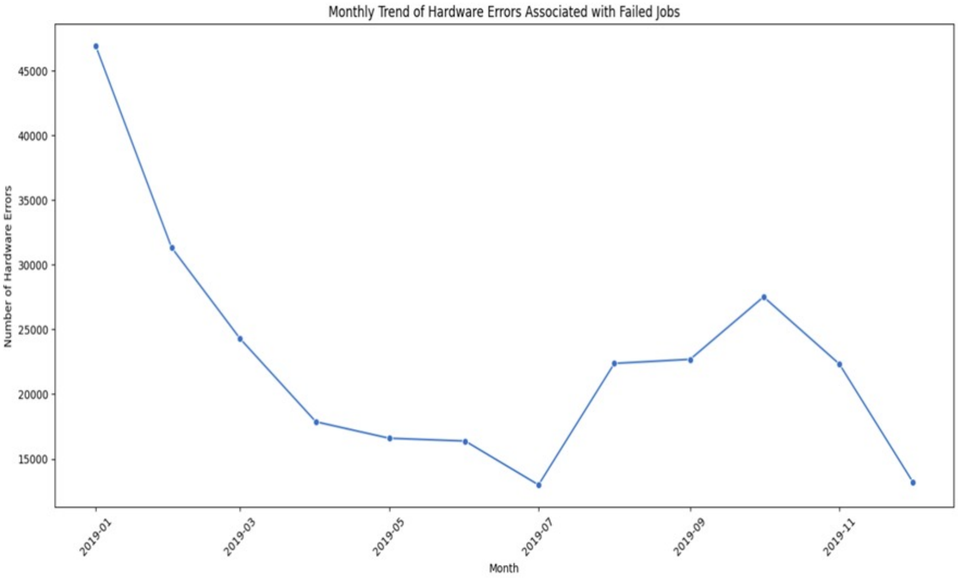


Figure 10.6: Monthly Trend of Hardware Errors Associated with Failed Jobs.

# Bibliography

[1] S.-H. Lim, R. G. Miller, and S. S. Vazhkudai, “Understanding the Interplay between Hardware Errors and User Job Characteristics on the Titan Supercomputer,” in *2020 IEEE International Parallel and Distributed Processing Symposium (IPDPS)*, New Orleans, LA, USA, 2020, pp. 180–190. DOI: 10.1109/IPDPS47924.2020.00028.

[2] Argonne National Laboratory, “Theta Supercomputer Architecture,” [Online]. Available: `https://www.anl.gov/aurora/theta`.

[3] Z. Zheng, *Log Analysis for Reliability Management in Large-Scale Systems*, Ph.D. Thesis, Department of Computer Science, Illinois Institute of Technology, July 2012.

[4] A.-M. Gainaru, F. Cappello, M. Snir, and S. Trausan-Matu, “Failure Prediction under the Microscope: A Closer Look into HPC Systems,” in *SC ’12: Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis*, Salt Lake City, UT, USA, 2012, pp. 1–11.

[5] M. Awasthi, R. Gupta, L. Yu, and W. Bland, “System Logs as a Source of Understanding and Predicting Faults in HPC Systems,” *Journal of Parallel and Distributed Computing*, vol. 120, pp. 43–52, 2018. DOI: 10.1016/j.jpdc.2018.04.011.

[6] D. Tiwari, S. Gupta, and S. Bagchi, “GUARD: Guaranteed Reliability Provisioning for Datacenter Systems,” in *SC ’15: Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, Austin, TX, USA, 2015, pp. 1–12. DOI: 10.1145/2807591.2807626.

[7] W. Bland, P. Balaji, D. Buntinas, and R. Thakur, “Evaluating the Impact of Node Failures on the Execution of Large-scale Parallel Codes,” in *SC ’12: Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis*, Salt Lake City, UT, USA, 2012, pp. 1–11. DOI: 10.1109/SC.2012.35.