A Visual Analytics Exploratory and Predictive Framework for Anomaly Detection in Multi-fidelity Machine Log Data

By

FNU SHILPIKA
DISSERTATION

Submitted in partial satisfaction of the requirements for the degree of

DOCTOR OF PHILOSOPHY

in

Computer Science

in the

OFFICE OF GRADUATE STUDIES

of the

UNIVERSITY OF CALIFORNIA

DAVIS

Approved:

Kwan-Liu Ma, Chair

François Gygi

Venkatram Vishwanath

Committee in Charge

2023

Contents

Abstract

A Visual Analytics Exploratory and Predictive Framework for Anomaly Detection in Multi-fidelity Machine Log Data

Maintaining robust and reliable computing systems, especially those that enable breakthrough work in computational science and engineering research, is a critical and challenging task. This dissertation's goal is to build both an exploratory mechanism for pattern identification from historical data and a predictive tool for identifying the large-scale systems' state with a visual analytics framework. Toward this goal, the work processes various system logs such as error logs, job logs, syslogs, console logs, and environment logs. To process environment log data that captures time-dependent phenomena, the work uses functional data analysis (FDA) to use some of FDA's benefits, such as the ability to study the sensitivity to change and maintain the data ordering. The visual analytics approach developed in this dissertation helps simultaneously monitor and review the changing time-series data by using new incremental and progressive FDA algorithms to promptly generate results for streaming time-series data, thus addressing the computational cost problems prevalent in FDA. A scalable visual analytics tool, MELA, identifies patterns and gleans insights from these diverse logs to effectively characterize system behavior and faults over time. A visual analytics machine learning pipeline promptly predicts a user application's exit status and potential errors. The dissertation also introduces a visual analytics solution for data exploration at varying temporal and spatial resolutions by extracting ranges of frequencies from environment/hardware logs using multiresolution dynamic mode decomposition (mrDMD). These frequencies are extracted at multiple resolutions in time and analyzed at each resolution allowing for coarse-grained (over the years) to fine-grained (over hours or minutes) analysis of the time-series data. This dissertation thus introduces faster, scalable, and interactive visual analytics solutions utilizing multiscale, exploratory, predictive, and multiresolution analyses of diverse large-scale system logs, bridging the gap between visual analytics and machine log analysis.

Acknowledgments

This dissertation is dedicated to many important individuals who helped my research journey. First, I would like to thank my research advisor, Dr. Kwan-Liu Ma, for his patience, motivation, enthusiasm, and firm guidance when I needed it. Second, I would like to thank my research advisors from Argonne National Laboratory, Dr. Venkatram Vishwanath, and Dr. Michael E. Papka, who always believed in my abilities and supported me through this journey. Because of their excellent advice and motivation, I could push through my failures and explore potential avenues in my research. I want to thank all my dissertation proposal and committee members, Dr. François Gygi, Dr. Venkatesh Akella, and Dr. Mohammad Sadoghi, for taking the time out of their busy schedules and providing useful feedback.

Also, I would like to thank all of my collaborators. I was very fortunate to have had the opportunity to collaborate with many researchers in various domains. As a result, I gained fantastic experiences and built my research fundamentals. I would like to thank Dr. Bethany Lusch, Dr. Murali Emani, and Dr. Filippo Simini, whose advice and help have made me a better researcher.

I also want to thank all my labmates at VIDI at UC Davis, especially Dr. Takanori Fujiwara, Dr. Senthil Chandrasegaran, and Dr. Tarik Crnovrsanin. The discussions with them were informative and engaging. Finally, I thank my husband, Nischith Chandrashekar, my parents, my family, and my furry friends, Daisy, Byte, and Sophie.

This research used resources of the Argonne Leadership Computing Facility, which is a DOE Office of Science User Facility supported under Contract DE-AC02-06CH11357.

With everyone's support, I could continue pursuing my dream as a researcher.

Chapter 1
Introduction

1.1 Motivation

To ensure normal operation and to extract adequate performance from the hardware systems such as those in an assembly plant or a supercomputer center, various monitoring mechanisms have been introduced to collect data about all aspects of the systems at high frequency in real-time. Therefore, understanding and studying the intricacies of the data procured from these monitoring mechanisms is critical for enabling advancements in research projects that utilize these systems and expect high system availability. High utilization of these systems is compromised if system failures cannot be handled in a reactive and timely manner. Jobs could run for weeks, and failures add significant overhead in already computationally and financially expensive research. Therefore, we study these diverse subsystems' past behaviors and use the data to predict future behaviors.

Our initial effort focuses on a 10+ Petaflops Cray XC40 supercomputer system with Intel Xeon Phi based compute nodes, specifically the Theta supercomputing system deployed at ALCF [4, 65, 96]. The nodes are interconnected with an Aries interconnect in a Dragonfly topology. We use log data from the Cray XC40 supercomputer that can be broadly classified as (i) environment logs, also referred to as SEDC logs, where SEDC (System Environment Data Collections) is a tool used to collect and report environmental data on all Cray systems in real-time, (ii) hardware error logs, and (iii) job logs. The data are essentially raw data from sensors located at a predefined spatial locality and recorded at various temporal resolutions. Therefore, we deal with multi-fidelity large-scale data from diverse sources within the HPC system. For example, the SEDC log data is recorded and stored approximately every 10-30 seconds, so the SEDC

dataset size approaches the gigabyte to terabyte range every few weeks. The hardware error log contains data recorded from various control systems linked with one another within the subsystems. This results in duplicate events that need to be handled during data pre-processing. The hardware error log data ranges in tens of gigabytes, and the job log data size can be hundreds of megabytes for a year. Altogether, these three types of logs are about 5 GB per day. It is a daunting task to process data of these volumes for any analyses. We also analyze data procured from the K computer's [118] rack environment logs on November-17th 2017. The logs contain data from 864 compute racks, with 1,163 different sensor measurements (e.g., CPU temperatures, circuit voltages, fan speeds) collected every 5-minute interval (i.e., 288 timestamps in a day).

1.2 Background

1.2.1 Machine Log Error Analysis

The data procured from the supercomputing systems are primarily raw data captured from sensors, and the size of these logs is typically in the terabytes range. Therefore, we deal with large-scale logs from diverse sources with multiple fidelities. For example, the environment log data is captured at a frequency of approximately 10-30-second intervals. In this case, a few weeks of this log data is hundreds of gigabytes. The hardware and job log data size ranges in tens of gigabytes. This work aims to build a scalable visualization system that allows us to analyze data with temporal and spatial locality enabled by a machine learning back-end prediction mechanism. When it comes to hardware errors, we look at data from multiple control systems, which are linked to one another. Hence, we deal with duplicate events that need to be pre-processed. Our contributions using the supercomputer logs will apply to Cray XC40 systems deployed at various facilities worldwide, including several US Department of Energy (DOE) national laboratories.

Significant efforts have been made to improve system resilience through system log analysis in the past. Failure detection and root cause diagnosis use diverse log sources that could be used to address failures. Survey papers [78, 142] on failure prediction

give an in-depth idea of current fault prediction methodologies and their shortcomings. Past efforts to standardize the pre-processing of error logs as a first step to tackling these issues include a three-step pre-processing method (event categorization, event filtering, and causality related filtering) to reduce the size of log data [192], an online clustering algorithm to represent textual and temporal data in the log files succinctly [77], and an online clustering algorithm that evolves with streaming event patterns [49, 51].

Machine learning algorithms have been used for in situ predictions of HPC system faults. The decision tree machine learning algorithm has been used to predict hard disk failures [52, 139] from statistically significant/filtered parameters. A context-free grammar-based rapid event analysis has been used for online anomaly prediction with lead times of 3 minutes to node failures [26]. The Random Forest machine learning algorithm has been used to predict node [90, 122] and job failures [156] with lead times of up to one hour [122]. A two-stage prediction model has been used to predict disk failures [52] or input parameters [17]; the stages serve to filter out relevant data. In our work, we adopt the use of a 2-stage machine learning pipeline for job exit status prediction where the first stage filters down to failed jobs, and the second stage predicts the exit status of these jobs.

Some of the automatic log analysis tools [31, 49, 51, 64, 114, 192] use correlation analysis, signal analysis, pattern mining, event pattern-based correlations, resilience at application level, and spatial/temporal event analysis. HELO [51] is an event log mining tool that extracts event formats via messaging templates using pattern mining log files from large-scale supercomputers. ELSA (Event Log Signal Analyzer) [49] is a toolkit for event prediction, modeling the normal flow at a stable event state and following the abnormal flow in the event of system failure. It models events based on its behavior through a combination of signal processing and data mining. One of the steps involving log analysis is data filtering. An adaptive semantic filtering (ASF) method [103] efficiently filters the duplicated messages in a log by computing the semantic correlation between two events where the correlation threshold adapts depending on the temporal gap between the events. LogMine [64] is an unsupervised,

scalable end-to-end one-pass framework for the analysis of massive heterogeneous logs. LogDiver [114] provides support for lossless data compression, modeling application failure paths, and cross-validation of models/results. However, LogDiver and LogMiner do not provide visualization support. DeepLog [32] is a deep neural network model that uses stacked Long Short-Term Memory (LSTM) networks for anomaly detection, dynamically updating the models to accommodate changing log patterns. However, it does not have a lead time analysis. RAVEN [130] maps RAS (reliability, availability, and serviceability) logs on a map of the physical system for Titan, which is a Cray supercomputer at Oak Ridge National Laboratory. IBM provides Blue Gene Navigator [97] used by system administrators to monitor the system through basic log statistical visualizations. These tools lack the support for scalable and interactive visual analysis of HPC system logs.

1.2.2 Visual Analytics of Machine Logs

Although there are many automatic log analysis tools, little work has been done with visualization to perform failure analysis and correlations of failures using multifidelity machine log data gathered from various source at system and subsystem levels.

LogAider [31] and LogMaster [42] have generic, easy-to-use visualizations. La VALSE [59] is a tool with support for a scalable user interface. They explore large amounts of hardware data in supercomputers. Fujiwara et al. [46] utilized node-link diagrams and the matrix-based representations with hierarchical aggregation techniques to visualize any type of network topology. Li et al. [102] developed flexible visualization for analyzing the network performance on the Dragonfly network. Although, analyzing one or two types of log data would help interpret a subset of the mechanics of the large-scale computing system, it would not suffice to provide complete insight into comprehending the complexity of such systems. Therefore, we analyze multiple logs (hardware, jobs, environment, syslogs, console logs etc.) for our study.

DCDB Wintermute [123] is an online generic framework implemented on top of the Data Center Data Base (DCDB) monitoring system that enables Operational Data Analysis (ODA) [12]. In this work, multiple log data types, including job, hardware, and

environment logs, are analyzed. However, each type of data is analyzed separately, and there is no correspondence in the analysis and visualization of results of these different types of logs.

1.2.3 Time-series Data Analysis for Functional Data

The ability to instantaneously process and analyze high-frequency time-series data in real-time becomes pertinent to examining various underlying system phenomena to detect and promptly react to system failures or inefficiencies.

High-velocity time-series data is intrinsically functional as it can be represented in the form of curves or surfaces with weak assumptions of smoothness being permitted [174]. Functional data analysis (FDA) is a branch of statistics for analyzing such data [136]. FDA incorporates statistical methodologies to capture the underlying properties and structure of the data. The statistical methods and models in FDA are typically presented in the form of continuous functions [135]. A wide range of methods have been developed to handle functional data, such as functional principal component analysis (FPCA), functional regression models, and functional canonical correlation analysis [135]. As these names indicate, many FDA methods resemble those developed for conventional discrete analysis. However, because FDA handles data with continuous functions, FDA has advantages in studying the derivatives of the data and maintaining the temporal order of data [174]. Recently, FDA has seen tremendous growth with applications in various fields, such as biology, meteorology, medicine, finance, and engineering [3, 117, 175]. While FDA methods provide new capabilities for analyzing time-series data, they often suffer from their high computational costs, which increase with the number of time points. This can be a critical problem with real-world monitoring applications as they keep generating new time points, resulting in infinitely long time series. Although FDA has gained more traction in recent years, discrete analysis (PCA, UMAP, t-SNE etc.) is largely preferred over its functional counterparts. Therefore, there is a need to reduce the computational overheads of using the FDA on streaming time-series data.

1.2.4 Multiscale Time-series Data Analysis

Analysis of multiscale systems by linking microscale to macroscale events, both spatially and temporally, at varying granularities has seen significant innovations in algorithms and methodologies. These multiresolution (MRA) analyses have been popularized through wavelet-based and window-based techniques in both time and frequency (Fourier Transforms) domains. MRA's fundamental procedure is extracting and subtracting spatial or temporal features recursively. Dynamic Mode Decomposition (DMD) [15, 61, 144–146, 167] assembles low-dimensional spatiotemporal modes from a multi-scale dynamic system. DMD algorithm is a combination of spatial dimensionality-reduction technique (Proper Orthogonal Decomposition (POD) [18]) and Fourier Transforms in time. Multiresolution Dynamic Mode Decomposition (mrDMD) [95] merges the concepts of wavelet theory and MRA with DMD, thus naturally integrating the recursive isolation of spatiotemporal modes at multiple scales and creating approximations of larger dynamic models. In recent years, both DMD and mrDMD have served as powerful means for studying the dynamics of nonlinear systems in fields including fluid mechanics [95, 144, 167], financial analysis [111], control systems [134] neuroscience [15], streaming analysis [68, 132] and denoising [30, 146], as well as foreground and background separation in video analysis [95, 112]. The benefits of the intrinsic multiscale spatiotemporal approximation of this technique can be leveraged for analysis of large-scale machine logs

Chapter 2
Error Log Anomaly Detection in Large Scale High Performance Computing Systems: A Survey

2.1 Introduction

Large scale high performance computing (HPC) systems perform the crucial task of solving mission-critical problems in a wide range of fields like physics, climate, biology, chemistry, national security, etc. These computing systems' capacity, complexity, and dynamicity have shown a significant upward trend in growth over the last few years. Therefore, it becomes crucial to fortify the resilience of these systems by developing fast and effective system failure preventative measures through analysis of system log data. The recent directions in error log analysis focus on several areas. Some measures focus on online or reactive measures that study the data as the information is recorded. Others focus on studying historical log data to derive patterns that are used to categorize future data recordings. Predicting these system failures has also garnered more interest. Numerous machine learning and deep learning methodologies are employed to quickly identify and react with proactive measures in the event of a system anomaly. The error log analysis methods developed also focus on the type and nature of the data. For example, studies focus specifically on hardware, software, application, user-specific, or network failures. There have been extensive surveys developed in the past years on system log analysis that compare the analysis methods, impact, and generalizability of the existing works. In this chapter, we extend the efforts of the previous surveys [6, 35, 67, 79, 142, 155, 160, 182] by incorporating relevant efforts from the fields of visualization [14, 45, 46, 87, 99, 102, 125, 150, 189], cloud computing [89, 98,

148, 187], and infrastructure [8, 153] systems while preserving the same terminologies considered standard in the previous survey in the HPC field. Furthermore, we aim to identify opportunities for growth and bridging gaps by curating a list of log analysis surveys and research methodologies from these wider areas (i.e., cloud, infrastructure, visualization, etc.) to encourage cross-domain adaption in future works.

Our work aims to provide a survey in system log analysis beginning from studies between 2007 and 2022. Our survey targets log analysis not only in HPC systems but also log analysis in cloud, infrastructure, visualization, and warehouse domains. Here, the common denominator is maintaining stable system serviceability, reliability, and availability [191] through analysis of large-scale, high-volume, and high-frequency data in these environments. The processing methodologies adopted in one of these environments [54] may provide valuable insights and knowledge that could be advantageous when adapted to another environment dealing with the same data type. Our contributions are as follows:

- A literature survey on log analysis and outcomes in HPC, cloud, visualization, and infrastructure systems.
- A literature survey of papers ranging from 2007 to 2022 identifying both state-of-the-art and contemporary developments.
- Discussion of the gaps in contemporary state-of-the-art practice and future research opportunities.

2.2 Background and Definitions

Log data are generated by timely insertions of descriptive and critical reports reflecting a system state into files. Creating these files is called logging and serves as a fundamental step for further processing and log analysis techniques [67]. This section provides general terminologies consistently used in log analysis surveys by Avizienis et al. [6], Salfner et al. [142], and Juak et al. [79].

2.2.1 Logging Terminologies - Faults, Errors, Symptoms, and Failures

- A **failure** is defined as “an event that occurs when the delivered service deviates from correct service.” [6]. Here, failure refers to an abnormal system state observed by a user or a sensor component. A failure occurs only when this abnormality spills over to the output resulting in erroneous results. E.g., node failures in supercomputers leading to node unavailability, failure events in IBM Blue Gene systems are tagged as FATAL.

- On failure of a system, "undetected or detected errors may cause out-of-norm behavior of system parameters as a side effect. We call this out-of-norm behavior a **symptom**." [6] In the context of fail slow errors, symptoms are faults converting from one form to another, resulting in a cascading chain of errors and root causes [58].

- An **error** is a deviation from a current stable system state of the resulting in deviation of a correct state, i.e., "things going wrong" [6]. Errors may not lead to failures, however, “an error is the part of the total state of the system that may lead to its subsequent service failure.” [6]

- “**Faults** are the adjudged or hypothesized causes of an error, root causes of errors"; errors are “manifestations of faults” [6,79]. Usually, faults are not instantly observable and have been classified in literature as transient, intermittent, and permanent faults.

Although these terminologies may not be evident or utilized in cross-domain platforms, the problem categories can be easily identified based on the definitions provided here.

To assess the quality of the prediction methods some common metrics used are, precision (*pr*), recall (*rec*), and accuracy (*acc*):

$$pr = \frac{TP}{TP+FP} \quad rec = \frac{TP}{TP+FN}$$

$$acc = \frac{TP+TN}{TP+FP+FN+TN}$$

where TP is the number of true positives, i.e., the number of correctly predicted failures; FP is the number of false positives, i.e., the number of jobs wrongly predicted as failures; FN is the number of false negatives; and TN is the number of true negatives [152]. "Precision is the fraction of the jobs that were predicted as job failures that were correct. Recall is the fraction of the job failures that were correctly predicted. Accuracy is the ratio of correct predictions to the total number of predictions. Usually, an increase in precision leads to a decrease in recall and vice versa" [152].

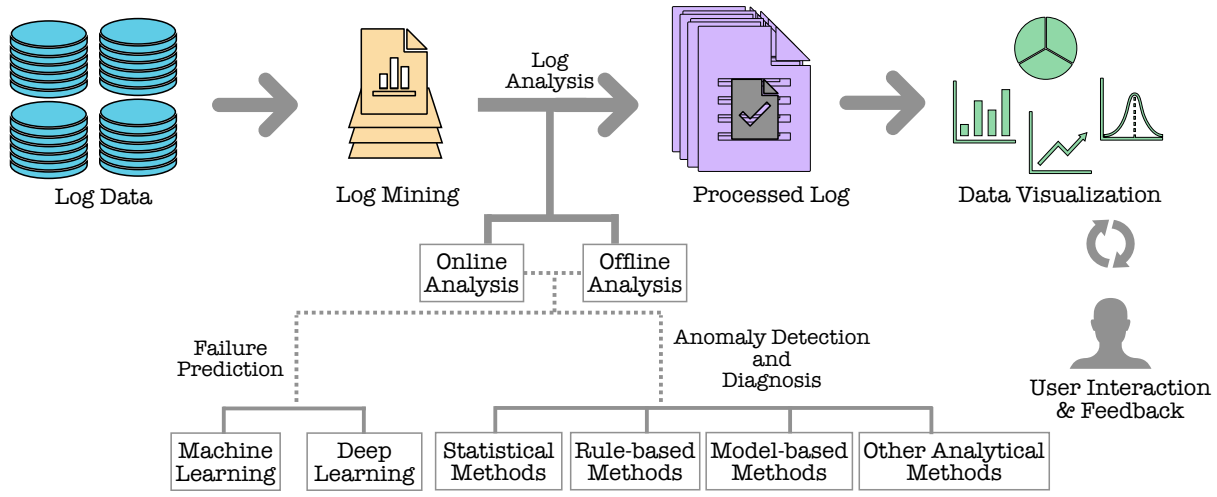

Figure 2.1: An overall workflow for log analysis

Fig. 2.1, shows a typical workflow involved in log data analysis. Traditionally the logging files generated are procured and stored in a wide variety of databases that are tuned to address the various types of data (e.g., numerical, text-based, or event-based). These databases store either data in raw or preprocessed formats and mostly utilize state-of-the-art compression techniques [154]. The log data is mined for filtering useful information by eliminating missing data and correcting for noisy information. Log mining and an in-depth log analysis result in processed log data carrying useful patterns and knowledge that facilitate monitoring, designation, and troubleshooting of these system states by domain experts. These final actions are usually achieved with an effective visual analytics frontend with a user-in-the-loop mechanism. These visual analytics systems allow for user or domain expert feedback which is used to update the visualization and guide the further exploration of the results produced

by the log analysis. The log analysis methods can be broadly classified as online or offline methods. Online methods have gained traction in recent years as these allow for fast and reactive analysis of streaming data. Offline methods are reserved for analyzing larger volumes of historical log data to extract and isolate patterns of errors and system behaviors. Failure prediction is another popular technique used for improving system resilience. Recent advances in machine learning and deep learning help predict these apocalyptic failures reactively. Other log diagnostic techniques include purely statistical-based, rule-based, and mathematical-based approaches. Log mining and log parsing methodologies are not in the scope of this survey.

2.3 Analysis of Log Data

We categorize this section based on the previous surveys [67, 79, 108, 142]. Each subsection gives an overview of the analysis methodologies. We categorize these further based on whether the analysis is online or offline, log data types utilized, visualization support, and the application domain (e.g., cloud, infrastructure, HPC, etc.).

2.3.1 Anomaly Detection and Diagnosis

Statistical and Correlation Zaman et al. [186] address the challenges faced while localizing and isolating faults caused in multi-process applications through an online bug diagnostic tool called SCMiner. SCMiner uses data mining and statistical anomaly detection techniques, which include filtering, PCA analysis, optimization (an apriori algorithm, a sequence abstraction method), and buggy function localization (offline function signature mapping), on system call traces to detect faults with a precision of 65% for smaller traces and close to 90% for original and larger traces. FDiag [21] and FDiagV3 [20] is a similar effort on cluster systems hardware and software events. It uses a systematic methodology for rebuilding event order with statistical correlation analysis using components like a message template extractor, a statistical event correlator, and an episode (strongly correlated system events) constructor. FDiagV3, which uses PCA and ICA-based correlation methodologies, showed recurrent failures and faults were prevalent in a small set of nodes and identified eight unknown causes of compute

Table 2.1: Summary of Anomaly Detection and Diagnosis in Log Data (Statistical and Correlation Methods) (Y/N/- stands for Yes/No/Unknown)

Methods	Analysis Category	Data Type	Online	Scalable	Visualization
Zaman et al. [186]	Statistical and Data Mining	Text-based Sequence	Y	Y	N
Harutyunyan et al. [66]	Statistical, graph-based, and Data Mining	Text-based Sequence	Y	Y	N
Gainaru et al. [48]	Statistics and signal processing	Text-based Sequence	Y	Y	N
ELSA [50]	Statistics, correlation and signal processing	Text-based Sequence	Y	Y	N
FDiag [21]	Statistics and correlation	Text-based Sequence	N	Y	N
FDiagV3 [20]	Statistics, correlation, PCA, ICA	Text-based Sequence	N	Y	N
Lu et al. [110]	Statistics, correlation	Text-based Sequence Numerical	N	Y	N
DISTALYZER [121]	Statistical and Machine Learning	Text-based Sequence	N	Y	N

node hangups. An offline log analysis method [110] for diagnosis of root cause related to CPU, memory, network, and disk, in Spark [185] log files used features related to execution time, data locality, memory, and garbage collection. They developed weighted factors based on a statistical analysis approach to decide the probability of root causes.

Gainaru et al. [48] used correlation analysis and dynamic window strategy to extract frequent events irrespective of time lapse between events, thus modeling a holistic system behavior capable of predicting future system failures at 85% precision and 60% accuracy. For each event, they accumulate the probability for a future event that is predicted with the details about all past events. Subsequent studies [49] used signal processing concepts like wavelet transformation and autocorrelation to categorize

system events as periodic, silent, and noisy signals. This approach allows for the automated filtering of signal categories eliminating the need for preprocessing. In addition, their online anomaly detection and correlation methodologies predict future faults. Finally, their toolkit for event log analysis, ELSA (Event Log Signal Analyzer) [50], was improved with an adaptive and efficient prediction module using data mining and signal processing. Signal processing techniques model the normal system behavior and use data mining techniques to identify real-time deviations due to failures. Their process yields a precision of 91.2% and a recall of 45.8%.

Harutyunyan et al. [66] uses streaming log data in a cloud-based application to identify anomalous behavior in an incoming stream. The authors claim that the infeasibility of retrospective analysis owing to large storage and processing considerations makes employing a meta-data model or graph a reasonable approach to identifying the fundamental structure of the data. Furthermore, the graph structure, built using pairwise correlations of event types, represents the statistical structure of the metadata with useful content, thus eliminating the need for retrospective analysis. The work then uses probabilistic mismatch criteria to compute the degree of abnormality, which would constitute a symptom or error in HPC terminology. A usecase showed mismatch scores (i.e., degree of mismatch between run-time and historical scenarios) of 40.9% and 100%. Using a semantic correlation between two events an adaptive semantic filtering (ASF) method [103] is proposed to efficiently filter the duplicated messages in a log where the correlation threshold adapts depending on the temporal gap between the events. A summary of the relevant works is shown in Tab. 2.1

Model-based and Rule-based Methods "Rule-based methods seek to establish a set of rules, i.e., if-then- statements that trigger a failure warning if certain conditions are met. Such rules are (usually) automatically generated from training data" [79] or explicitly determined by domain experts. Model-based methods isolate and capture system behaviors, based on pre-determined rules, in specific prototypes and identify anomalies based on deviations from these normal states.

CloudRaid [109] is a tool used to detect distributed concurrency bugs in cloud

Table 2.2: Summary of Anomaly Detection and Diagnosis in Log Data (Model-Based and Rule-Based Methods) (Y/N/- stands for Yes/No/Unknown)

Methods	Analysis Category	Data Type	Online	Scalable	Visualization
CloudRaid [109]	Model-based	Text-based Sequence	N	Y	N
LogSed [81]	Model-based	Text-based Sequence	Y	Y	N
Jia et al. [80]	Model-based	Text-based Sequence	Y	Y	N
LOGAN [161]	Model-based	Text-based Sequence	N	Y	Y
Das et al. [25]	Rule-based	Text-based Sequence	N	Y	N
Aarohi [26]	Rule-Based Machine Learning (NLP)	Text-based Sequence	Y	Y	N
Watanabe et al. [177]	Rule-Based	Text-based Sequence	Y	N	N
FlipTracker [60]	Rule-Based	Text-based Sequence	N	Y	N

systems (Apache Hadoop2/Yarn, HBase, HDFS, and Cassandra) by mining and re-ordering (using a strict happen-before order) those message orders likely to expose errors or symptoms. They observed that different message orderings shared the same root cause, and the tool successfully identified 8 (3 critical) previously unidentified bugs. LogSed [81] uses a time-weighted control flow graph (TCFG) to model healthy execution flows and detect errors or symptoms (through interleaving multiple threads in logs) by identifying any deviations from this model. Without prior system knowledge, tests on cloud platform logs revealed anomalies with 80% precision and recall on average. A subsequent work [80] developed a hybrid graph model to capture normal executions of inter- and intra-web services and identify deviations in error occurrences. The results of log mining showed 80% precision and 70% recall, and log anomaly detection showed 90% precision and 80% recall on average. LOGAN [161] is an assistive

tool that generates behavioral reference models to promptly identify root causes in the distributed cloud and cluster platforms. Using log correlation and comparison algorithms any deviations from the modeled norm are flagged and stored for future analysis (log alignment). Watanabe et al. [177] developed an online pattern-matching failure prediction method to automatically match log messages that have strong similarity (a similarity-based classification) with error message formats. After messages are classified, their pattern matching method uses Bayes theorem to calculate the probability of a failure occurring in a pattern. This method is format independent, learns patterns in real time, and is message-order independent. They evaluated the predicted failures in a cloud datacenter platform with 80% precision and 90% recall. FlipTracker [60] is a framework to extract resilient code sequences from user applications by tracking error propagation and resilience properties. Using a dynamic data dependency graph and a table tracking corrupted traces, they extract the resilience computation patterns in code sequences and thus help improve application resilience at the user level. Das et al. [25] use statistical machine learning (natural language processing or NLP) methods to predict node failures in a Cray system. They employ a phrase extraction mechanism (TBP or time-based phrases) to isolate node failures. Their evaluation for TBD generated 83% recall rates and 98% precision, and a time before failure (lead time) of up to two minutes. Aarohi [26], a subsequent work, used machine learning NLP-based, specifically a context-free grammar-based rapid event analysis, training and fast inference scheme to detect failures in real time. Their evaluation resulted in 3 minutes of lead times to node failures, 86% Recall, 88% precision, and 80% accuracy on logs from four different Cray systems. DISTALYZER [121] uses statistical and machine learning techniques to compare features extracted from logs of normal and abnormal executions to derive the strongest link between system components and performance and hence determine the root causes of performance issues boosting the performance by 45% and 25%. A summary of the relevant works is shown in Tab. 2.2

Machine Learning and Deep Learning Methods MEPFL [193] is a technique that uses system trace logs for latent error prediction and fault detection in dynamic, asyn-

Method	Analysis	Data Type	Online	Scalable
MEPFL [193]	*Classification (Random Forests, K-Nearest Neighbors, and Multi-Layer Perceptron):* Latent error: recall= 98%, precision= 99.8% faulty microservices: accuracy=93% fault types: recall=95.2%, precision=98.3%	Text-based Sequence	Y	Y
MING [104]	*2-Phase LSTM model, Random Forest Model, followed by a ranking model* Average: Precision=92.4%, recall=64.5%, and F1-score= 75.2%	Text-based Sequence Numerical	N	Y
DESH [27]	*3-Phase methodology LSTM-RNN based model* Average: Precision≥84%, recall=87.5%, and F1-score≥85.7%, accuracy≥83.6%	Text-based Sequence	Y	Y
PreFix [188]	Random Forest Classification Average: recall=61.81% false positive ratio =1.84 * 10^-5	Text-based Sequence Numerical	Y	Y
AirAlert [19]	*XGBoost classifier* Average(%) *component level*: Storage Location: Precision=71.11 recall=100.00 F1-score=83.17 Physical Networking: Precision=69.07 recall=100.00 F1-score=81.71 *Average(%) service-level:* Website Application: Precision=82.75 recall=76.74 F1-score=79.6 Cloud Network: Precision=75.9 recall=67.07 F1-score=71.22	Text-based Sequence Numerical	Y	Y
Soualhia et al. [156]	*Random Forest Classification* Average: accuracy=85.6% precision=94.2% recall=85.9%	Text-based Sequence	Y	-
PRACTICE [126]	*Neural Networks* non-SBE: precision=88% recall=62% SBE: precision=82% recall=95%	Numerical	N	-
LogRobust [190]	*Bi-LSTM* Microsoft Data: Precision=69%, recall=99%, and F1-score=81% synthetic HDFS dataset 20% injection: Precision=92% recall=97% F1-score=95%	Text-based Sequence	Y	-

Figure 2.2: Summary of Anomaly Prediction in Log Data (Machine Learning (ML) and Deep Learning (DL) Methods) 'Y/N/-' represent 'yes/no/unknown'

chronous microservice runtime environments. They use system trace logs of a target application and its faulty versions generated by fault injection to train prediction models that predict latent errors, faults, and fault types of microservice applications. The prediction models are either binary classification, multi-label classification, or single-class classification models (Random Forests, K-Nearest Neighbors, and Multi-Layer Perceptron) deployed at trace-level and microservice-level. They evaluated infrastructure systems of container orchestrator and service mesh and open source microservice applications. Latent error prediction produced 98% recall and 99.8% precision; faulty microservices with a Top-1 had 93% accuracy, and fault types had a 95.2% recall and 98.3% precision. MING [104] is a 2 Phase prediction mechanism to predict the failure proneness of a node in a cloud environment that integrates analysis of multiple data types: an LSTM (long short-term memory) model for temporal data (local node features), a Random Forest model for spatial data (global system features). This is followed by a ranking model to rank the failure proneness of nodes based on the results of the

two-phase models and a cost-sensitive function to categorize nodes as faulty. Their evaluation produced on average 92.3%, 64.2%, and 75.7% in Precision, Recall, and F1-measure, respectively. Desh [27] extends the research [26] by utilizing a three-phase deep learning methodology using LSTM networks to identify supercomputer failures with sufficient lead times. The 3 phases include training to identify likely failure log message chains, retraining the recognized chains with expected lead times, and predicting lead times during testing to predict when a specific node fails. They evaluated their methodology on four HPC datasets and obtained more than 3 minutes of lead time, > 84% precision, > 83.6% accuracy, and > 85.7% F1 score, 87.5% recall rates across all four systems.

PreFix [188] uses datacenter networks' (DCNs) log data and identify if a switch failure occurs during runtime to prevent network device failures. They employ both historical and current switch hardware failure data. They identified a derived feature set consisting of message template sequence, frequency, seasonality, and surge and then used ML (Random Forest) to efficiently filter noises, sample imbalance, and computation overhead. They included data noise in the optimization scheme and extracted the modularity of system log information, thus solving the data imbalance problem. Failures in syslogs show common patterns before failures across devices. They evaluated their method on three switch models from 20 datacenters and obtained an average of 61.81% recall and $1.84*10^{-}5$ false positive ratio. AirAlert [19] is an outage management system to predict outages in cloud services before their occurrence. It predicts outages using a Bayesian network and gradient boosting tree-based classification (XGBoost) and by collecting and correlating alert signals from the entire cloud network through causal relationship inference. The evaluation results are presented in Fig. 2.2. Soualhia et al. [156] developed a failure mechanism that predicts in advance if a job is likely to fail or not, thus reducing the future execution time caused by the rescheduling of failed jobs. They use a month's execution and resource utilization trace data from applications at Google. For predicting job scheduling outcomes using random forests, they obtained a precision of 94.2%, a recall of up to 85.9%, and an accuracy of 85.6%. Using GloudSim,

Google's toolkit used to simulate the similar workload on their computing infrastructure, they obtained a 40% improvement in job failure rates by rescheduling jobs that were predicted to fail. Other studies have used similar approaches to predict job failure in supercomputers [34, 152]. Nie et al. used PRACTISE [126], neural networks to predict GPU soft errors. They do so by studying the dependence between temperature, power, GPU reliability, and GPU soft errors. They found that the relationship between temperature, power consumption, and GPU errors affects the data centers' operations and reliability. Their work provides useful insights into the difference between CPU and GPU utilization and functionality. Their evaluation on the Titan supercomputers' (Oak Ridge National Laboratory) GPU resulted in a precision and recall of 88% 62% for non-SBE (single bit error), respectively, and a precision and recall of 82% 95% for SBE. Zhang et al. work, LogRobust [190], states that the current methodologies in log analysis do not account for unseen log events or log sequences, noise, and evolving log traces but assume that the data is stable with distinct log events. LogRobust first extracts semantic vectors from log events; it then uses an attention-based Bi-LSTM model to extract the weighted context from log sequences based on its importance. Thus, it handles unstable log events and sequences. Their evaluation on a Microsoft industrial dataset yielded a precision of 69%, recall of 99%, and F1-score of 81%. Lin et al. [105] proposed a unified cloud platform employing Hadoop and Spark for batch processing of log data along with data warehouse and analysis tools like Hive and Shark. Spark and Hadoop are used for the distributed batch-processing analysis (with in-built machine learning capabilities) of large-scale data. Their results showed that a unified cloud platform analysis model harnesses the benefits of each platform and should provide high stability, availability, and efficiency. The summary and evaluation results for the relevant works are provided in Fig. 2.2

2.3.2 Visualization in Log Data

2.3.2.1 Visualization in Log Data Types

Large-scale system logs often contain diverse and ambiguous trace messages. The log data traces can comprise of numerical data, text-based, and event-based data. These

Research Solution	Approach		Online Analysis	Visual Analytics Support	Data Type Analyzed			Generalizability
	Goals	ML Prediction/Data Analysis			Numerical Data (e.g., sensor data)	Event Data / Hardware Data	Usage Data / Job Data	
Shilpika et al. (Chapter 6)	Multi-scale and multifidelity end-to-end error log analysis system that visualizes and analyzes job, hardware and environment at varying temporal and spatial resolutions	Uses multiresolution dynamic mode decomposition (mrDMD) technique that depicts high-dimensional data as correlated spatial-temporal modes, to extract modes isolated at specified frequencies and compare modes with customized baselines	N	Y	Y	Y	Y	Cray Systems
Shilpika et al. [150]	An online visual analytics approach for monitoring and reviewing time series data streamed from a hardware system with a focus on identifying outliers by using functional data analysis (FDA).	Developed new incremental and progressive algorithms to generate the magnitude-shape plot. Interactive visualization of analysis using the MS plot with FPCA (functional principal component)	Y	Y	Y	N	N	Applicable to monitoring time series data from any supercomputer system
Shilpika et al. [152]	Multifidelity end-to-end error log analysis system that analyzes job logs, hardware error logs and environment logs at varying temporal and spatial resolutions augmented by a visual analytics tool	2-stage ML (Random Forest Classifier): Stage 1 (Job Pass/Fail) - 65% - (precision = 67%, recall = 95%) Stage 2 (Job Exit Status)- 92.3% Hardware Error Prediction: RNN (accuracy=84.6%)	N	Y	Y	Y	Y	Cray Systems
MELA [151]	A scalable visualization tool for exploratory analysis of reliability, availability, and serviceability (RAS) logs records. It is designed to trace causes of failure events and investigate correlations	Scalable multilayer diagrams for visualizing millions of log records. A scalable asynchronous query engine for interactive querying support.	Y	Y	Y	Y	Y	Cray Systems
DCDB Wintermute [123]	Generic framework to enable online Operational Data Analytics (ODA)	Block system for instantiation of ODA models using a tree representation of the sensor space	Y	Y	Y	Y	Y	Data Center Data Base (DCDB) monitoring system set up required
Nakka et al. [122]	Data mining classification schemes to predict failures in failure and usage data logs with and without the root cause	Decision tree to predict node failure up to one hour in advance Precision = 74% recall = 81%		-	-	Y	-	Tested on Los Alamos National Lab. supercomputing cluster data
Klinkenberg et al. [90]	Node failure prediction for selected time windows using descriptive statistics & supervised ML to create a prediction model from monitoring data	Node failures, both caused by hardware issues and soft lockups Random Forest classifier - Precision = 98 %, recall = 91%		-	Y	Y	N	Nodes with Broadwell and Westmere processors
LogAider [31]	A generic visualizer that mines spatial correlations and temporal correlations over a certain period within a log or across multiple logs	Temporal Event Correlated with precision = 99.9% recall = 99.9 K-means clustering → spatial correlation		Y	-	Y	Y	IBM Blue Gene/Q systems
DeepLog [32]	Deep neural network model that models a system log as a natural language sequence. The system incrementally updates adapting to new log patterns	Error Event Sequence → Deep neural network model with Long Short-Term Memory (LSTM) Precision = 95%, recall = 96%		-	-	Y	Y	Tested on HDFS and Open Stack Log Dat sets
LogMaster [42]	LogMaster is for mining correlations of event and the work proposes event correlation graphs (ECGs) to represent event correlations, and present an ECGs-based algorithm for predicting events	Mining correlations of events with precision = 81.19% recall = 20.73%		N	-	Y	Y	Hadoop cluster logs and Los Alamos National Lab. supercomputing cluster logs

Figure 2.3: Comparative Analysis. 'Y/N/-' represent 'yes/no/unknown'

trace messages often contain insufficient and missing context leading to ambiguous error messages, [127]. Further, as a system evolves, different keywords are used in different log files, making global mappings and interpretations hard. A comparative analysis between our work and a few relevant related works using visualization support is shown in Fig. 2.3. In the next sections we discuss research advancements in log data analysis in the visualization field.

Numerical Data:Visualization system have previously been developed using numerical data to study the large-scale system behavior. Fujiwara et al. [45] developed a multi-coordinated visual analytics system that uses the Dragonfly network to investigate the temporal behavior and optimize the communication performance of a supercomputer. Their subsequent work [46] utilized node-link diagrams and the matrix-based representations with hierarchical aggregation techniques to visualize any type of network topology. Li et al. [102] developed flexible visualization for analyzing the network performance on the Dragonfly network. Shilpika et al. [150] used functional data analysis (FDA) to incrementally and progressively update the streaming time se-

ries data collected from hardware systems with a focus on identifying outliers by using FDA. Li et al. [99] developed a visual analytics framework for analyzing HPC datasets produced by parallel discrete-event simulations (PDES). This framework leverages automated time-series analysis methods and visualizations to analyze both multivariate time-series and communication network data.

Text-based Data:Text-based system logs can be system generated or user generated by an operator or technician. These logs may contain widely varying abbreviations and operator-specific jargon. To help filter and analyse millions of records of such text data consisting of these large inconsistencies several visual analytic approaches have been proposed [14,189]. Brundage et al. [14] use metrics such as word occurrence frequency and information-theoretic metrics to visually highlight common and uncommon issues and fixes that occur in the maintenance logs. Past works on text-based analysis display content and structure using features such as term frequencies, co-occurrences, and sentence structures. ConceptScope, [189], provides a conceptual overviews incorporating domain knowledge using Bubble Treemap visualization, multiple coordinated views of document structure, and concept hierarchy with text overviews.

Event-based Data: Nguyen et al. [125] used an interactive visual analysis tool that provides a high-level overview of CCTs (Calling Context Tree). CCTs help understand the execution and performance of parallel programs using performance metrics with call paths. The work uses semantic refinement operations to progressively explore performance bottlenecks on applications running on a multiple parallel processors. Kesavan et al. [87] developed EnsembleCallFlow to support the exploration of ensembles of call graphs, a combination of performance metrics and application execution contexts. They introduce ensemble-Sankey, combining resource-flow (a Sankey plot to describe the graphical nature of the call graph) and box-plot (to convey the performance variability within the ensemble) visualization techniques. Mueller et al. [119] studied the performance and behavior of cloud computing systems by building a visual analytics tool based on a layout method and similarity measures that visualized numerous behavioral lines, successfully identifying and suggesting performance bottlenecks

within the system.

2.3.2.2 Visualization in Streaming Data Analysis

Visualization of streaming data requires frequently updated results. Computational cost and cognitive load form major bottlenecks while visualizing results generated from incoming data streams. As for the cognitive load, Krstajic et al. [93] discussed the trade-offs between updating a view when a new data point is fed, which may lead to loss of mental map, or not updating a view, which may lead to loss of information. A comprehensive survey by Dasgupta et al. [28] further characterized challenges in perception and cognition of streaming visualizations.

Incremental algorithms aim at visualizing and updating results while avoiding rising computation costs. For example, Tanahashi et al. [163] built a streaming storyline visualization. Liu et al. [106] introduced a streaming tree cut algorithm to visualize incoming topics from text streams. Crnovrsanin et al. [22] developed a GPU accelerated incremental force-directed layout. Other researchers developed incremental dimensionality reduction [43] and change-point detection methods [88]. Katragadda et al. [86] developed a visual analytics tool for high-velocity streaming data using hardware and software sandbox environments.

While incremental algorithms allow for updating results within reasonable computation costs, these costs are sometimes unavoidable. To avoid larger computational costs in such scenarios, another potential approach for streaming data analysis is progressive visual analytics [101, 170]. Progressive algorithms provide a trade off between computational latency and result quality. Here, progressively refined intermediate results are visualized until the final results are computed. Thus, for streaming visualization, we can utilize progressive algorithms, such as the progressive version of t-SNE [133], UMAP [92], time series clustering [88], and MS plot [150]. The summary and evaluation results for the relevant works are provided in Fig. 2.3

2.4 Gaps and Future Work

Machine learning methods have gained considerable favor in log anomaly diagnosis and prediction in large-scale systems, and the trends show continuous growth. In this survey, we have isolated a subset of these studies in the area of cloud computing, infrastructure, visualization, and HPC systems. The methods employed in cloud computing usually account for online streaming analysis and hence focus on scalability. These methods and approaches could be adopted in the HPC and visualization fields to account for the rapid growth in the systems and workloads handled. Real-time online analysis methods that employ incremental and progressive updates [22, 100, 150, 158] to the already computed results are slowly gaining importance in the visualization field. However, more studies could adopt these methods. We feel these methods could significantly improve the lead time in the current analyses. Also, traditional analysis methods like signal processing, correlation, rule-based methods, etc., form a section of the analysis pipeline. Therefore, these studies should handle accounting for the errors propagated or useful data filtered through these sections in the pipeline.

Root-cause analysis is a tricky problem to solve as large-scale systems report errors from varied sources. In addition, there is a need to account for the age of these systems when handling data procured over multiple years. Current analysis pipelines do not account for unseen log events or sequences and instability of the log data from new error events that are caused due to system decline from fail-slow hardware [58]. There is a need to account for these performance issues, especially when studies use data over a larger number of years. This is not a trivial problem to solve since these incidents are harder to distinguish from those caused by software faults; hence more research in this field is required. The root-cause analysis could be made robust by procuring additional datasets that report the system's state, for example, sensor readings reporting temperature, power measurements, software, hardware, network logs, syslogs, etc. Currently, only a few studies consider a subset of these dataset types in their analysis [123,151,152]. Using these varied datasets could provide a more holistic understanding of the underlying system behavior and help in the identification of failure chains that are otherwise

missed.

2.5 Conclusion

As we approach exascale system capabilities, the issue of serviceability, availability, and resilience remains crucial for system design and maintenance. When the size of applications and the urgency in procuring results utilizing these ever-growing heterogeneous systems pose additional challenges, building technologies that can adapt to these transitions becomes crucial. We survey over 40 papers from cloud computing, infrastructure, visualization, and HPC systems and use their conclusions to provide a review in the field of log analysis. We have identified several areas in the current state of the practice from varied fields, which could help guide future research to effectively identify and build a more holistic prototype of the system's error propagation patterns.

Chapter 3
Functional Data Analysis: Real-Time Monitoring of Time-Series Data

3.1 Introduction

To ensure normal operation and adequate performance of hardware systems such as those in an assembly plant or a supercomputer center, various monitoring mechanisms have been introduced to collect data about all aspects of the systems at high frequency in real-time [128, 129]. The ability to instantaneously process and analyze the resulting time-series data thus becomes pertinent to examining various underlying system phenomena to detect and promptly react to system failures or inefficiency.

High-velocity time-series data is intrinsically functional as it can be represented in the form of curves or surfaces with weak assumptions of smoothness being permitted [174]. Functional data analysis (FDA) is a branch of statistics for analyzing such data [136]. FDA incorporates statistical methodologies to capture the underlying properties and structure of the data. The statistical methods and models in FDA are typically presented in the form of continuous functions [135]. A wide range of methods have been developed to handle functional data, such as functional principal component analysis (FPCA), functional regression models, and functional canonical correlation analysis [135]. As these names indicate, many FDA methods resemble those developed for conventional discrete analysis. However, because FDA handles data with continuous functions, FDA has advantages in studying the derivatives of the data and maintaining the temporal order of data [174]. Recently, FDA has seen tremendous growth with applications in various fields, such as biology, meteorology, medicine, finance, and engineering [3, 117, 175].

While FDA methods provide new capabilities for analyzing time-series data, they often suffer from their high computational costs, which increase with the number of time points. This can be a critical problem with real-world monitoring applications as they keep generating new time points, resulting in infinitely long time series.

With this work [150], we aim to reduce the computational overheads of using FDA on streaming time-series data while retaining the benefits of in-depth analysis capabilities provided by FDA. To do so, we introduce a visual analytics approach for continuously monitor and review time-series data that grows over time with a focus on identifying outliers by using FDA. In particular, we design new incremental and progressive algorithms that promptly generate the magnitude-shape (MS) plot [24], which reveals outlier time-series by depicting the outlyingness of both the functional magnitude and shape of multiple input time-series. These incremental and progressive updates to the previously computed results address the computational overhead introduced by handling growing time series data in bulk. In addition, with the support of FPCA, our approach help analysts investigate the visually-identified outliers from the MS plot.

Our main contributions are as follows:

- New incremental and progressive algorithms to generate the MS plot, which enables analyses of time-series data collected from online data streams.
- Augmentation of analysis using the MS plot with FPCA and interactive visualization to aid reviewing clusters identified from the MS plot.
- Four case studies with multiple real-world datasets demonstrate the effectiveness of our approach in two different settings: cases when (1) new time points and (2) new independent time-series are added, respectively.

3.2 Related Work

In this section, we first discuss relevant works in streaming data visualization and then provide a background to FDA, including functional principal component analysis (FPCA) and magnitude-shape (MS) plots [24]—FDA methods utilized in our work.

3.2.1 Streaming Data Visualization

Visualization of streaming data is a challenging task since visualizations need to be continually updated with incoming data. The major bottlenecks in visualizing continuous streams of data are computational cost and cognitive load. Within these bottlenecks, the cognitive load is discussed by Krstajic and Keim [93]. For example, they summarized the trade-offs between updating a view when a new data point is fed from the stream, which leads to loss of mental map, or not updating a view, which leads to loss of information. Dasgupta et al. [28], through their comprehensive survey, further characterized challenges in perception and cognition of streaming visualizations and methods developed to address the challenges. In this work, while we consider the cognitive load of visualizations, we mainly address the computational cost when producing visualizations.

Here we discuss some of the past works on incremental updates to visualize up-to-date results while avoiding rising computation costs. For example, Tanahashi et al. [163] built a storyline visualization for streaming data by utilizing the previous steps' storylines when deciding the new data points' layout. Liu et al. [106] introduced a streaming tree cut algorithm to instantly detect and visualize incoming topics from text stream analysis. Crnovrsanin et al. [22] developed an incremental force-directed layout algorithm with GPU acceleration. Several works also developed and enhanced incremental methods for visual analysis of high-dimensional and time-series data, including dimensionality reduction [43,124] and change-point detection methods [88]. Katragadda et al. [86] developed VAStream, a visual analytics system that can handle high-velocity streaming data by using hardware and software sandbox environment optimized to streaming data analysis workflow.

Another potential approach for streaming data analysis is the use of progressive visual analytics [170]. Progressive visual analytics provides reasonable intermediate results within a required latency when the computational cost for an entire calculation is too high. This latency requirement is common with streaming data visualizations. Therefore, several researchers developed progressive algorithms for streaming visual-

ization, such as the progressive version of t-SNE [133] and time-series clustering [88].

Compared with the existing works above, our work addresses the problem when using an FDA method to generate visualizations from data streams. Our algorithm demonstrates a new capability for visual analytics of functional data, including time-series data, in a streaming, timely manner.

3.2.2 Functional Data Analysis

Functional data is data procured from continuous phenomena of space or time and represented in the form of smooth functions. For example, sensor readings at components in a supercomputer can be viewed as data produced from a function $\mathbf{X}$ of K variables (e.g., variables corresponding to temperature and voltage). Then, $\mathbf{X}_n(t)$ is an observation (e.g., temperature) corresponding to a machine component n (e.g., one CPU temperature sensor) at time t. While readings are collected at finite resolutions in practice, the temperature, for example, continuously exists over time. Thus, it is natural to model and analyze $\mathbf{X}_n(t)$ with a continuous function defined over time from the observations.

Over the past decades, the field of functional data analysis has seen various advances in functional data modeling, clustering, differential analysis, and outlier detection. The functional non-parametric statistics with free-modeling ideas were popularized by Ferraty and Vieu [38]. Ramsay and Silverman [137] applied parametric statistics, such as linear regression, principal components analysis, linear modeling, and canonical correlation analysis, to the functional domain. Horvath and Kokoszka [72] developed statistical methods and theory for inference (e.g., two-sample inference, change point analysis, and tests for dependence) of independent and identically distributed functional data as well as dependent functional data structures.

Besides various FDA methods, functional principal component analysis (FPCA) is one of the most classic and popular methods. The fundamental concept of FPCA includes capturing the principal directions of modes of variations along with dimensionality reduction. Using the basis spanned by the principal component, FPCA is able to summarize subject-specific features. Karhunen [85] and Loeve [107] made the first

advancements in FPCA through theories on optimal data expansion of a continuous stochastic process. Building on this work, Rao [138] worked on statistical tests for the comparison of temporal growth curves. Following works on FPCA investigated its properties and improved on methodology and convergence rates [63] Through continued systematic research on FPCA, several extensions and modifications of FPCA have been popularized, as listed in the survey by Shang [149]. Some of the practical applications of FPCA include functional magnetic resonance imaging (fMRI) [173], age-specific mortality rates [74], and analysis of income density curves [91].

Since functional data inherently consists of smooth, continuous sequences being collected from various sources, outlier detection is often performed as a preliminary step in analysis to distinguish sequences that follow different functions from the others. The depth of a point shows how deep a point is relative to a given data set by measuring its centrality with respect to the other points forming the data cloud. Statistic depth is a widely used non-parametric inference method (methods that make no assumptions about the probability distributions of the variables being assessed) for exploratory analysis of functional data. The concept of data depth was first proposed by Tukey [168] for visualizing bivariate data. Some of the popular depth measures include half-space depth [168], projection-depth [194], and spatial depth [172].

Several visualization tools have been developed for FDA, which serve as an effective way to communicate underlying characteristics otherwise not apparent through summary statistics and models. Hyndman and Shang [75] introduced tools to visualize large amounts of functional data, such as functional versions of bagplots (multidimensional boxplots)and highest-density-region plots. Some other popular tools include functional boxplots [168], and magnitude-shape (MS) plots [24]. The MS plot is designed to be used for visual identification of outliers by depicting the magnitude and shape outlyingness of each series of functional data. With simulated and real-world examples, the authors have shown how the MS plot is superior in identifying potential outliers. We enhance the MS plot for streaming data analysis by providing incremental and progressive algorithms to enable timely visualization updates.

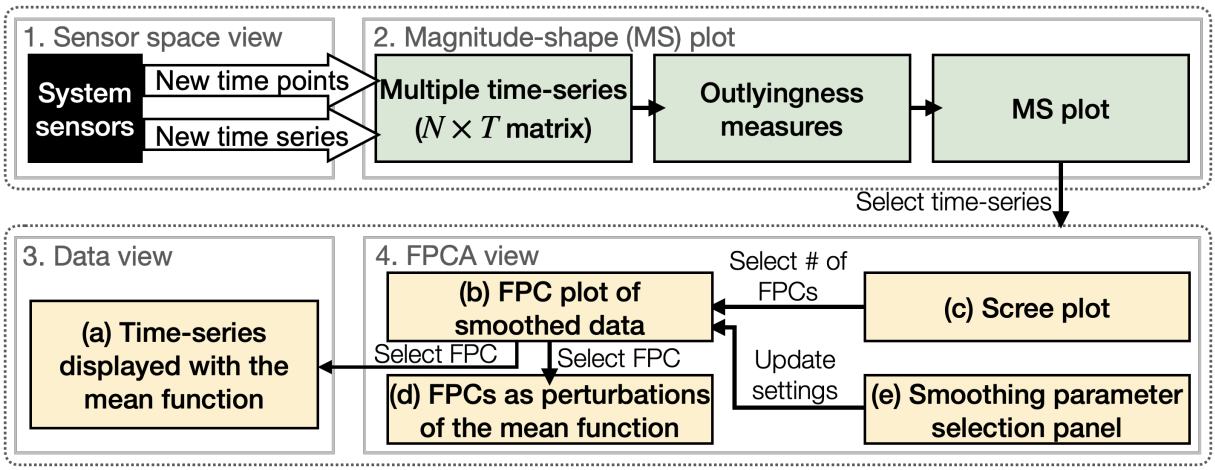

Figure 3.1: General architecture of our visualization tool (© 2022 IEEE).

3.3 Methodology

This section describes the architecture of our visual analytics tool and back-end analysis methods.

3.3.1 Overall Organization

Fig. 3.1 shows the organization of our visual analytics tool. The tool can be divided into two main components: (1) generation of the MS plot from streaming feed (top) and (2) interactive analysis over the generated MS plot with the auxiliary visualizations from other FDA methods, including FPCA and smoothing functions (bottom).

To make explanations concise and concrete, we describe our algorithms with streaming (multivariate) time series data as an example of functional data. While such data is our main analysis target, our approach can be applied to other types of functional data as we use general FDA methods such as the MS plot and FPCA. Here we denote the numbers of existing time points, variables (e.g., measured temperature and voltage), and time series in the current multivariate time series data with T, K, and N, respectively.

Time series data is collected from various sensors housed at various levels of the hierarchy within a system. If the sensor system topology is available and can be visualized, we have an optional sensor space view (Fig. 3.1-1) where we display the monitoring system components and highlight the individual components from which

the measurements/readings are being processed. In Fig. 3.1-2, we first initialize the magnitude and shape outlyingness measures from the current set of functional data. Then, as shown in Fig. 3.1-1, data updates from the stream can be the addition of either time point (i.e., $T \rightarrow T+1$) or time series ($N \rightarrow N+1$).

Based on the updates, we compute the directional outlyingness measures [24] incrementally without any approximations for the addition of a time point and progressively with approximations for the addition of a time series (Fig. 3.1-2). The progressive update introduces errors when compared to the actual results. If the errors exceed a predefined threshold, the results are recomputed for all the selected time series in the back-end and made available in the UI upon completion. We make sure that incremental and progressive updates do not co-occur to avoid overwriting results. The MS plot is updated accordingly. An example of the MS plot can be seen in Fig. 3.2-b. By default, we update the MS plot every 10 addition of time points (e.g., 5 seconds when a stream rate is every 0.5 second) to help analysts maintain their mental map while avoiding large information loss.

In the MS plot, each *circle* represents one time series. The user can select multiple time series to visualize them in the data view (Fig. 3.1-3). The selected time series are also processed by the FPCA pipeline (Fig. 3.1-4), and the results are shown in the functional principal components (FPCs) plot, the scree plot [16], and the plot showing FPCs as the perturbation to the mean function. In addition, the UI includes a panel for the user to select smoothing basis functions for applying FPCA. Sec. 3.3.3 describes the detail of each of these plots.

3.3.2 Incremental & Progressive Generation of the Magnitude-Shape Plot

We describe our algorithms[1] and system implementations that are designed to generate the MS plot from data streams.

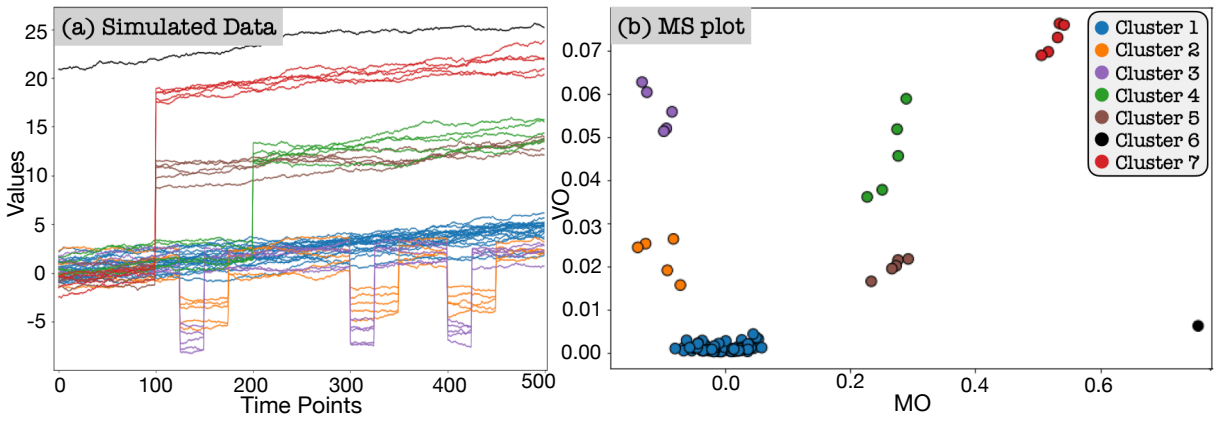

Figure 3.2: Visual outlier detection with the MS plot: (a) the simulated functional data and (b) the MS plot with magnitude outlyingness (MO) and variational outlyingness (VO) along x-, and y-axes, respectively (© 2022 IEEE).

3.3.2.1 Magnitude-Shape (MS) Plot

We briefly introduce the MS plot [24], which we use as a primary visual analysis tool, and extend it to use in a streaming setting. The MS plot is a scatterplot that shows, for each time series, the corresponding outlyingness measures: mean directional outlyingness and variation of directional outlyingness [23], for example, as x- and y-coordinates, respectively. As shown in Fig. 3.2, the MS plot depicts how much a time series has a different magnitude and shape with the other time series, and thus visually reveals outliers.

A time series can be considered an outlier if it behaves in a manner inconsistent with the other time series. One of the measures of the outlyingness of time series is the Stahel-Donoho outlyingness [194]. This measure is suited when values in time series data roughly follow elliptical distributions; however, it cannot capture the outlyingness well when they have skewed distributions. To address this limitation, Dai and Genton [24] introduced the directional outlyingness, which they also utilize in the MS plot.

The directional outlyingness is computed by splitting the data into halves around the median and using the robust scale estimator to handle any skewness. The directional

[1]The related source code is available at `https://github.com/sshilpika/streaming-ms-plot`.

outlyingness **O** is defined by:

$$\mathbf{O}\left(\mathbf{X}(t), F_{\mathbf{X}(t)}\right) = \left\{\frac{1}{d\left(\mathbf{X}(t), F_{\mathbf{X}(t)}\right)} - 1\right\} \cdot \mathbf{v}(t) \tag{3.1}$$

where $\mathbf{X}$ is a K-dimensional function defined on a time domain $\mathcal{T}$, $F_{\mathbf{X}(t)}$ is a distribution of a random variable $\mathbf{X}(t)$, d ($d > 0$) is a depth function which decides ranks of functional observations from most outlying to most typical, and $\mathbf{v}(t)$ is the unit vector pointing from the median of $F_{\mathbf{X}(t)}$ to $\mathbf{X}(t)$. With $\mathbf{Z}(t)$ as the unique median of $F_{\mathbf{X}(t)}$, $\mathbf{v}(t) = (\mathbf{X}(t) - \mathbf{Z}(t))/\|\mathbf{X}(t) - \mathbf{Z}(t)\|_2$ ($\|\cdot\|_2$ denotes the L^2-norm). Dai and Genton [24] use the projection depth as a default depth function d. Note that our algorithms also employ the projection depth to follow their default. The projection depth (PD) is defined as:

$$\mathrm{PD}\left(\mathbf{X}(t), F_{\mathbf{X}(t)}\right) = \frac{1}{1 + \mathrm{SDO}\left(\mathbf{X}(t), F_{\mathbf{X}(t)}\right)} \tag{3.2}$$

$$\mathrm{SDO}\left(\mathbf{X}(t), F_{\mathbf{X}(t)}\right) = \sup_{\|\mathbf{u}\|=1} \frac{\|\mathbf{u}^\top \mathbf{X}(t) - \mathrm{median}(\mathbf{u}^\top \mathbf{X}(t))\|_2}{\mathrm{MAD}\left(\mathbf{u}^\top \mathbf{X}(t)\right)} \tag{3.3}$$

where SDO is the Stahel-Donoho outlyingness [194] and MAD is the median absolute deviation. The three measures of directional outlyingness are defined as:

· Mean directional outlyingness (**MO**):

$$\mathbf{MO}\left(\mathbf{X}, F_{\mathbf{X}}\right) = \int_{\mathcal{T}} \mathbf{O}\left(\mathbf{X}(t), F_{\mathbf{X}}(t)\right) w(t) dt \tag{3.4}$$

· Variation of directional outlyingness (VO):

$$\mathrm{VO}(\mathbf{X}, F_{\mathbf{X}}) = \int_{\mathcal{T}} \|\mathbf{O}(\mathbf{X}(t), F_{\mathbf{X}}(t)) - \mathbf{MO}(\mathbf{X}, F_{\mathbf{X}})\|_2^2 w(t) dt \tag{3.5}$$

· Functional directional outlyingness (FO):

$$\begin{aligned} \mathrm{FO}\left(\mathbf{X}, F_{\mathbf{X}}\right) &= \int_{\mathcal{T}} \|\mathbf{O}\left(\mathbf{X}(t), F_{\mathbf{X}}(t)\right)\|_2^2 w(t) dt \\ &= \|\mathbf{MO}\left(\mathbf{X}, F_{\mathbf{X}}\right)\|_2^2 + \mathrm{VO}\left(\mathbf{X}, F_{\mathbf{X}}\right) \end{aligned} \tag{3.6}$$

where $w(t)$ is a weight function defined on $\mathcal{T}$. Dai and Genton [24] use a constant weight function, i.e., $w(t) = \{\lambda(\mathcal{T})\}^{-1}$ where $\lambda(\cdot)$ represents the Lebesgue measure. Note that **MO** is simply the mean of the directional outlyingness for each of the K dimensions;

thus, **MO** is a K-dimensional vector. On the other hand, VO and FO involve the computation of the L^2-norm, resulting in scalar values.

As shown in Fig. 3.2, we can visually identify the functional outliers with the MS plot. In this example, each time series (e.g., measured temperature) is from one univariate function; thus, MO is a scalar. MO and VO show the outlyingness in magnitude and shape. Thus, the central time series (i.e., time series similar to the other majority of time series) are mapped the region with small |MO| and small VO (e.g., Cluster 1 in Fig. 3.2). On the other hand, outliers that take different magnitudes from the others across time are located in the region with large |MO| and small VO (e.g., Cluster 6). Similarly, outliers that have a different curve shape from the others are placed in the region with small |MO| and large VO (e.g., Cluster 3). Time series with large |MO| and large VO (e.g., Cluster 7) are the curves with greatly outlying in both magnitude and shape. For the case where **MO**'s dimension is higher than 2 (i.e., $K > 2$), we can visualize the information of **MO** and VO with high-dimensional visualization methods, such as parallel coordinates, as Dai and Genton suggested [24]. In the following, we describe our algorithm for the case where $K = 1$ as this can be considered the main visual analysis target with the MS plot. Note that when $K = 1$, $\mathrm{SDO}(\mathrm{X}(t), F_{\mathbf{X}(t)}) = |\mathrm{X}(t) - \mathrm{Z}(t)| \,/\, \mathrm{median}(|\mathrm{X}(t) - \mathrm{Z}(t)|)$.

3.3.2.2 Incremental Updates of the MS Plot along Time

Even though the MS plot is generated with MO and VO defined with a function, in practice, MO and VO need to be computed with a finite but large number of time points. As the number of time points increases, more computations are required. Consequently, when newly measured time points are continually fed from the data stream, the MS plot generation gradually becomes infeasible in real-time. To solve this issue, we derive equations that enable incremental updates of MO and VO. Our incremental updates provide *exact* MO and VO without any approximation.

Using the projection depth as a depth function, the discretized versions of Eq. 3.1,

3.4, and 3.6 are:

$$\mathrm{O}(\mathrm{X}^T[t]) = \mathrm{SDO}(\mathrm{X}^T[t]) \cdot \mathrm{v}^T[t] = \frac{\mathrm{X}^T[t] - \mathrm{Z}^T[t]}{\mathrm{median}(|\mathrm{X}^T[t] - \mathrm{Z}^T[t]|)} \tag{3.7}$$

$$\mathrm{MO}^T(\mathrm{X}^T) = \sum_{t=1}^{T} \mathrm{O}(\mathrm{X}^T[t])\, w^T[t], \tag{3.8}$$

$$\mathrm{FO}^T(\mathrm{X}^T) = \sum_{t=1}^{T} \mathrm{O}(\mathrm{X}^T[t])^2 w^T[t] \tag{3.9}$$

where T is the number of time points available so far and superscript T represents that each measure is defined on a time range $[1, T]$. Practically, $\mathrm{Z}^T[t]$ is estimated from measured values; thus, we can assume $\mathrm{Z}^T[t]$ is simply the median of N time series $\{\mathrm{X}_1^T, \cdots, \mathrm{X}_N^T\}$ at time point t and median$(|\mathrm{X}^T[t] - \mathrm{Z}^T[t]|)$ is the median of $\{|\mathrm{X}_1^T[t] - \mathrm{Z}^T[t]|, \cdots, |\mathrm{X}_N^T[t] - \mathrm{Z}^T[t]|\}$. Also, here we assume T time points have an approximately constant time interval. To follow the default weight function by Dai and Genton [24], we also use a constant weight function as w^T. In the discretized case, $w^T[t] = 1/T$.

When adding a new time point at $T+1$, Eq. 3.8 and Eq. 3.9 become:

$$\begin{aligned} \mathrm{MO}^{T+1}(\mathrm{X}^{T+1}) &= \sum_{t=1}^{T+1} \mathrm{O}(\mathrm{X}^{T+1}[t]) w^{T+1}[t] \\ &= \frac{1}{T+1}\Big(T\mathrm{MO}^T(\mathrm{X}^T) + \mathrm{O}(\mathrm{X}^{T+1}[T+1])\Big), \end{aligned} \tag{3.10}$$

$$\begin{aligned} \mathrm{FO}^{T+1}(\mathrm{X}^{T+1}) &= \sum_{t=1}^{T+1} \mathrm{O}(\mathrm{X}^{T+1}[t])^2 w^{T+1}[t] \\ &= \frac{1}{T+1}\Big(T\mathrm{FO}^T(\mathrm{X}^T) + \mathrm{O}(\mathrm{X}^{T+1}[T+1])^2\Big). \end{aligned} \tag{3.11}$$

Then, because of Eq. 3.6, $\mathrm{VO}^{T+1} = \mathrm{FO}^{T+1} - (\mathrm{MO}^{T+1})^2$.

When computing MO^{T+1} and FO^{T+1}, we have already calculated MO^T and FO^T. Thus, to obtain MO^{T+1} and FO^{T+1} for all N time series, we need to only calculate the directional outlyingness O for the newly added time point. This process has time complexity $\mathcal{O}(N)$ when using the PD as a depth function. Also, the required memory space to save the previous results, MO and FO, for all N time series is $\mathcal{O}(N)$. Therefore, we can update the exact values of the three measures of directional outlyingness with

small time and space complexities that do not relate to the increasing number of time points, T. Note that while the equations above are for the incremental addition, as seen in Eq. 3.10 and Eq. 3.11, the incremental deletion is also supported, which can be derived with minor adjustments of signs, etc.

3.3.2.3 Progressive Updates for New Time Series

When the number of measured time series, N, in a system is large, the overhead of computing the directional outlyingness measures can be large. The original MS plot requires recomputation when new time series are added (e.g., adding temperatures obtained from different compute racks). To provide useful intermediate results or enable the incremental addition of time series, we design a progressive algorithm that generates the MS plot with estimated directional outlyingness measures. We also provide a refinement mechanism to maintain the MS plot quality.

Unlike incremental updates along time (Sec. 3.3.2.2), when adding new time series, we cannot obtain exact solutions while keeping the time complexity constant in terms of an increase in the number of time series. This is because all computations are related to Z^T (the median of N values of X^T); thus, the update of Z^T based on a new time series requires recomputation of the measures for all time series at all time points. Therefore, we (1) incrementally update the results by assuming $Z^{T,N+1} \approx Z^{T,N}$ (superscript N shows the corresponding measure is defined on N time series) as long as the errors are within the predefined error threshold and (2) progressively update the results with $Z^{T,N+1}$ if the predefined error threshold is crossed. For a newly added time series, with the assumption of $Z^{T,N+1} \approx Z^{T,N}$, we can easily compute O, MO^T, and FO^T with Eq. 3.7–3.9. Also, $VO^T = FO^T - (MO^T)^2$. The condition $Z^{T,N+1} \approx Z^{T,N}$ is checked using the Kullback–Leibler (KL) divergence of the mean absolute deviation between the new and original time series. Since KL divergence values can range anywhere between 0–infinity, we have set the error threshold 10 by default, and this can be varied by the user. We found that an error threshold of 10 provided reasonable results without much information loss through data processing. However, the automatic identification of the error threshold is not within the scope of this work.

When adding a new time series, there is a good possibility that the time series closely follows the shape and magnitudes represented by the existing function. By avoiding updating Z^T for each addition of a time series, we significantly reduce the computational overhead. However, as stated above, once the difference between $Z^{T,N}$ and $Z^{T,N+1}$ becomes larger than the threshold, our algorithm starts to recompute the measures for each time series one by one. Similar to the incremental update in Sec. 3.3.2.2, the deletion of time series is also supported with minor changes to the above equations.

3.3.2.4 Implementation with Visual and Computational Considerations on Update Frequency

We implement our tool with the design recommendations for progressive visual analytics systems by Turkay et al. [169]. In case of receiving a new time point, we update the outlyingness measures at each time point in the back-end. However, the MS plot is not updated at every addition of a time point; instead, we update it at a predefined number of arrived time points (10 by default). When receiving a new time series, we update both the outlying measures and the MS plot. This procedure adds a new circle to the MS plot. When $Z^{T,N+1} \not\approx Z^{T,N}$, the outlyingness measures no longer give reasonable results. Hence, the previously computed measures need to be updated with $Z^{T,N+1}$. This update can be computationally expensive to do on the fly if the dataset size is large. Thus, we compute the results asynchronously in the back-end, and the front-end visualization is updated on completion. All visual updates are performed with animated transitions for easy interpretation of changes.

3.3.3 Reviewing the MS Plot with Auxiliary Information and Functional Principal Component Analysis

Fig. 3.3 shows our tool's UI, consisting of the (a) MS plot, (b) sensor space, (c) data, and (d) FPCA views. The analysis starts from the selection in the sensor space view, which shows the spatial information related to the data, if available. For example, in Fig. 3.3, we analyze multiple temperatures measured at each compute rack in a supercomputer—390 temperature readings per rack. In Fig. 3.3-b, we visualize the locations of the racks together with color encoding the number of outlier readings.

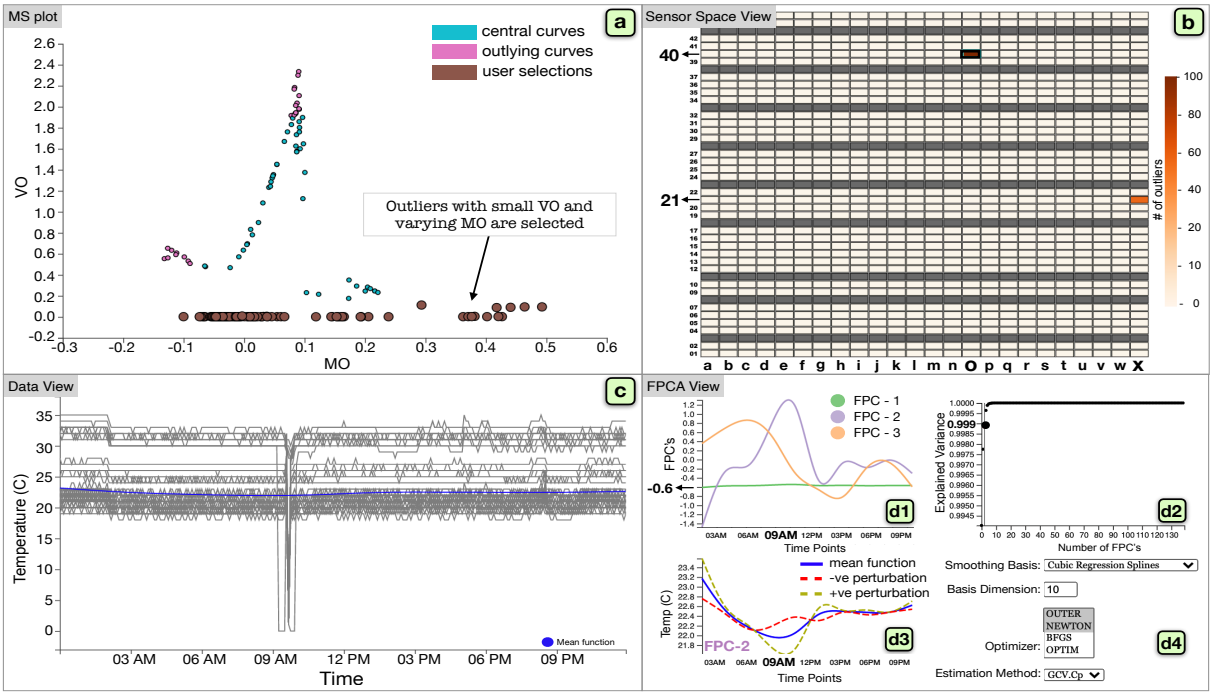

Figure 3.3: The UI of our visual analytics tool. (a) The MS plot shows MO and VO as a scatterplot, which is being updated incrementally and progressively. (b) The sensor space view provides the related spatial information if available (here shows the compute rack information). (c) The data view depicts the time series selected in the MS plot. (d) The FPCA view displays the FPCA results and settings, which include: the (d1) FPC plot, (d2) scree plot, (d3) FPC as a perturbation of the mean plot, and (d4) functional smoothing parameter selection panel (© 2022 IEEE).

With mouse actions (clicking or lasso selection), the analyst can select items of interest (i.e., racks in Fig. 3.3-b), and then the UI shows the related points in the MS plot. An example in Fig. 3.3-a shows the result of selecting the dark orange cell (i.e., the rack contains many outliers) around the top center (`o40`) of Fig. 3.3-b. Here the outlier counts are the results previously computed from the MS plot for each component.

The MS plot (Fig. 3.3-a) is generated with the algorithms described in Sec. 3.3.2. We color circles by their membership in either central or outlying curves (teal: central, pink: outlying). For clarity, we define the central curves as curves with MO within 25-75% of the value range and VO below 75% of the value range. This adjustable range is selected while considering the normal operation region for sensor measurements, such as voltage and temperature. Analysts can interactively select interesting areas (bigger brown circles) to examine more details in other views (Fig. 3.3-c, d).

Fig. 3.3-c visualizes the selections as line charts. As a typical central curve, a blue line shows their mean function—a smoothed line around their mean at each time point. These visualized lines can be further analyzed with the FPCA view (Fig. 3.3-d). To apply FPCA [174] on the lines shown in Fig. 3.3-c, we first need to smooth the lines. Fig. 3.3-d4 allows analysts to select the smoothing basis functions, the number of basis, the method for optimizing the smoothing parameter, and the smoothing parameter estimation method [69]. We show the default settings in Fig. 3.3-d4.

FPCA is then performed on the smoothed lines, and FPCs are generated, as shown in Fig. 3.3-d1. Analogous to PCA, FPCs preserve the variance of functions as much as possible by defining a weight for each time point in a continuous curve. From the shape of each FPC, we can identify the time range that has a strong influence on each FPC. For example, in Fig. 3.3-d1, the first FPC (FPC-1) has the same weight around -0.6 across time, while the second FPC (FPC-2) has a large weight at around 9 AM. Because most of the lines are relatively flat in Fig. 3.3-c, FPC-1 seems to preserve the major variance related to all the time points; FPC-2 seems to preserve the variance related to the distinct drop around 9 AM in Fig. 3.3-c. This can be confirmed by our time series selection method that relates to each FPC. When analysts select one FPC from Fig. 3.3-d1, we compute the FPC score [174] for each time series, which represents how strongly the time series is related to each FPC. Then, the tool selects and highlights the top-k ($k = 10$ by default) time series that highly relate to the selected FPC. The corresponding circles are also highlighted in Fig. 3.3-a. Therefore, FPCA identifies the influential time and categorizes time series using FPCs. While the top-3 FPCs are shown in the FPC view by default, analysts can use the scree plot (Fig. 3.3-d2) to select the number of FPCs to be shown. Similar to PCA, it shows the number of dimensions (FPCs) and the cumulative explained variance ratio. From this, analysts can judge how many FPCs are required to capture the data variance reasonably.

In the FPC as a perturbation of the mean plot (Fig. 3.3-d3), we plot the mean function of the selected data (the same one in Fig. 3.3-c but with a different y-range) and the functions obtained by adding and subtracting a suitable multiple, $\sqrt{2\xi_i}$ [137], of an

FPC chosen from the FPC plot, `FPC-i` ($i = \{1,2,3,\ldots\}$), where ξ_i is the eigenvalue corresponding to `FPC-i`. The obtained functions are shown as positive and negative perturbations. This explains the fluctuation of the measured data (i.e., y values in Fig. 3.3-d3) the `FPC-i` captures. For example, in Fig. 3.3-d3, where FPC-2 is selected, the positive perturbation has a clear drop from the mean function around the 9 AM followed by fluctuations around the mean function, which we cannot discern in the data view. This indicates that time series highly related to FPC-2 are mainly characterized by this variation around 9 AM, which can be confirmed with Fig. 3.3-c. By showing only the selected time series and their FPCA results, we can reduce the visual clutter and computational costs of FPCA.

To summarize, from Fig. 3.3-b, we find two racks (`o40` and `x21`) that behave abnormally and select `o40` since it includes many outliers. Then, from the MS plot, we select time series with low VO but varying MO. Indeed, from Fig. 3.3-c, we can see that while most time series follow the mean behavior, some show a significant drop in the temperature soon after 9 AM. Applying FPCA to the selected time series extracts FPCs and the modes of variation, revealing that FPC-2 captures the variation related to the drop (Fig. 3.3-d1). Indeed this variation was linked to a system board failure inside (`o40`). Although this was a minor variation than the other variations in the dataset (large VO in Fig. 3-b), using FPC helped reveal this relatively smaller anomaly. The perturbation plot of FPC-2 (Fig. 3.3-d3) captures readings that flip across the mean close to 9 AM in the duration of the failure. We provide further analysis of this data in Sec. 3.4.3.

3.4 Case Studies

We have shown an analysis example in Sec. 3.3.3. We further demonstrate the effectiveness of our tool with four case studies, including analyses of the Canadian weather, ozone level, supercomputer's hardware log, and Biometrics datasets. Each dataset is preprocessed to handle missing values and to extract relevant information for the analysis.

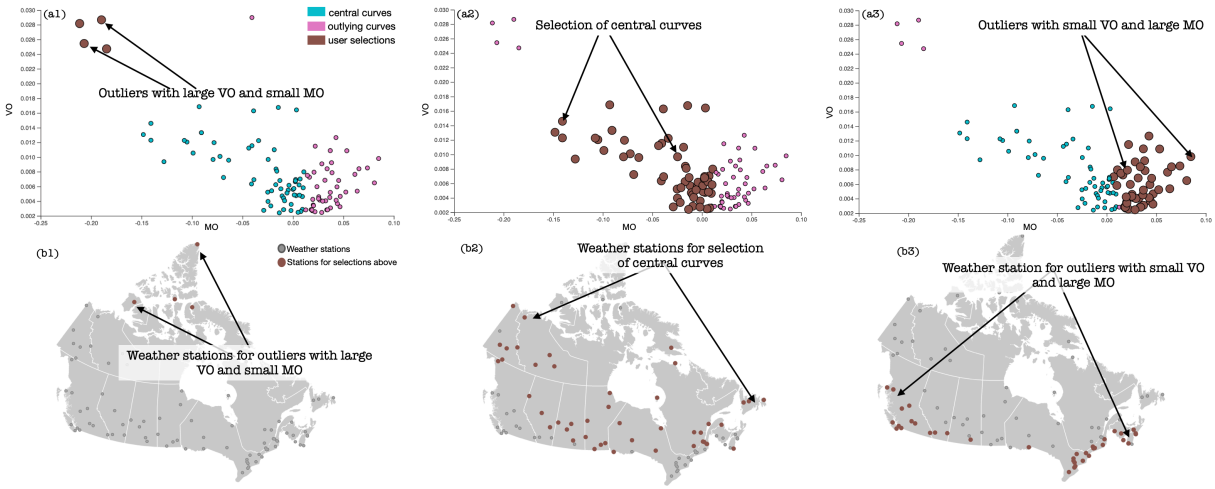

Figure 3.4: The characterization of Canadian Weather with the MS plot. (a1, a2, a3) show the different selections of the weather stations performed in the MS plot. (b1, b2, b3) show the corresponding weather stations.

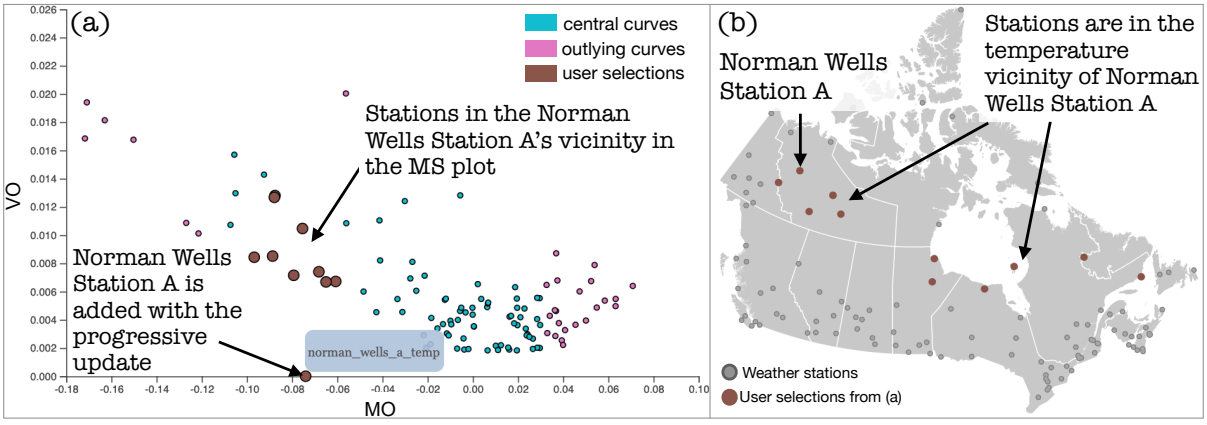

Figure 3.5: The results after the progressive update in the MS plot, which adds a series of temperature readings at a new station, Norman Wells Station A. From the MS plot (a), we select the new station and the stations placed close to the new station in the MS plot. (b) shows the corresponding stations in brown.

3.4.1 Study 1: Analysis of Canadian Weather Data

We analyze the historical climate data in Canada [56]. We filter the data by province or territory, weather stations, and the time period. We choose 118 stations for the years 2019 and 2020. The dataset consists of daily measurements of temperature, precipitation, wind speed, rain days, etc., along with minimum, maximum and average values. Here we show how the MS plot characterizes the readings, specifically temperature measurements. We then illustrate the results from the incremental and progressive

updates on the MS plot.

As shown in Fig. 3.4-a1, after updates in a period of time, the MS plot starts to reveal several outliers. We select two distinct groups of outliers and central curves in Fig. 3.4-a1, a2, a3. Fig. 3.4-b1, b2, b3 show the corresponding stations visualized in the space view. In Fig. 3.4-a1, we select outliers that have large VO and small MO. Fig. 3.4-b1 shows these stations (Arctic Bay, Resolute, Thomson River, and Alert) lie in the far north region of Canada, each of which has bitterly cold temperatures during the winter and high temperatures of up to $20°C$ during the summer. On the other hand, from Fig. 3.4-a2, b2, the stations that have the central curves are located in the central regions of Canada. At these locations, the summer is usually warm with temperatures range from $15°C$ in May to the mid-$30°C$ in July and August while the winter normally begins in November and temperatures generally remain below the freezing point. In Fig. 3.4-a3, b3, we select measurements with high magnitude and low variational outlyingness. The low variational outlyingness tells us that the weather in these stations do not show large fluctuations when compared to the central curves. The corresponding stations are in the southwest and southeast regions of Canada, which are characterized by temperature typically varying from $-7.2°C$ to $28.3°C$ and is rarely below $-16.6°C$ or above $33.8°C$. For the above observations, we can see that the MS plot characterize the temperatures at the different stations well, and successfully identify the outliers, especially the stations selected in Fig. 3.4-a1, where the temperature has excessively low magnitudes and high variations.

We illustrate an analysis with the progressive update in Fig. 3.5. As described in Sec. 3.3.2.3, the progressive update is performed with the assumption in the medians—$Z^{T,N+1} \approx Z^{T,N}$—as long as the difference falls within the error threshold. With this assumption, the MS plot produces a sufficient result while avoiding significantly high computational cost. In Fig. 3.5-a, under the assumption above, the MS plot places the newly added temperatures measured at Norman Wells Station A. The new point in the MS plot is at the lower central region slightly away from the rest of the points. We select the new point and additional points close to it to see if the placement of the

new station makes sense. From Fig. 3.5-b, all selected stations, shown in brown colors, lie in the spatial and temperature vicinity of the newly added feature corresponding to Norman Wells Station A. Therefore, we see that although the newly added station introduces some errors, these errors are not significant. Also, if these errors cross the error threshold, our back-end mechanism asynchronously updates the results with the exact value of $Z^{T,N+1}$ for the entire dataset.

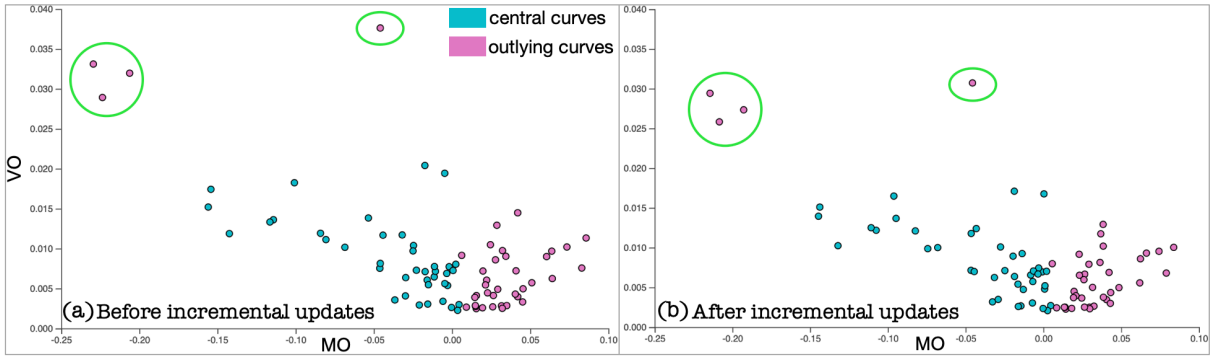

Figure 3.6: The MS plot (a) before and (b) after the incremental updates. From (a) generated with temperatures for 300 days, 20 additional days are added in (b). Both (a) and (b) have similar distributions of point positions. However, each point's VO tends to be smaller in (b). This is more salient in the high VO outliers indicated with the green circles.

We also demonstrate an analysis with the incremental update in Fig. 3.6. Fig. 3.6-a shows the MS plot generated with the temperatures measured at the first 280 days at 100 stations. Then, we keep observing the MS plot until the temperatures in additional 20 days in October are added (Fig. 3.6-b). Note that the incremental updates of time points does not cause additional errors unlike the progressive updates. From Fig. 3.6-a, b, we see that the distribution of points in the MS plot is not changed much before and after the updates. This is expected since the temperatures did not widely fluctuate within these 20 days. However, at the same time, we can see that each point's VO becomes slightly smaller in Fig. 3.6-b. Especially, for the outliers with high VO—highlighted with green circles, this tendency can be seen more clearly. We can consider that this smaller change is caused by stable temperatures in October across Canada.

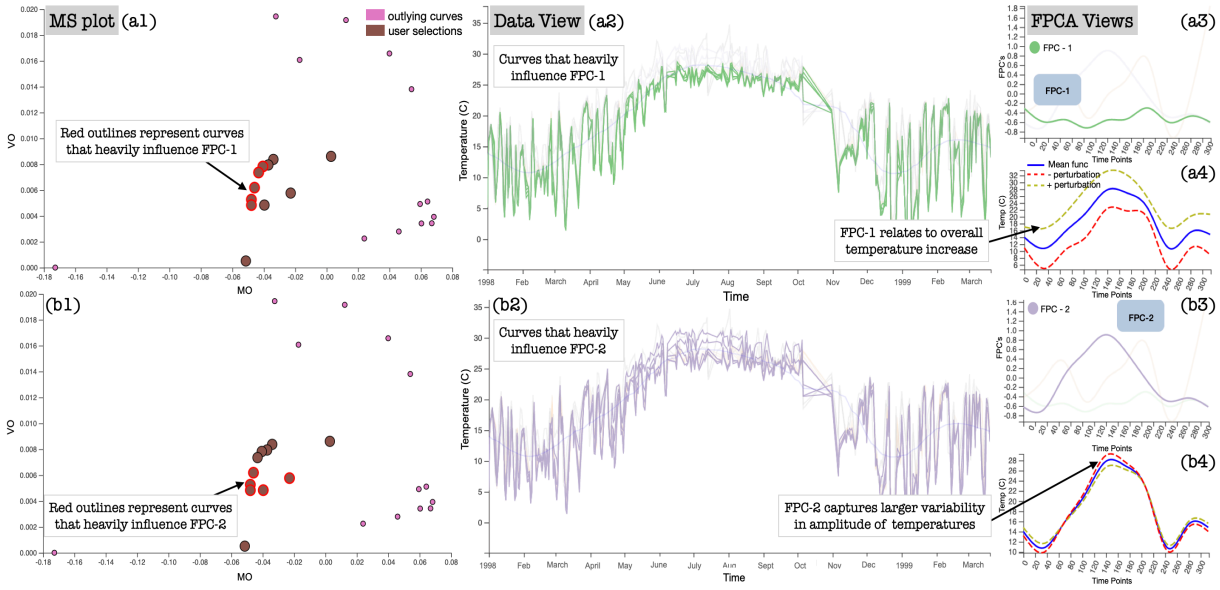

Figure 3.7: The in-depth analysis of central curves identified by the MS plot. The figure shows (1) MS plot is displayed on the left, (2) the Ozone temperature curves in the middle, and (3) the FPC analysis (consisting of FPC's and FPC as perturbations of mean) plots in the right.

3.4.2 Study 2: Analysis of Ozone Level Data

Ozone levels in the air fluctuates due to some related parameters (e.g., temperatures) which is often the case with a stochastic process. When the fluctuations cross a certain threshold, it results in adverse effects on human health. Hence, it is crucial to accurately monitor these parameters and issue a warning of a dangerous level. The ozone dataset contains 75 features which include temperature, wind speeds, ozone background level, and relative humidity each measured at multiple geo-potential heights at different times of the day. Here, we analyze temperatures measures at various heights on the ozone layer. The results generated with our tool is shown in Fig. 3.7.

In this case study, we demonstrate the usefulness of the analysis with FPCA to supplement or improve the MS plot. While the MS plot in Fig. 3.7-a1 reveals clear outliers, here we analyze the central curves in detail. We first select the central curves from the MS plot, as shown with brown circles in Fig. 3.7-a1. With the help of FPCA, we can categorize subgroups of these central curves and see how these individual subgroups vary from the others. Fig. 3.7-a3 shows the FPCA result of the selected central curves. While the FPCA result originally depicts the top-3 FPCs, we individually

select the first and second FPCs (FPC-1 and FPC-2). Then, the related information is highlighted in all views.

We first select FPC-1 from the FPC plot, as shown in Fig. 3.7-a3; then related central curves are highlighted with the red outlines in a1 and emphasized with the corresponding FPC color in a2. In Fig. 3.7-a4, the mean function is visualized with its positive and negative perturbations. From Fig. 3.7-a2, we see that all curves related to FPC-1 have the same fluctuation patterns (i.e., most of them are heavily overlapped at every time point). In Fig. 3.7-a4, by looking at the difference between the mean function and its positive or negative perturbation, we can see that the effect from FPC-1 is merely to add and subtract a constant throughout the entire duration.

When we select FPC-2 in Fig. 3.7-b3, Fig. 3.7-b4 accounts for overall variability in the temperature amplitudes in the entire duration of measurement. The perturbations (shown in dotted red/yellow lines) show that the related curves flip across the mean at around time point 80, and there is another flip at around time point 200. The captured variability is also seen in Fig. 3.7-b2. Therefore, FPC-2 accounts for larger amplitude fluctuations in the readings over the time period of the measurements. The corresponding curves are also highlighted in the MS plot with the red outlines Fig. 3.7-b1. Hence, with this example, we find interesting patterns in variations within a group, which would have otherwise gone unnoticed.

Once these patterns are identified, it could be modeled to detect similar patterns in the MS plot. For example, by adjusting weight w_t^T assigned to each time point for computing MO and VO, we can make the MS plot more sensitive to value changes in the corresponding time periods (e.g., $120-200$ in the above case).

3.4.3 Study 3: Analysis of Supercomputer Hardware Logs

We analyze rack environment logs obtained from a supercomputer. Rack environment logs provide information collected from various sensors housed in the compute rack subsystem (system board, air/water cooling, power supply, etc.) of a supercomputer. These logs contain readings of voltages, temperatures (water/air/CPU), fan speeds, etc. that can be utilized to find abnormalities during failure events. Therefore, studying

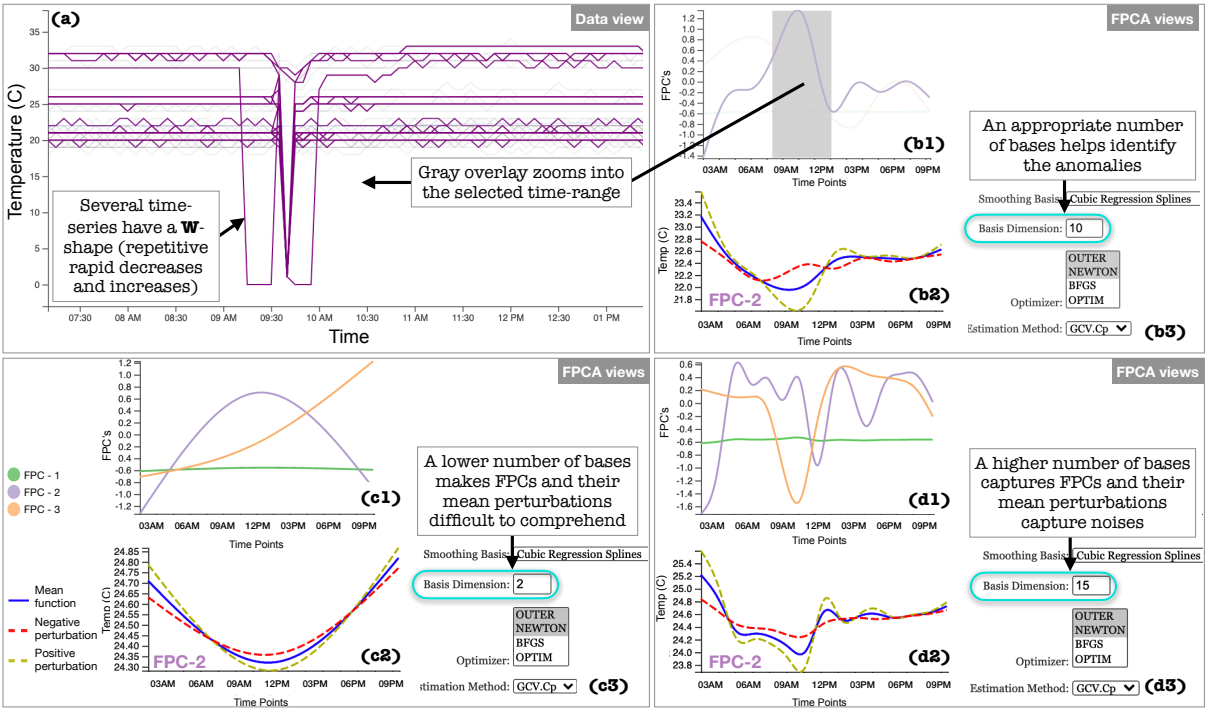

Figure 3.8: The effect of the number of basis functions on the FPCA results. (a) shows the data view. (b1, b2, b3) the FPCA views produced with 10 basis functions. Similarly, for (c1, c2, c3) and (d1, d2, d3), 2 and 15 are used as the numbers of basis functions, respectively.

these logs helps us understand the various underlying hardware failure patterns. These studies aid in maintaining robustness and improving reliability in large-scale machines. Thus, there have been past efforts to identify and analyze supercomputer log errors with visual analytics tools [47, 59, 151].

We specifically analyze data procured from the K computer's [118] rack environment logs on November-17th, 2017. The logs contain data from 864 compute racks, with 1,163 different sensor measurements (e.g., CPU temperatures, circuit voltages, fan speeds) collected every 5 minute interval (i.e., 288 timestamps in a day). For this case study, we select to analyze temperatures collected at 180 sensors per rack. Therefore, in total, we end up with $180 \times 288 = 51,840$ sensor measurements per rack. We have already analyzed the same data in Sec. 3.3.3 with the MS-plot, and we have identified different temporal patterns with FPCA. Here, we demonstrate the effect of settings of the smoothing function on FPCA in more detail.

The data collection used for FDA is usually observations gathered discretely over

time at fixed or random time intervals; however, many FDA methods, including FPCA, expect input data to be gathered at fully defined trajectories, i.e., at time interval close to zero. This leads to an analysis problem with an extremely large number of time points. Therefore, to model the underlying stochastic process of the data while overcoming the curse of dimensionality, data smoothing is often applied. While the smoothing helps in regularization and elimination of unnecessary roughness or noise from the analysis, the settings related to smoothing are highly important to obtain the desired FPCA results.

Fig. 3.8 shows how the choices in the number of basis functions used to model our dataset changes the FPCA results. In Fig. 3.8-b3, we choose 10 as the number of basis functions (the same setting with Fig. 3.3). The corresponding views are updated (b1, b2). In Fig. 3.8-b1, we select FPC-2, and we use the brushing tool, as shown with a gray overlay, to select the time range that shows the rapid increase in the FPC. Then, as shown in Fig. 3.8-a, the data view zooms into the related time range and shows a set of time-series highly related to FPC-2 within that range. Therefore, with the selection of an FPC and brushing, we can identify time-series and a time range that associate with peculiar variations on the selected FPC. However, if we use too few basis functions to model the functional phenomena, we may not effectively capture important underlying information. For example, in Fig. 3.8-c3, we select 2 as the number of basis functions. This choice is able to model the drop in some time-series by FPC2, as shown in Fig. 3.8-c2; however, this setting fails to capture the sudden drop and return, which is well modeled in Fig. 3.8-b1, b2. On the other hand, as shown in Fig. 3.8-d1, d2, d3, where the number of basis functions is 15, using too many basis functions makes FPC-2 too closely following the dataset. This imply that the smoothing function performs curve-fitting rather than smoothing and captures too much noise in our data.

In Fig. 3.9, we select FPC-2 and FPC-1, and the linked views are updated accordingly, Fig. 3.9-a for (FPC-2) and Fig. 3.9-b for (FPC-1). Fig. 3.9-(a1,b1) shows user selections and Fig. 3.9-(a2,b2) are the corresponding FPCs. To review more details of the user-selected readings in Fig. 3.9-a1, related to FPC-2, we select FPC-2 from Fig. 3.9-a2; then,

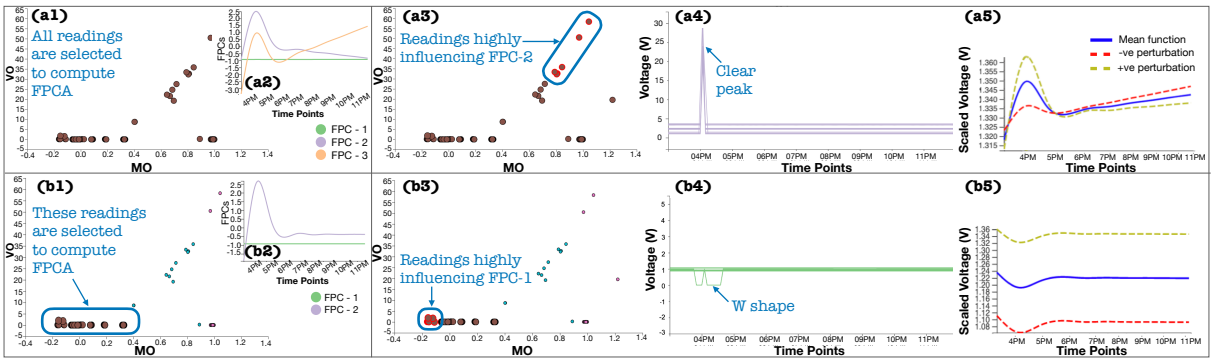

Figure 3.9: All readings are selected in the MS plot (a1) and then three FPCs are computed with the selected readings (a2). From FPCs, FPC-2 is selected (a2). The corresponding readings are highlighted in the MS plot (a3) and visualized in the data view (a4). The FPC as a perturbation of the mean plot is updated accordingly (a5). The similar procedure is applied to (b1–b5), where two FPCs are generated with the readings of low MO and low VO (b1, b2) and then FPC-1 is selected (b3–b5) (© 2022 IEEE).

the readings that heavily influence FPC-2 are shown in Fig. 3.9-(a3,a4). In Fig. 3.9-a3, the corresponding readings are highlighted with red outlines, and they have large MO and VO. In Fig. 3.9-a4, the readings show a spike at around 4 PM, lasting for about 30 minutes. Fig. 3.9-a5 shows how these readings fluctuate with respect to the overall mean of the selected readings (shown in navy blue). This plot helps us identify the overall trend of the readings when compared to the mean behavior of the selected readings. The readings show an overall flipping across the mean function at 4 PM. This example demonstrates how the MS plot captures outliers and how the FPCA view can also find similar outliers to validate the findings from the MS plot.

In Fig. 3.9-b1, we select the area with low MO and low VO. Fig. 3.9-b2 shows 2 FPCs that capture all the data variation. As this area is expected to follow the main trend because of low VO, we choose FPC-1 and visualize the top-5 readings that heavily influence FPC-1 (Fig. 3.9-b3, b4), which contain the "W" shape along with flat curves. This variation of the curves shown in Fig. 3.9-b4 is smaller when compared to the peak in Fig. 3.9-a4. The MS plot captures this irregular pattern by placing these readings in the negative MO and slightly higher VO, as highlighted in Fig. 3.9-b3 with red outlines. In Fig. 3.9-b5, the FPC perturbation to mean plot captures this small variation and the overall trend with respect to the mean behavior. Using this example, we show that

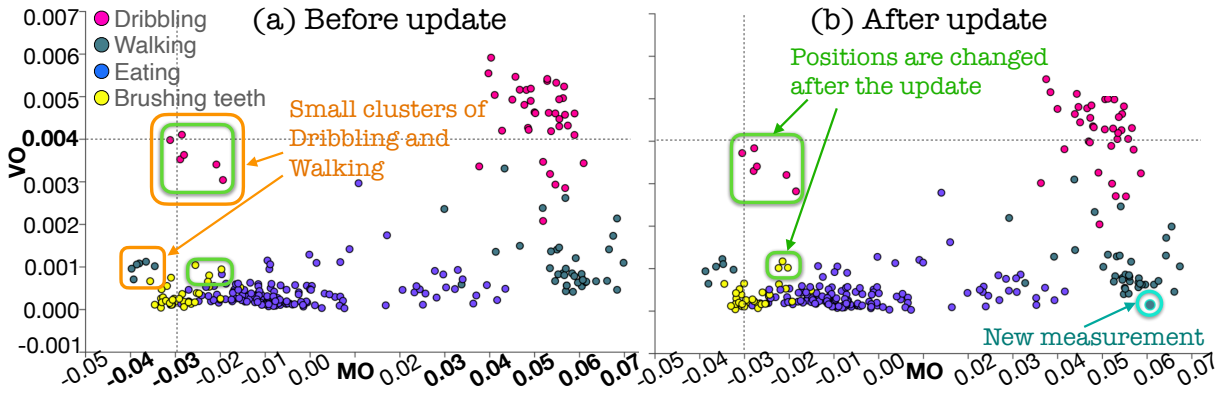

Figure 3.10: The MS plot for accelerometer measurements (a) before and (b) after the update. (a) is generated with 1,000 time points (50 seconds), and (b) with additional 100 time points and 1 measurement (© 2022 IEEE).

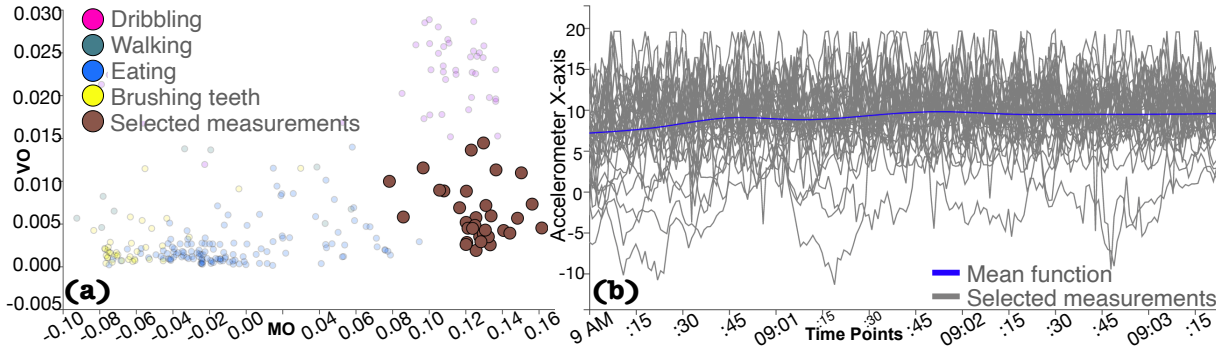

Figure 3.11: The (a) MS plot and the (b) data view. The data view is shown for selected measurements (brown) from (a) (© 2022 IEEE).

incremental MS plot and FPCA help identify intra-cluster trends by grouping similar trends within the same spatial locality of a cluster.

3.4.4 Study 4: Analysis of Biometrics Data of Daily Activities

Fig. 3.10-a shows the MS plot generated from 1,000 time points (50 seconds) for subjects performing activities, such as dribbling, walking, eating, and brushing teeth. Here the circles in the MS plot are colored based on the subjects' activities. We see that the MS plot generally groups the different activities well. This result is reasonable because each activity involves different frequencies and magnitude of changes in the subject's hand position. For example, when compared with brushing teeth and eating, dribbling and walking involve much larger movements, resulting in high MO. Also, as dribbling is the only sports activity, we can expect that it has a significantly different shape of time

series from the other activities; thus, dribbling has high VO.

However, we find that the individual walking and dribbling activities are separated into two clusters. One cluster in each activity is placed at the far left of the MS plot compared to the other, as shown in orange box. On examining MO values, we notice that each cluster placed at the left side has negative MO values but similar magnitudes of MO with the corresponding other cluster (e.g., for walking, one has MO values from -0.04 to -0.03 while another has values from 0.03 to 0.07). Also, these two clusters for each activity have similar VO values. From these observations, we can consider that the subjects in those clusters with negative MO values might have worn the smartwatch differently from most subjects; consequently, x measurements recorded the opposite signs from the others. This demonstrates the usefulness of the MS plot to find these anomalies visually.

While performing incremental updates, we observe that circles corresponding to either dribbling or brushing teeth show fluctuations in the MS plot. For example, in Fig. 3.10-b, where the MS plot in Fig. 3.10-a is updated with 100 additional time points, we see that the circles shown in green box have small but noticeable changes in their positions (e.g., the annotated `Dribbling` cluster is moved down). This is likely because dribbling and brushing activities involve frequent hand movements when compared to the other activities. The collective use of the MS plot and the animated transitions

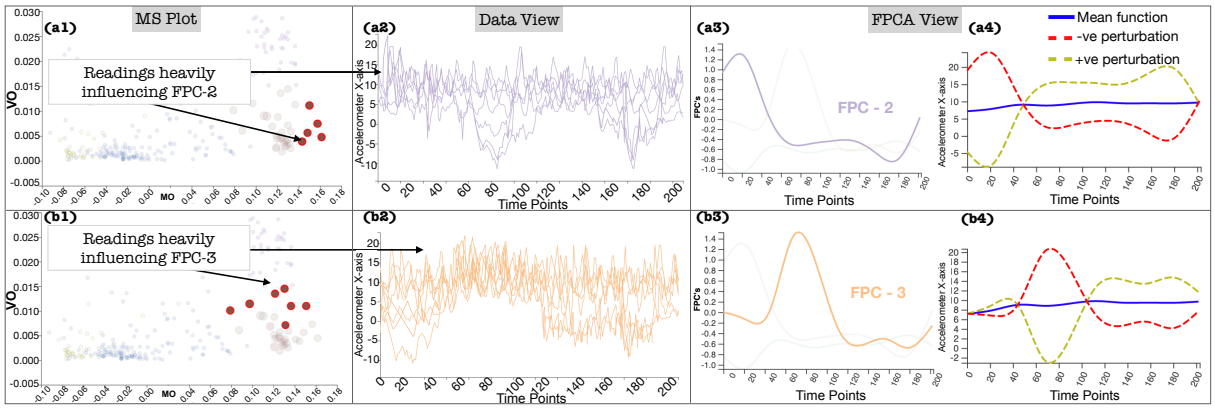

Figure 3.12: The MS plot, data view, FPCA view after the selections of FPC-2 (a1–a4) and FPC-3 (b1–b4) from Fig. 3.11 (© 2022 IEEE)

.

(described in Sec. 3.3.2.4) helps us capture these small changes that occurred across the updates. Fig. 3.10-b also shows a new measurement (annotated by teal circle) added to the MS plot using the progressive update for a subject performing the walking activity. Although the new addition is made with the approximation, it is reasonably close to the cluster for walking activities. Further evaluations of the effect of the approximation are presented in Sec. 3.5.

Next, we review a cluster using the MS plot and FPCA together. As an example, as shown in Fig. 3.11-a, we select a cluster (highlighted by brown) corresponding to walking with large MO and relatively small VO. For the sake of clarity, we have chosen to show the MS plot after obtaining the first 200 time points (10 seconds). Fig. 3.11-b shows the data view for the selected time series/circles. Because of the visual clutter caused by many curves, not much can be deduced from the current data view. Using FPCA, we can still identify trends and patterns in such cluttered time series. We apply FPCA with the default setting and observe that the first 3 FPCs correspond to 90% of the variation in the selected time series. We select FPC-2 and FPC-3, and the linked views are updated accordingly, as shown in Fig. 3.12-a (FPC-2) and Fig. 3.12-b (FPC-3). Fig. 3.12-(a1,b1) and Fig. 3.12-(a2,b2) show the MS plot views and the data view for the biometrics data, respectively. FPC-2 (Fig. 3.12-a3) has a distinct peak at earlier time points followed by a drop at later time points, while FPC-3 (Fig. 3.12-b3) has a distinct peak at around time point 80. Similar patterns can also be seen in Fig. 3.12-a4 and b4, which inform the positive and negative perturbations of the measurements with respect to the mean function of the selected subgroup. These peaks clearly capture a phase shift in the subjects' walking patterns. The corresponding MS plot and data views (Fig. 3.12-a1, a2, b1, and b2) show the measurements that heavily influence the selected FPCs. Now, the phase shift is visible in the data view. For example, in Fig. 3.12-b2, we can see that multiple subjects tend to have larger values around time point 80 than at other time periods. Also, from the MS plot view in Fig. 3.12-b1, we can see that the patterns in FPC-3 tend to be seen only in the circles with larger VO than the other selected circles. Similarly, the circles that heavily influence FPC-2 can be seen only in

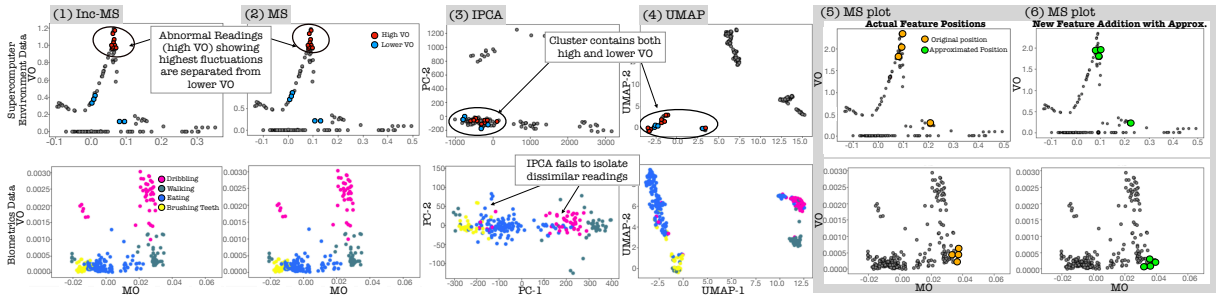

Figure 3.13: The comparison of results among the (1) incremental MS plot (`Inc-MS`), (2) MS plot (`MS`), (3) incremental PCA (`IPCA`), and (4) `UMAP`. (5) shows the exact position of features obtained by `MS` and (6) shows the progressive addition of the same features with approximation (© 2022 IEEE).

the area with large MO.

To summarize, a combination of the incremental, progressive MS plot and FPCA helps filter distinct patterns in time series data of activities. Certain activities form a unique cluster of patterns or an anomaly cluster. Analyzing these clusters further show us what these patterns look like and where they lie within the corresponding cluster. While this scenario uses the measurements from daily activities, similar high-velocity data is often collected, for example, for clinical care [39], and our approach would be useful to monitor the measurements to make time-critical decisions.

The four case studies demonstrate how a combination of the MS plot and FPCA can identify not only outliers but also peculiar behaviors of data within these outliers or find missing outliers in the MS plot to improve the visual outlier detection with the MS-plot.

3.5 Discussion

We have presented our approach's effectiveness to analyze streaming time series data and its applicability to various datasets and future use. We further discuss the strengths of the MS plot by comparing it with DR methods often used to categorize time series data visually [2]. We provide expert feedback to support the effectiveness of our tool.

Comparison with DR methods. Fig. 3.13 compares 2D plots generated by using ordinary MS plot [24] (`MS`), our incremental MS plot (`Inc-MS`), incremental PCA (`IPCA`) [140], and `UMAP` [115] for supercomputer hardware logs and biometrics data to show how these

methods capture the underlying data patterns. IPCA and UMAP are chosen as representative linear and nonlinear DR methods that are used for visual pattern identification in multiple time series [2,47].

First, as expected, for both datasets, MS and Inc-MS produce identical results since Inc-MS does not apply any approximation. For the supercomputer logs, we compare the four methods' ability to discern the outlier readings with large VO (abnormal) and lower VO (normal). To avoid visual clutter, we ignore the rest of the readings shown in gray color. UMAP and IPCA produce discernible clusters. Although most of the clusters group readings with similar trends, some of the clusters show both normal (blue) and abnormal (red) readings (verified by using the data view in our tool). This issue can occur as they are not specifically designed to capture outlyingness. For the biometrics data, while MS and UMAP isolate different activities into clusters, IPCA fails to do so. The UMAP result with the setting (specifically, n_neighbors=4, min_dist=0.1, and metric="euclidean") that showed reasonable clarity between clusters still show larger overlap in the measurements from different activities (e.g., dribbling and eating) when compared to MS. Another strength of the MS plot is in its interpretability. Since the MS plot directly shows the magnitude and shape outlyingness in the x- and y-coordinates, we can easily interpret why some time series are grouped together (as demonstrated in Sec. 3.4). On the other hand, DR methods such as incremental PCA and UMAP require additional steps to understand each revealed cluster [44]. In Fig. 3.13-5 and 6, we examine the effect of approximation in our progressive algorithm by comparing it to the exact solution obtained with MS. The added time series are colored orange and green in each result. When the errors are within the user-specified error threshold, the approximations can be assumed to produce close to valid results. In fact, as shown in Fig. 3.13-5 and 6, the newly added time series are in close vicinity of the actual values.

In Fig. 3.14, we use the supercomputer hardware logs to compare the completion time of methods and their streaming counterparts used in Fig. 3.13. For this evaluation, we use the same experimental platform as the one used in Sec. 3.6. We scale the data starting with a data size of $N \times T = 100 \times 100$ to $30,000 \times 30,000$. We use scikit-learn's

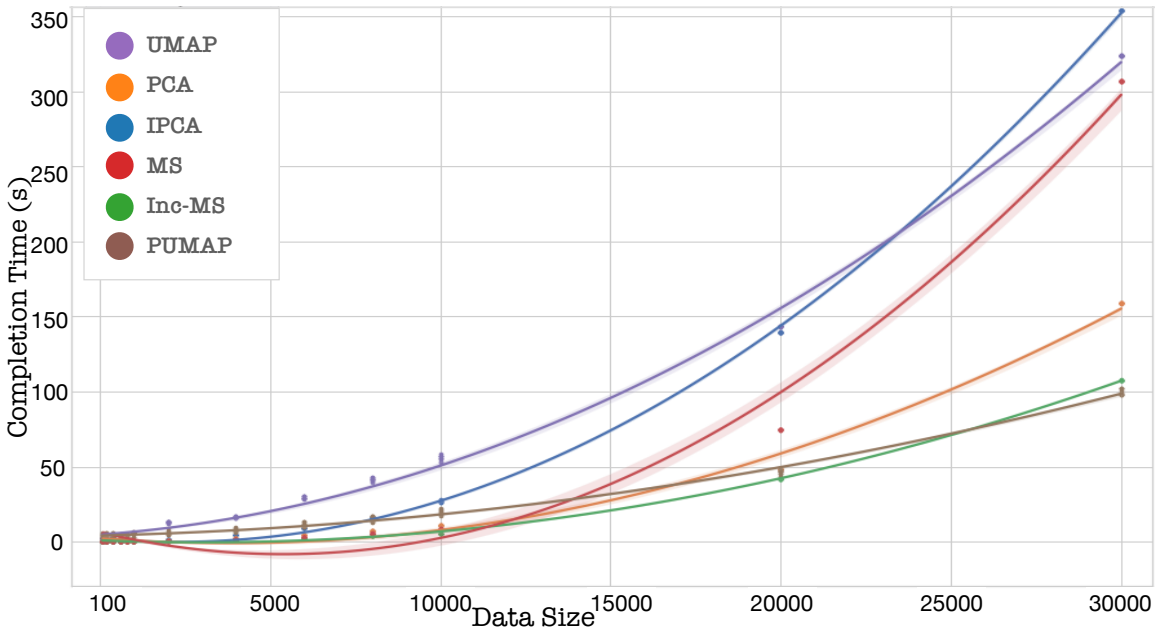

Figure 3.14: The comparison of completion time, showing how performance scales with datasize. We use a regression plot to interpolate the results (© 2022 IEEE).

implementation for `IPCA`, McInnes et al.'s implementation for `UMAP` [115], and Ko et al.'s implementation for `PUMAP` [92]. `Inc-MS` provides a method that incrementally updates the results, which is called *partial fit*, and a method that processes input data all at once called *(initial) fit*. We start by processing 5% of the time points with the *initial fit* and use *partial fit* containing 5 time points for the rest. We use a regression plot to interpolate the results. In Fig. 3.14, for each data size, the `Inc-MS` plot either gave approximately the same performance as `MS` (for data size in the range of hundreds) or outperformed `MS` by a factor of 2 times for a data size of $20,000 \times 20,000$ and 3 times for the data size of $30,000 \times 30,000$. We have improved the original MS algorithm by updating the results already computed rather than recomputing the overall result every time a new time point is added; this significantly improves performance. `Inc-MS` implementation outperforms `MS`, `PCA`, `IPCA`, and `UMAP` while having a runtime close to `PUMAP`. `PUMAP` provides approximated results, but `Inc-MS` provides exact results. Note that as N increases, `PCA` outperforms `IPCA` [43].

Expert feedback. We evaluated our tool with two industry experts with over 5 years of experience analyzing large-scale computing systems' hardware logs daily. We installed

the tool locally on their system and provided the details of each component's functionalities. The experts were impressed by our overall design and provided a detailed assessment of its components. They had no trouble interpreting the details presented in each view. For the overall system, they commented: "*The updates to the MS plots showed a smooth transition and were easier to follow. Upon selecting a cluster of measurements in the MS plot, we can view where the similar time series lie on the system in the space view and what pattern they represent in the data view. The FPC plots provide a summarized representation of the selected data, which was helpful.*" They indicated that the UI provides a good intuition of the overall system behavior in real-time. The overall feedback is encouraging, although they also noticed some limitations of the tool. Upon selecting from the FPC plot, the measurements that highly influence the selected FPC are highlighted in the data view and the MS plot. However, the experts mentioned that it would be useful to view components having little to no influence on this FPC as well. At the time of writing this dissertation, we have added a range slider that controls the threshold of selecting influential time series (circles, curves) on FPCs.

3.6 Performance Evaluation

We evaluate the performance of our algorithms for two example use scenarios. In each scenario, we mimic a practical streaming analysis situation by feeding time points and time series from a real-world dataset. For the experiments, we use the MacBook Pro (13-inch, 2019) with a 2.8 GHz Quad-Core Intel Core i7 processor and 16 GB 2133 MHz LPDDR3 memory. Completion times are averaged over 10 executions.

Evaluation with supercomputer hardware logs. Supercomputer hardware logs, which contain readings of voltages, temperatures (water/air/CPU), fan speeds, are collected from various sensors housed in the compute rack subsystem (e.g., system board and air/water cooling). Studying these logs to identify failure patterns can aid in improving the robustness and reliability of large-scale machines. There have been past efforts to analyze supercomputer/cloud hardware logs with visual analytics tools [47, 59, 151, 180, 181]. Our logs (analyzed in Sec. 3.3) are collected from the K computer [118] and

contain data from 864 compute racks, with 1,163 different sensor readings per rack collected every 5-minute interval (i.e., 288 timestamps in a day).

We examine our incremental and progressive algorithms' efficiency for this log data, which has extremely high volume but relatively low velocity (5-minute interval). We assume that we have already had the data of size $4,032 \times 336,960$ (i.e., 2 weeks and $336,960$ temperature readings across all racks). Then, we add a newly arrived time point to this existing data. While the update of the MS plot without our incremental algorithm (i.e., recalculation on $4,033$ time points) is finished in 8.2 minutes, the incremental addition of a time point is completed in 2.5 seconds. To evaluate our progressive update with approximation, we first process the same data, but one time series is excluded (i.e., the size of $4,032 \times 336,959$). Then, we add one time series and update the MS plot using our algorithm with approximation, which is completed only in 16.6 seconds, while the overall update without approximation is done in 8.4 minutes. For both cases, the updates without our algorithms exceeded the data collection interval and caused wastage of computing resources.

This proof-of-concept experiment shows that our algorithms achieve prompt updates for large datasets. In practice, the analysts may select a smaller dataset (e.g., 1 day instead of 2 weeks), and the ordinary MS plot may update the result within the current update interval (i.e., every 5 minutes). However, most recent supercomputers collect sensor data at 0.03–10Hz [65, 143] (i.e., about every 0.1–30 seconds). Our approach can process large volumes of data to promptly identify anomalies in these large-scale systems.

Evaluation with Biometrics data. The generic design of our tool enables us to monitor time series collected from different types of hardware systems. As another use scenario, we analyze data collected from smartwatch accelerometer sensors of 51 subjects [178]. The subjects performed daily activities, including ambulation (e.g., walking, dribbling), fast hand-based (e.g., brushing), and eating activities. Each subject contributed 54 minutes of data collected at 20Hz sensor polling rate for a total of 3 minutes per activity. Each row in the dataset contains *Subject-id*, *Activity Code*, *Timestamp*, x, y, z.

For brevity, we use the sensor value for the x-axis, i.e., signed acceleration along the x-direction. However, one can also individually apply our algorithms to each of the other measurements of x, y, and z.

Here we assume that we have already collected the data of size $30,000 \times 5,000$ (i.e., 25 minutes and 5,000 subject-activity pairs). Then, we incrementally add $1,200$ time points (i.e., every 1 minute). While the incremental additions of $1,200$ time points are completed in 0.8 seconds, the update without using our incremental algorithm is finished in 18 seconds. For our progressive update with approximation, we further add 10 time series into this data, and the update is done in 5.5 seconds. Without the progressive update, the overall process time was 19 seconds. These results show that updates with our algorithms are fast enough to handle a large number of time series with extremely high velocity (i.e., 20Hz, or 0.05-second interval) in real-time.

Experimental evaluation with different data sizes. We further evaluate our algorithms with different numbers of time points and time series while using the supercomputer hardware logs (`SC Log`) and the biometrics data (`Biometrics`). Tables 3.1 and 3.2 show the completion times for the addition of new time points and new time series, respectively.

In Tab. 3.1, we first process an initial set of data consisting of varying time points (100–20,000) and a fixed number (1,000) of time series. We then add one new time point and update the existing outlyingness measures incrementally. The completion time for the initial data fit increases as time points are increased, while the incremental addition has the completion time *independent* of the data size. The incremental addition always takes about 1ms for `SC Log` and 180ms for `Biometrics`, which are fast enough to support online streaming analysis. Note that `Biometrics` is characterized by widely varying high-frequency fluctuations, and the data complexity contributes to the computation time where the bottleneck relates to the depth measures.

For the time series addition, as shown in Tab. 3.2, we first process an initial set of data consisting of varying time series (100–10,000) and a fixed number of time points (1,000). We then add one new time series and update the existing outlyingness measures using

Table 3.1: Completion time (in seconds) of the initial data fit (Initial Fit) and the incremental addition of one time point (Partial Fit) (© 2022 IEEE).

Dataset	N	T	Initial Fit	Partial Fit
`SC Log`	1,000	100	0.0109	0.0009
`SC Log`	1,000	1,000	0.0640	0.0011
`SC Log`	1,000	10,000	0.5373	0.0013
`SC Log`	1,000	20,000	1.3357	0.0010
`Biometrics`	1,000	100	0.0457	0.182
`Biometrics`	1,000	1,000	0.388	0.173
`Biometrics`	1,000	10,000	2.537	0.196
`Biometrics`	1,000	20,000	4.990	0.176

Table 3.2: Completion time (in seconds) of the initial fit and progressive addition of one time series with and without approximation (© 2022 IEEE).

Dataset	N	T	Initial Fit	Approx.	Non-approx.
`SC Log`	100	1,000	0.0038	0.0006	0.0088
`SC Log`	1,000	1,000	0.0495	0.0039	0.0654
`SC Log`	10,000	1,000	0.5910	0.0680	1.0079
`SC Log`	20,000	1,000	1.7038	0.1009	2.5148
`Biometrics`	1,000	1,000	0.0520	0.0027	0.1890
`Biometrics`	5,000	1,000	0.3350	0.0184	1.1533
`Biometrics`	10,000	1,000	0.7328	0.0217	1.690

the progressive algorithm with and without the approximation.

With the approximation, it takes less than about 100ms to compute. Without the approximation (i.e., equivalent to an ordinary plot), the overall computation time would be almost close to the initial fit of the data, leading to more significant wait times.

Although FDA has gained more traction in recent years, discrete analysis is largely preferred over its functional counterparts. This is mainly because the analyses are computationally less expensive and are easily accessible through multiple implementations in popular programming languages. In this work, we have demonstrated how FDA in an online streaming analysis environment can help identify and capture the underlying data characteristics. With the use of FPCA, through FPC's and functional mean per-

turbations, we identify modes of variations within clusters over the continuum, which would have gone unnoticed using a traditional PCA.

This research was supported in part by the U.S. National Science Foundation through grant IIS-1741536 and the Argonne National Laboratory through contract 8F-30225. We thank Keiji Yamamoto for providing useful information regarding the supercomputer data.

3.7 Conclusion

The advent of exascale systems makes it pertinent to build applications that can process data in a timely and reactive manner. We have built a visual analytics tool that processes large streaming time series data, which are intrinsically functional, using functional data analysis. This is achieved through incremental and progressive computations of the magnitude-shape outlyingness measure through the addition of new time points or new time series. To further understand the underlying errors, we use functional principal component analysis to identify different modes of variation and the type of fluctuation (amplitude, phase shift) for each mode. We plan to extend our tool's visualizations to support the analysis of multivariate functions and also incremental and progressive algorithms to use different depth measures.

Chapter 4
A Visual Analytics Solution for Analyzing Multifidelity HPC System Logs

4.1 Introduction and Background

Maintaining robust and reliable computing systems, especially those that enable breakthrough work in computational science and engineering research, is critical yet challenging. These systems are expected to run jobs that take weeks, and any failure in the system could add significant overhead in the already computationally expensive research and development. Therefore, understanding the underlying system properties from diverse subsystems is an initial step in an attempt to better understand the current failures that occur in the system. We focus our efforts on the Cray XC40 system, specifically the Theta supercomputing system deployed at ALCF [5]. Theta is an 11.69 Petaflops system that was designed in collaboration with Intel and Cray. The system comprises of 4,392 nodes, each containing a 64-core processor, 192 GB of DDR4 RAM, 16 gigabytes (GB) of in-package high-bandwidth memory (MCDRAM), and a 128 GB SSD. The 4,392 nodes are housed in 24 racks, interconnected using an Aries interconnect with a Dragonfly configuration designed for good global bandwidth and low latency. The data we procure from Theta is primarily raw data captured from sensors, and the size of these logs are typically in the terabytes range. Therefore, we deal with large-scale logs from diverse sources with multiple fidelities. For example, the environment log data is captured at a frequency of approximately 30-second intervals. In this case, a few weeks of this log data is hundreds of gigabytes. The hardware and job log data size ranges in tens of gigabytes. The goal of this work is to build a scalable visualization

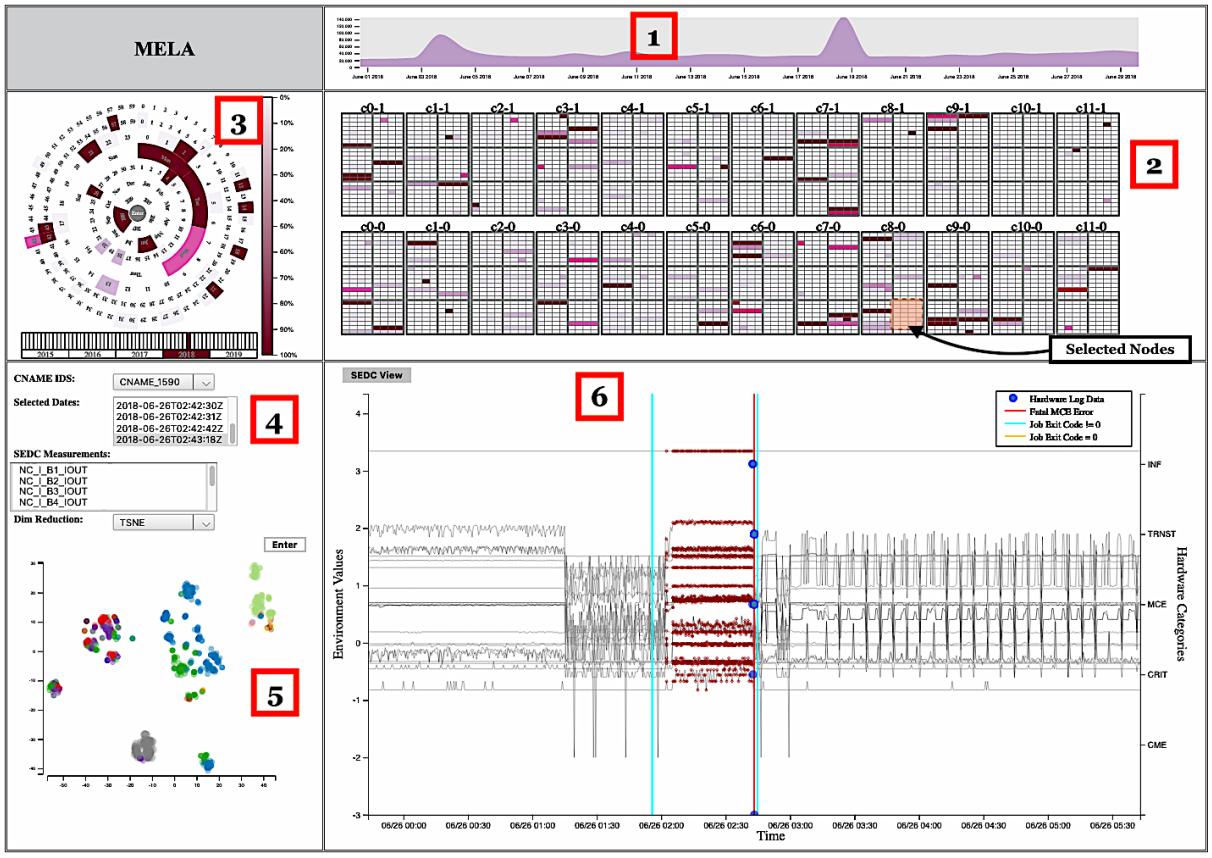

Figure 4.1: Error Log Analyser: 1) Linear Timeline View for review and selection of errors. 2) Node Layout View for distribution of hardware log data at each node filtered by the type of log (e.g. informational, warnings, fatal, etc.). 3) Radial View for distribution of the hardware error data along the radial time dial. 4) Selection View for filtering node ids and timeframe for analysis of environment log (SEDC) data. 5) Dimensionality Reduction on the SEDC data clustered using Weighted Clustering Ensemble with Different Representations 6) Detailed View of SEDC log data for the selected timeframe with hardware and job log information (© 2019 IEEE).

system that allows us to analyze data with temporal and spatial locality. When it comes to hardware errors we look at data from multiple control systems, which are linked to one another. Hence, we deal with duplicate events that need to be preprocessed. Our contributions using the Theta supercomputer logs will apply to other Cray XC40 systems deployed at various facilities worldwide, including several U.S. Dept. of Energy national laboratories.

Significant efforts have been made to improve system resilience through system log analysis in the past. Failure detection and root cause diagnosis use diverse log sources that could be used to address failures. Although there are many automatic

log analysis tools, little work has been done with visualization to successfully perform root cause analysis and correlations of failures. Some of the automatic log analysis tools [31, 49, 51, 64, 114, 192] use correlation analysis, signal analysis, pattern mining, event pattern-based correlations, resilience at application level and spatial/temporal event analysis. HELO [51] is an event log mining tool that extracts event formats via messaging templates using pattern mining log files from large-scale supercomputers. ELSA (Event Log Signal Analyzer) [49] is a toolkit for event prediction, modeling the normal flow at a stable event state and following the abnormal flow in an event of system failure. One of the steps involving log analysis is data filtering. An adaptive semantic filtering (ASF) method [103] efficiently filters the duplicated messages in a log by computing the semantic correlation between two events. LogAider [31] and LogMaster [42] are tools used for mining event correlations but use generic, easy-to-use visualizations. LogAider reveals potential correlations that include across-field, spatial, and temporal correlations. LogMine [64] is an unsupervised, scalable end-to-end one-pass framework for the analysis of massive heterogeneous logs. LogDiver [114] provides support for lossless data compression, modeling application failure paths and cross-validation of models/results. However, LogDiver and LogMiner do not provide visualization support. DeepLog [32] is a deep neural network model that uses stacked Long Short-Term Memory (LSTM) networks for anomaly detection, dynamically updating the models to accommodate changing log patterns. However, it does not have lead time analysis. RAVEN [130] maps RAS (reliability, availability, and serviceability) logs on a physical system map for Titan, a Cray supercomputer at Oak Ridge National Laboratory. IBM provides Blue Gene Navigator [97] used by system administrators to monitor the system through basic log statistical visualizations. These tools lack the support for scalable and interactive visual analysis of HPC system logs. La VALSE [59] is a tool with support for a scalable user interface. They explore large amounts of hardware data in supercomputers. However, analyzing one or two types of log data alone would not suffice to provide complete insight in comprehending the complexity of such large-scale systems. Therefore, we analyze multiple logs (hardware, jobs, and

environment) for our study.

Our contributions in this work [151] are as follows:

- We built a scalable interactive visualization tool, which we call MELA (Multifidelity Event Log Analyzer), for studying event log data.
- The tool has a built-in user interaction mechanism which allows the analysis to be rerun based on feedback.
- We have designed a ring layout for visualizing the spread of errors across the time scale.
- With MELA, we analyzed diverse log datasets comprising of hardware, job, and environment logs collected at multiple fidelities.

4.2 A Visual Analytics Tool - MELA

In this section, we describe the interactive visual analytics tool, MELA, for distributed processing and analysis of the Theta logs. MELA uses the D3 [11] Javascript frontend and a combination of InfluxDB [76] and Elasticsearch [55] databases at the backend. The tool consists of multiple sub-components. This section describes each of these sub-components and their functions.

4.2.1 Node Layout

The node layout for Theta is shown in Fig. 4.1(2). The system is composed of 24 compute cabinets, each with 3 cages, where each cage contains 16 blades and each blade has 4 nodes. The visualization in this figure shows that 24 racks are grouped into two rows, the racks form groups based on the network topology, and each group is displayed in the same column. The colors of each node (Fig. 4.1(1)) display the count of hardware errors for a selected time period. As an example here, we have shown the hardware errors for June 2018. The darker colored nodes represent a higher count in errors. This visualization also allows for the selection of nodes in groups or individual node selection for further filtering and analysis of the various control system outputs. The

orange rectangle overlay-selection tool allows for multiple selection of nodes. As an example, we have selected nodes in cabinet C8-0.

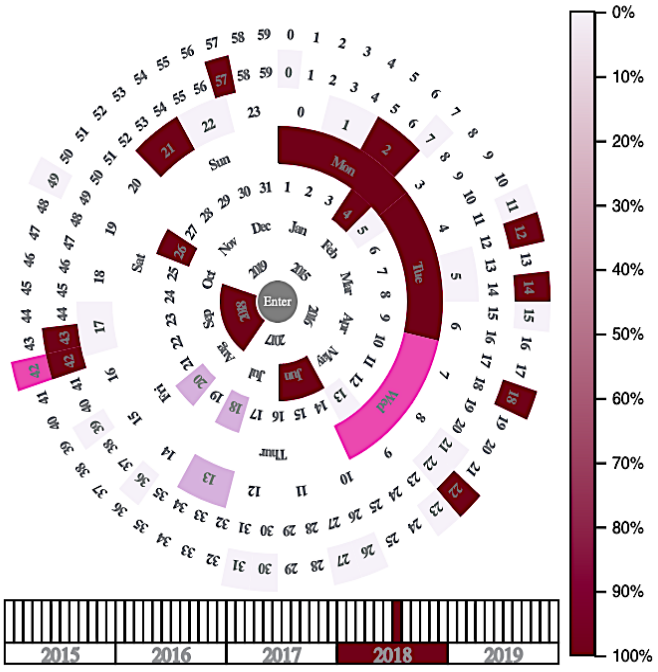

Figure 4.2: The ring layout represents the distribution of hardware errors in a concentric circle format which represents a clock face. The innermost to outermost rings show the years, months, days, days of week, hours, minutes, and seconds respectively. Here, June 2018 is selected (© 2019 IEEE).

4.2.2 Ring Layout

The ring layout (Fig. 4.2) allows for an aggregated view of the errors distributed over time. The ring layout represents time in a form similar to a clock. The rings from smallest to largest represent the years, months, days, days of week, hours, minutes, and seconds respectively. The colors in the ring represent the counts of errors distributed on the temporal scale. We see a larger number of errors occurring on Mondays and Tuesdays and mostly on the 4^{th} and 26^{th} of June. The maintenance of Theta occurs bi-weekly on Mondays. This could explain the source of the large number of errors occurring on Tuesdays. This visualization provides a better understanding of the temporal distribution of errors across multiple granularities when compared to traditional linear temporal views. We can view data for up to 10 years and could expand to include more years. The ring layout is an interactive view allowing users to select any of the

arcs within the ring to further filter the data displayed in the connected visualizations.

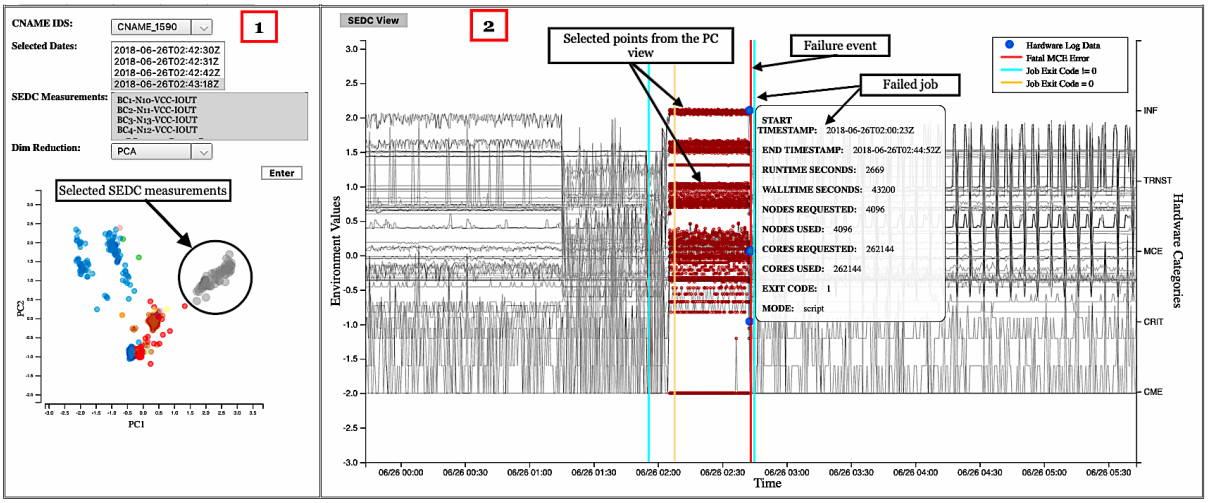

Figure 4.3: The figure shows the (1) Selection and Dimensionality Reduction layout on the left and the (2) Detailed layout on the right. Filter selections are made to investigate an error event that occurred at node 1590 at 2018-06-26 02:43:18. For more details refer to Fig. 4.4 (© 2019 IEEE).

4.2.3 Selection Layout

The third level of the visualization is a selection view, shown in Fig. 4.3(1). This view serves as a filtering mechanism where temporal, node id, and environment log (SEDC) measurements can be selected for further analysis. The SEDC log contains information about the power, temperature, current, fan speed, etc. at various fidelities. A user can view the distribution of errors on the radial view, linear timeline view, or the node layout, and one can further select the nodes and SEDC log measurements of interest. The timeframe is picked to be 180 minutes in either direction of the selected timestamp. This data holds useful patterns which would help with the error analysis.

We analyze the filtered nodes, timeframe and SEDC measurements using the weighted clustering ensemble method in different representation spaces [183]. The algorithm clusters data in multiple representation spaces, effectively minimizing information loss encountered by analysis in a single representation space. This approach allows for the capture of intrinsic patterns in the temporal data. The method uses a weighted pairwise similarity matrix derived from different cluster validation schemes applied to the partitions in the representation spaces. The cluster validation schemes

include Modified Huber's Γ index (MHΓ) [62], Dunn's Validity Index (DVI) [62], and Normalized Mutual Information (NMI) [159]. The three candidate partitions are then combined to a final partition using an agreement function [40]. It is less sensitive to small fluctuations in SEDC measurements and captures the overall trend in the patterns. The clustered multidimensional SEDC log data are then displayed in a 2D view using a chosen dimensionality reduction method, PCA [41], LDA [113] or t-SNE [171]. These methods are known to capture underlying data distributions with minimal loss in information. However, as we will see later in Section 4.3, a simple PCA, LDA or t-SNE will not suffice in capturing the underlying intrinsic structure of this diverse dataset. Therefore, we color the points by how they were clustered before the dimensionality reduction. The users can also select points using a lasso tool which then highlights the selected points (shown in dark red) in the detailed plot (Fig. 4.3(2)) shown on the right.

4.2.4 Detailed Layout

The fourth level of our visual analytics tool is a tabbed view. Fig. 4.3(2) shows the SEDC measurements plotted in gray, the values for which are shown on the y-axis to the left. The y-axis on the right shows the different categories of the hardware log data. The hardware log information is plotted as scatter plots shown in blue circles. As the user hovers over these points a tool-tip shows detailed information about the data. A fatal event such as a node shut down is shown as a red line. The detailed plot shows how different SEDC measurements behave around such fatal events. The users can remove any measurement from the analysis if it shows that a measurement skews the results unfavorably. For example, a cabinet-level measurement may not be affected by a node-level failure. Some interesting insights that we gather from this plot are: increase in blower fan speed and drop in voltage and current values during a failure event. More information is provided in the case study discussed in Section 4.3. The lines in cyan are the jobs that have ended with an exit code that is not zero. In the example shown in Fig. 4.3(2), we see that a job has failed following a fatal event. However, there have been cases where the job executes to completion despite a node shutdown. These are cases, such as with workflow managers with built-in resilience, when the jobs on the

compute nodes are offloaded to another idle compute node by the application using the node. The detailed view gives a high-level view of how the log information is spread across the timeline. Note that there is always a small intrinsic bias in these plots since the sensor clocks may not be perfectly synchronized.

4.2.5 Database backend

System log data get progressively larger as we deal with logs from various sources across multiple years. Our current environment log dataset is ~125GB in size for June 2018, making the entire dataset close to ~1.5TB for a year's worth of compressed log information to process. Therefore, there is a need to use a database that would allow for faster aggregations and efficient distributed processing of information, especially in the temporal scales. To cater to this need, we use a combination of InfluxDB [76] and Elasticsearch [55]. InfluxDB is a time-series database used for high writes and query loads. It is mainly used to store large amounts of timestamped data. Since our environment logs are timestamped and collected every 10-30 seconds, using this database is an appropriate choice for storing environment logs. For hardware logs and job logs, we use Elasticsearch which is an open-source, distributed, RESTful search and analytics engine capable of providing faster temporal aggregations. The timestamps for jobs and hardware errors could have significant overlap, so using InfluxDB would mean creating multiple redundant measurements. The aggregation of categorical data (stored as tags) in InfluxDB is not currently possible. Therefore, Elasticsearch is better suited for storing hardware and job log information. Each operation in the visualization such as filtering and selection makes an asynchronous call to the databases which allows for simultaneous and faster rendering of the data.

4.3 Case Studies

In this section, we describe two cases of node failure and how MELA helps identify failure events from this immense and diverse log dataset. We also illustrate why a dimensionality reduction technique alone would not suffice to capture the complex intrinsic patterns of the SEDC measurements.

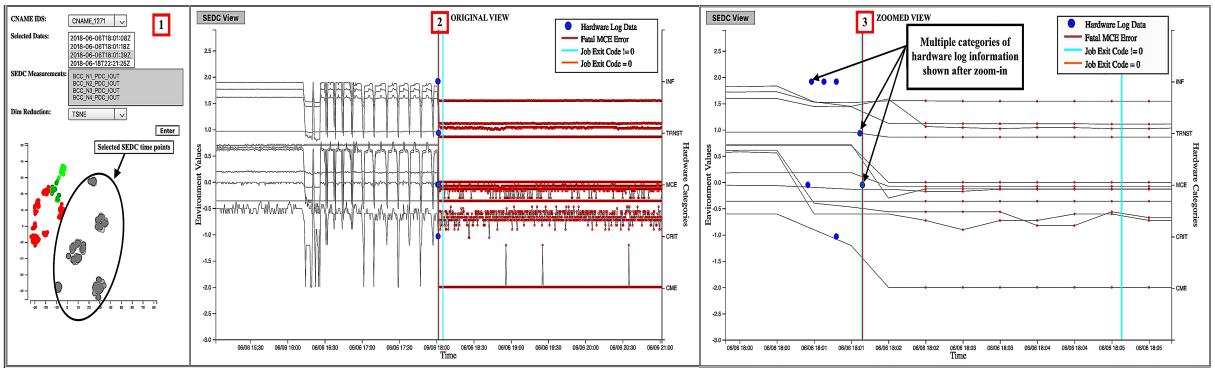

Figure 4.4: The figure shows the (1) Selection and Dimensionality Reduction layout on the left with PCA and the Detailed layout on the right: (2) original zoomed-out view (3) and user zoomed-in view. The filter selections are made to investigate an error event that occurred at node 1271 at 2018-06-06 18:01:39 (© 2019 IEEE).

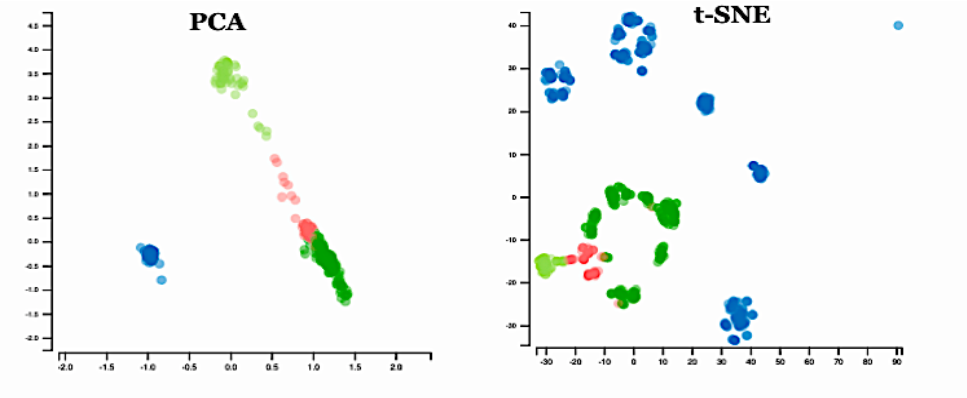

Figure 4.5: The figure shows the (1) Dimensionality Reduction view on the left with PCA and (2)Dimensionality Reduction layout on the right with t-SNE. The filter selections are made to investigate an error event that occurred at node 1271 at 2018-06-06 18:01:39 (© 2019 IEEE).

4.3.1 Case Study 1:

Fig. 4.4 shows the detailed layout view readings for node 1271. A failure event occurred at 2018-06-06 18:01:39 which led to a node shut down. The user can identify the fatal events of interest with the help of the node layout (Fig. 4.1(2)) and filter the desired timeframe using either the ring layout (Fig. 4.1(3)) or the linear time layout (Fig. 4.1(1)). The weighted clustering ensemble of the time-series data clusters the SEDC measurements for the specified timeframe. The selected clusters are shown in the dimensionality reduction (DR) view (colored gray) in Fig. 4.4(1). The dimensionality reduction method is only used as a means of plotting this higher dimensional data.

Using the lasso tool in the visualization, we select a single cluster from the DR view (shown in gray). This highlights the cluster points in the detailed view (Fig. 4.4(2)). We see that all of the time points after the fatal event have been clustered into one group. Similarly, the time points from 06/06 16:00 - 06/06 18:00 are grouped into the same cluster. Therefore, we can group similar trends over an extended timeframe using this methodology which is not always the case while using dimensionality reduction methodologies. The detailed layout has a zoomable (Fig. 4.4(3)) interaction which allows the user to zoom into the error event of interest and investigate further.

In the zoomed view, Fig. 4.4(3), each blue circle is a hardware error log event. A tool-tip showing more details about these events pops up as the user hovers over these circles. For this case study, a closer investigation revealed that a Processor Reorder Buffer (ROB) timeout was caused as a result of multiple Denali Core Non Fatal Errors on BUS0 causing “NL3" Processor Requests to not make forward progress on BUS0. The node failure caused a job running on the node using an overall 256 nodes to quit unexpectedly with approximately 3 hours still pending on the overall job walltime of 6 hours.

4.3.2 Case Study 2:

Fig. 4.3 shows the selection layout (1) and the detailed layout (2) for a fatal event that caused node 1590 to shut down at 2018-06-26 02:43:18. The failure event caused the job to stop unexpectedly within an hour of its 12-hour runtime. On closer inspection of the hardware log data, there are multiple instances of ROB timeouts and MCE (Machine Check Exception) errors that give a good initial understanding of the underlying events and how they are linked spatially and temporally. The node failure was due to uncorrectable errors on Banks 0 and 255–a system software error in this case. Therefore, the node was rebooted.

From Fig. 4.3(1) and Fig. 4.4(1), we see that the well-known dimensionality reduction techniques will not always capture the underlying trends accurately. In Fig. 4.3(1), although the SEDC time points before the failure have been clustered in one group, there is no clear separation between the other cluster points. Figs. 4.5(1) and 4.5(2)

show two different DR views for the same dataset. In Fig. 4.4(1), the gray points correspond to the selection of the blue cluster by the user. These are shown in the detailed layout in dark red on the right in Fig. 4.4(2). Although t-SNE is known to capture the underlying structure of the dataset with a good degree of flexibility, it is known to be hard to interpret. When using PCA, the best choice of the number of principal components that covers the maximum variance is something that the user needs to consider. Although there are improvements in the DR techniques to better identify the underlying structure of the data, our experiments have shown that the temporal weighted clustering ensemble method does a good job at capturing the intrinsic nature of the dataset (Figs. 4.3(2) and 4.5(2)). This algorithm is tuned to pick the ideal cluster count. Therefore, we use dimensionality reduction methods as a way to display the already clustered data.

The two case studies mentioned here have similarities in error tokens (messages) in the log data which can also be modeled using topic modeling. Some previous work [25] has used methods like Topics Over Time (TOT) [176], an unsupervised topic model loosely based on the LDA [10] model to identify rare compute node failures. MELA also provides support for the identification of similar log tokens, which is beyond the scope of the current work.

4.4 Conclusion and Future Work

We have developed a visual analytics tool for analyzing diverse log data of a Cray XC40 supercomputing system, specifically the Theta supercomputer at Argonne National Laboratory. We tackle two main issues: scalability and interactivity. We are currently able to interactively filter and analyze millions of records of log information with a built-in feedback mechanism. The feedback mechanism helps filter or include information that would help improve the deduction of error patterns in the data. Our visualization tool, MELA, enables a user, including system administrators, to analyze information in the terabyte range from various control systems at different fidelities from diverse logs. We present a novel way to visualize temporal data with hardware information drawn

on a temporal clock face visualization (ring layout). It provides a clean aggregation of how the errors are spread out along the temporal scale. We map the hardware logs onto a physical map (the node layout) of the Theta supercomputer. Researchers and system administrators could use this tool to identify and record how the events in diverse logs correlate and investigate events of interest further.

Our next goal is to incorporate end-to-end interactive visual analytics support for identifying patterns such as correlations, detecting anomalies and predicting faults. In conjunction with the TOT analysis, this tool could be used for root-cause analysis with built-in feedback in the visualization for incorporating corrections and other refinements. We plan to extend our visual analytics tool to be usable for analyzing data from different supercomputers.

Chapter 5
Machine Anomaly Detection and Prediction

5.1 Introduction

Understanding and maintaining robust and reliable supercomputers is critical for enabling advancements in research. High utilization of these systems is compromised if system failures cannot be handled in a reactive and timely manner. Jobs could run for weeks, and failures add significant overhead in already computationally expensive research. Therefore, we study these diverse subsystems' past behaviors and use the data to predict future behaviors. For this work, we focus our efforts on a 10+ Petaflops Cray XC40 supercomputer system with Intel Xeon Phi based compute nodes. The nodes are interconnected with an Aries interconnect in a Dragonfly topology. We use log data from the Cray XC40 supercomputer that can be broadly classified as (i) environment logs, also referred to as SEDC logs, where SEDC (System Environment Data Collections) is a tool used to collect and report environmental data on all Cray systems in real-time, (ii) hardware error logs, and (iii) job logs. The data are essentially raw data from sensors, located at a predefined spatial locality, and recorded at various temporal resolutions. Therefore, we deal with multi-fidelity large-scale data from diverse sources within the HPC system. For example, the SEDC log data is recorded and stored approximately every 10-30 seconds, so the SEDC dataset size approaches gigabyte to terabyte range every few weeks. The hardware error log contains data recorded from various control systems linked with each other within the subsystems. This results in duplicate events that need to be handled during data pre-processing. The hardware error log data ranges in tens of gigabytes, and the job log data size can be hundreds of

megabytes for a year. Altogether, these three types of logs are about 5 GB per day. It is a daunting task to process data of these volumes for any analyses.

We analyze three log data types and glean insights from their correspondence with each other at their various temporal and spatial resolutions. We use this information to build models that predict if a job will fail and job failure exit status. Our system comprises database storage with a machine learning back-end pipeline and a front-end visualization. The machine learning back-end pipeline is split into a SEDC view pipeline and a hardware/job analysis pipeline. The SEDC view pipeline aids in the processing and analysis of environment log data to identify patterns, some of which could indicate anomalous HPC system behavior. The hardware/job analysis pipeline is split into four stages (Fig. 5.1): (1) hardware topic generation, (2) hardware error prediction, (3) job failure prediction, and (4) job exit status prediction. We call Stages 3−4 the *job prediction pipeline*. In Stage 1, we group related hardware errors into topics using the Topics over Time (ToT) algorithm [176]. Stage 2, hardware error prediction, uses past hardware errors to predict the most likely future hardware errors at the component level on which a job is run. Then in Stage 3, we predict if a job will fail. If a job failure is predicted, in Stage 4, we predict the exit status, using the predicted hardware errors from Stage 2 and the hardware errors from the past. This system back-end comprises separate machine learning pipelines for analyzing and processing each type of log data since they vary in characteristic from text data to sensor data. In doing so we can identify how trends and patterns relate across types within the same temporal/spatial vicinity. Root cause diagnosis is not the main focus of this work, since what we analyze are raw datasets, and the root cause for the previous errors was not provided. We cannot assume that various HPC subsystems are perfectly synchronized, and any processing should account for the switching of events within a predefined time frame. Our contributions will apply to other Cray XC40 systems deployed at various HPC facilities worldwide. This will provide insights to system administrators to better perform job failure analysis and to proactively undergo system maintenance to reduce downtime.

In this work, we make the following contributions:

- Accomplishments:
    1. Prediction accuracy of job Exit Status - 92.3%.
    2. Prediction accuracy of job Exit Status in case of a component failure
        - with cabinet/cage/slot information - 88%.
        - without cabinet/cage/slot information - 55%.
    3. Visual representation of prediction results of hardware faults/job failures. Visualization also shows correlations between environment (SEDC) log data and hardware error data with the timelines that are visually synchronized. We demonstrate this capability in our case study.
    4. Implicit feedback mechanism which allows for modifications to the predicted event hardware sequences and the job exit status by domain experts, which is incorporated in our system's future analysis.
    5. The hardware error prediction and two-stage job prediction pipeline in our system identified 92.65% of the correctly-predicted jobs at least 30 minutes before the failure event.

- Steps used to achieve the above accomplishments and useful insights:
    1. In the SEDC/environment log measurements, fluctuations are seen on slot level and also across slots on which jobs are run. (Each slot consists of four nodes.)
    2. Two-stage job prediction pipeline helps filter out jobs that may have fault tolerance built into them. (These will be processed separately). This improves prediction accuracy.
    3. Hardware error data are duplicated for job failure events that occur close to each other. For example, data two minutes before a job starts and two minutes after a job ends are used to process a job. This results in an overlap

of hardware errors across multiple jobs. Once we filtered out the duplicate errors, the prediction accuracy increased from 81.2% to 84.6%.

4. We use recurrent neural network models to predict hardware event sequences and the Topics over Time algorithm to generate the input topic patterns.

5.2 Related Work

To enable prompt evasive actions for failure detection in HPC systems, significant efforts have been directed towards identifying and predicting failures. Survey papers [78,142] on failure prediction give an in-depth idea of current fault prediction methodologies and their shortcomings.

Past efforts to standardize the pre-processing of error logs as a first step to tackling these issues include a three-step pre-processing method (event categorization, event filtering, and causality related filtering) to reduce the size of log data [192], an online clustering algorithm to represent textual and temporal data in the log files succinctly [77], and an online algorithm with clusters that adapt to streaming event patterns [49,51]. We discuss some of the insights gained from data pre-processing in Section *V*.

Machine learning algorithms have been used for in-situ predictions of HPC system faults. The decision tree machine learning algorithm has been used to predict hard disk failures [52,139] from statistically significant/filtered parameters. A context free grammar-based rapid event analysis has been used for online anomaly prediction with lead times of 3 minutes to node failures [26]. The Random Forest machine learning algorithm has been used to predict node [90,122] and job failures [156] with lead times of up to one hour [122]. A two-stage prediction model has been used to predict disk failures [52] or input parameters [17]; the stages serve to filter out relevant data. In our work, we adopt the use of a 2-stage machine learning pipeline for job exit status prediction where the first stage filters down to failed jobs, and the second predicts the exit status of these jobs.

DCDB Wintermute [123] is an online generic framework implemented on top of the Data Center Data Base (DCDB) monitoring system that enables Operational Data Analysis (ODA) [12]. In this work, multiple types of log data, including job, hardware, and environment logs, are analyzed. However, each type of data is analyzed separately, and there is no correspondence in the analysis and visualization of results of these different types of logs. There are a few automated log analysis tools [31, 49, 51, 64, 114, 130, 192] (some of which were discussed in Chapter 3, Sec. 4.1) that use event pattern-based correlations, signal analysis, pattern mining and spatial/temporal event analysis for system failure analysis. There are tools for mining event correlations called LogAider [31] and LogMaster [42] with generic, easy-to-use visualizations. DeepLog [32] is a deep neural network model with stacked Long Short-Term Memory (LSTM) networks, dynamically updating the models to accommodate changing log patterns. Unlike our work, this work does not provide lead time analysis on the predictions. MELA [151] is a visual analytics tool for analyzing diverse log data that tackles issues with scalability and interactivity and is used to visualize and process data with multiple-year logs. A basic log statistical visualization, Blue Gene Navigator [97], provided by IBM, is used by system administrators to monitor their systems. La VALSE [59] has a scalable user interface where large amounts of hardware data in supercomputers are explored. These visualization tools lack the support for scalable and interactive visual analysis of diverse HPC system logs with an online/dynamic prediction engine back-end, which we address in our work.

Analyzing one or two types of log data would not suffice to provide complete insight in comprehending the complexity of large-scale systems. Although several attempts have been made in the past to process multiple types of error log data, most studies avoid using environment logs in their analysis as it can be computationally expensive to process hundreds of measurements per node. We aim to address these shortcomings as described in the following sections.

5.3 System Overview

Fig. 5.1 depicts the architecture of our system with three main sections: databases, back-end processing, and front-end visualization. Our system runs on a machine with two 6-core, 2.4-GHz Intel E5–2620 processors with the Intel Haswell architecture and 384 GB memory per node. The SEDC/environment, hardware error, and job log datasets are 3-4 GB per day, 1-2 GB per month, and a few hundred MB per year, respectively. After the initial data mining step on the hardware error logs, the data size is tens of GBs. Our system is implemented with Python 3.7, a Flask server [57] back-end, and a D3 [11] visualization front-end. For SEDC log storage, we use InfluxDB [76], a database optimized for speedy, high-availability storage, and retrieval of time series data. Since the job and hardware error logs have multiple overlapping measurements recorded at unequal intervals by various components, we store them in an SQL database. A demo video are available online [37].

The next sections present an overview of the various system components and their interactions.

5.3.1 Visual Analytics System

The visualization section is split into three main views: 1) *selection view*, 2) *main view* and 3) *job view*.

The *selection view* consists of 1) a SEDC view, to list measurements in the environment log, 2) a nodes view, to list the nodes used by the selected job, and 3) a recurrent neural network (RNN) prediction view, to list hardware error log predictions. Our system predicts up to 20 hardware errors that are likely to occur within the next few minutes with 81% accuracy.

The *main view* consists of 1) a SEDC cluster view, 2) a hardware topic clouds view, and 3) a hardware topic abstract view. In the SEDC cluster view, SEDC measurements for the selected nodes are clustered using Agglomerative clustering, and the mean function of the time-series data for each cluster is displayed, as shown in Fig. 5.2. The mean functions of each cluster are displayed using separate colors. The user can view the individual measurements by clicking on each measurement in the selection SEDC

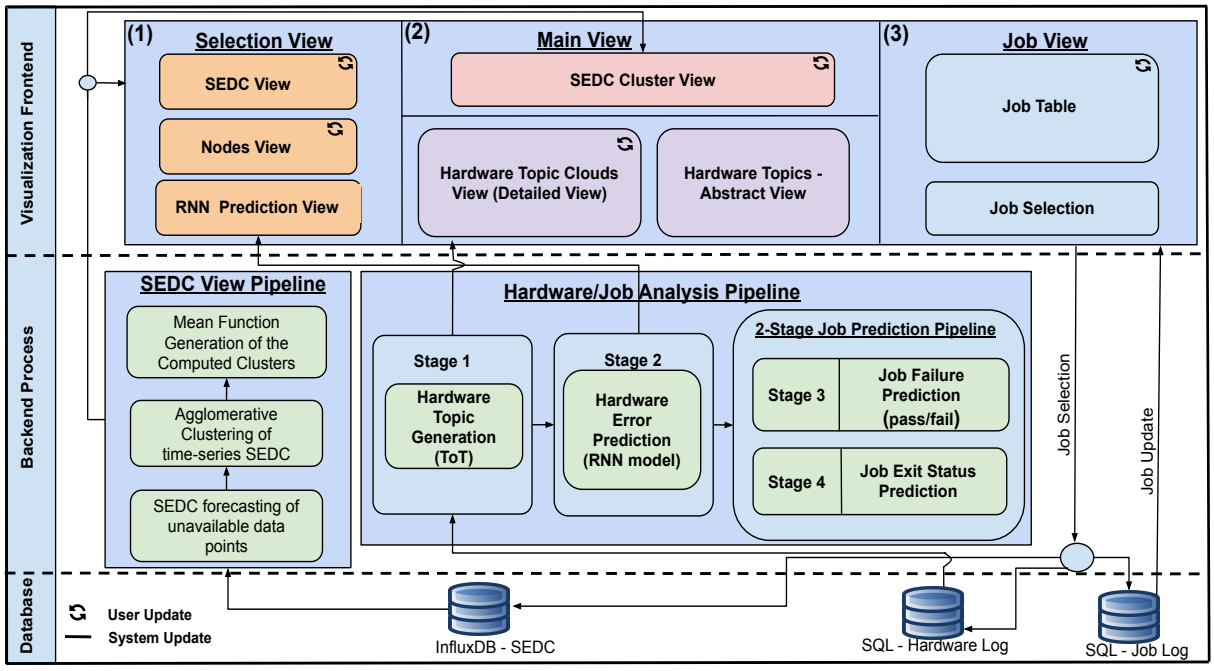

Figure 5.1: Architecture overview of our system for analyzing multi-fidelity HPC system logs. Our system is comprised of the back-end processing pipeline and the visualization front-end. The visualization front-end is split into (1) the selection view, (2) the main view, and (3) the job view. The backend process is split into the SEDC view pipeline and hardware/job analysis pipeline. The hardware/job analysis pipeline consists of 4 stages (1) hardware topic generation, (2) hardware error prediction (3) job failure prediction, and (4) job exit status prediction. Stages 3 and 4 are referred to as 2-stage job prediction pipeline since they handle job failure predictions (© 2022 IEEE).

view, and these measurements are displayed in black.

The hardware topic clouds view is used to visualize the hardware errors grouped into topics using the Topics over Time (ToT) algorithm [176]. This algorithm has shown promising results for error log analysis in the past [25]. Here, the "words" are tokens from hardware error messages. This algorithm identifies and filters topics across time while integrating trends over longer time frames than traditional topic modeling methods such as LDA [10]. Each group of words belonging to one topic at a particular timestamp is displayed using a word-cloud display. The font size of the words corresponds to their influence/score in the topic. The user can filter words by score. The pan and zoom interactions allow for a more comprehensive view of the topic/SEDC trends in the timeline. This view shows how various topics evolve during the job's lifetime. Since we cannot assume that the Cray XC40 subsystems are perfectly

synchronized, we group the hardware error log data in a granularity of 10 seconds.

The hardware topic abstract view displays at most 5 error messages per topic, a maximum of 25 error messages in total, sorted by topic modeling scores. In cases where there are not enough significant error messages assigned per topic, this count is lower. This view consists of two components: (1) the topic view and (2) the dimensionality reduction (DR) view. The user can choose the DR method: PCA [82], UMAP [116], or t-SNE [171]. This DR view gives a 2D representation of the top 23 topics. This view identifies the types of error messages–for example, network-level or node-level–and the nodes on which these errors persist. Once the user learns the error types, the relevant nodes, and the corresponding time range, they can switch to the hardware topic clouds view to further investigate.

The *job view* consists of a table of jobs on the HPC system, either currently running or from the past. The user needs to select a particular job to analyze.

Using the feedback mechanism, the user can update the hardware event sequences predicted by the RNN. These updated sequences are passed through the two-stage job prediction pipeline to predict if the job will pass or fail and the job's most likely exit status. The user is also able to update the job exit status prediction. The two-stage job prediction models are retrained using the updated job exit status. In the next section, we discuss how we process this group of diverse datasets individually and together to feed into the visualization front-end.

5.3.2 Back-end Processing Pipeline

Jobs running on supercomputers routinely run for up to 24 hours on thousands of nodes. Each node has multiple SEDC measurements, reporting data such as power supplies, processors, memory, and fan readings from sensors. A node also reports hardware errors. As the data size increases, so does the computational overhead. Therefore, we designed our system to process one job at a time. When the user selects a job, our system processes the job, SEDC, and hardware error logs for the time frame within which the job is scheduled to run.

5.3.2.1 Data Pre-processing

Environment/SEDC log data contains information from various sensor readings of the HPC system fetched at 10-30 second granularity. This dataset's sheer size poses a challenge during the initial data mining stage and contributes to significant processing overhead. We could have SEDC readings from close to 150 sensors per node. The dataset also has missing entries since the sensors are not immune to failures. To address this problem, we use the `ARIMA` (Autoregressive Integrated Moving Average Model) [70] time-series forecasting method to fill in the missing data, as described in the next section. The hardware error log contains information about hardware failures, categorized as INFO, WARN, or FATAL. INFO messages report the progress of the application at a coarse-grained level. WARN messages highlight conditions that could hamper normal operation. FATAL messages highlight severe errors that could lead the application to fail or abort. In conjunction with job log data, which gives information about system usage by applications, we use this dataset to identify recurring patterns of errors that often lead to failure. We analyze the log data for the year 2018. We split the log data: 48% as a training set, 32% as a validation set, and 20% as a test set.

5.3.2.2 SEDC View Pipeline

The SEDC view pipeline processes the environment log data. The Cray XC40 supercomputer has approximately 700 SEDC readings associated with slots in the supercomputer. Each slot comprises four nodes, and we can filter the measurements per node.

First, we use the `ARIMA` methodology to fill in missing environment log readings. `ARIMA` [70] is a type of statistical model for analyzing and forecasting time-series data. This model aims to describe the autocorrelations in the data. Autoregression (AR) extracts dependencies between current observations and a certain number of lagged observations. Integration (I) is used for making the readings stationary by subtracting observations from the previous observations. Moving Average (MA) extracts dependencies on residual errors of lagged observations and the current observation. Any time-series data that is non-seasonal, not random white noise, and exhibits patterns, can be modeled with `ARIMA` methodology. We use `ARIMA` methodology to forecast miss-

ing time-series values due to a failure in procurement from unforeseen circumstances, including software failures. However, in cases where a node failed or devices were shut down, the readings are assigned a zero. The `ARIMA` model can be written as:

$$y_t = c + \alpha_1 y_{t-1} + ... + \alpha_p y_{t-p} + \beta_1 \epsilon_{t-1} + ... + \beta_q \epsilon_{t-q} + \epsilon_t$$

where y is the differential of the time-series, p is the order of the autoregressive part, and q is the order of the moving average part. If there are over 10 minutes of data readings missing, we assume that the sensor was shut down and assign zero to the missing entries within this duration. Our `ARIMA` implementation uses the Python statsmodels library.

To cluster the SEDC log data, we first filter down to readings from the nodes selected by the user. These readings are collected from 10 minutes before the job started to 10 minutes after the job ended. An evaluation of the job log and hardware log showed that the 10-minute buffer captures sufficient information without adding too much overhead to our system front-end and back-end. Our goal is to cluster readings that follow a similar trend during the job's lifetime. We employ the Agglomerative Hierarchical Clustering (AHC) [1], an algorithm that clusters time-series data incrementally while constructing a hierarchical bottom-up tree-shaped structure of the clusters. AHC is used to group objects into clusters based on their similarity. This helps us track overall trends in SEDC measurements by grouping similar trends together. In the bottom-up computation, each leaf node of the tree is treated as a singleton cluster. Then successive merges of pairs of clusters lead to a final single cluster containing all readings. The readings in each cluster are then represented in the visualization by their mean function. We tune the clustering algorithm to pick up to five clusters, which sufficiently captures the overall trends without causing too much visual clutter. Our interactive visualization allows the user to view individual SEDC readings as well. The next three sections describe the four stages in the hardware/job analysis pipeline.

5.3.2.3 Hardware Topic Generation

Stage 1 in the hardware/job analysis pipeline includes using the Topics Over Time algorithm [176] to capture the error messages and group them into topics. Hardware

error logs have duplicated events, which we combine or filter out during the pre-processing stage. However, visualizing hardware error messages is challenging because of the data size from numerous errors reported by job nodes. Therefore, we employ the Topics over Time (ToT) algorithm [176], an LDA-style topic model, for modeling localization and word co-occurrences in time. The model captures word associations with topics that are localized in time. This model is shown to capture the structure or weights of the words in topics and how these structures change over time. Each of the words belonging to a topic identified by the ToT model is then displayed using a word cloud in the hardware topics clouds view. Words with more influence on the topic are displayed in larger font sizes. This helps the user view how error messages evolve as the job approaches completion. The user can filter the messages by ToT topic scores and nodes. Although ToT models could also be used to predict incoming hardware event sequences, we choose RNNs for this task because the predictions are roughly two times faster and provide about the same accuracy.

5.3.2.4 Hardware Error Prediction

In stage 2 of the hardware/job analysis pipeline, we perform hardware event sequence forecasting with recurrent neural networks (RNNs). RNNs [184] were designed for sequence prediction problems. Recent developments in text analysis, with the help of deep learning, provide state-of-the-art performance and good accuracy in prediction in text-based data. Simple language models try to predict a probability for the next word given the current word. However, RNN language models can capture the entire context of the input sequence to improve predictions. RNNs can be programmed to use an internal state (memory) to process input sequences with a variable length. RNNs have been used in the past for system log anomaly detection [13] to model the normal distribution of system events in system logs and identify complex relationships. A Long Short Term Memory (LSTM) layer, which is a type of RNN layer, helps prevent back-propagated errors from vanishing or exploding (the "vanishing gradient problem" [71]). Therefore, it is possible to remember information for long periods of time. Regardless of the distance between context resets in event sequences, an LSTM handles

the problem of keeping or resetting context across temporal event sequences, which a traditional RNN fails to do.

We use an RNN to predict the next sequence of error events for a job. The inputs are the error messages and error categories from past error events. We use the Keras [36] library and the Adam optimizer. Our RNN model contains an LSTM layer with 64 units, tanh activation, and dropout set to 0.1; a dense layer with 64 units and the ReLU activation; a dropout regularization layer set to 0.5; and a dense output layer of width 100 with softmax activation. Our system uses the sequence of hardware error events ten minutes before the current timestamp to predict up to 20 hardware events that are likely to occur soon. We then feed these predicted events and the original sequence into the two-stage job prediction pipeline. Using our RNN model to predict future events at the component level, a user can identify components that are likely to fail and the type of failure. The feedback mechanism allows the user to update the predicted results, which our system then uses for future predictions.

5.3.2.5 Two-Stage Job Prediction Pipeline

Stages 3 and 4 in the hardware/job analysis pipeline form the two-stage job prediction pipeline. They predict job failures and the exit status of the failed jobs.

Stage 3 uses a random forest classifier to predict if a job will successfully finish or fail. The dataset used in stage 3 contains consolidated hardware and job log data from the selected time frames. The jobs predicted to fail proceed to the next stage, job exit status prediction.

In stage 4, we use a random forest classifier to predict each job's exit status. With the help of the predicted exit status the users can identify if the job will fail through user termination, network error, user errors, memory errors, or signal termination. A random forest classifier fits many decision tree classifiers on various data subsets to avoid reducing the test prediction accuracy due to over-fitting. Our random forest classifier is trained using 100 decision trees, a maximum tree depth of 8, the Gini function to measure the quality of a split, and a minimum number of splits of 2. We chose these hyperparameters using cross-validation.

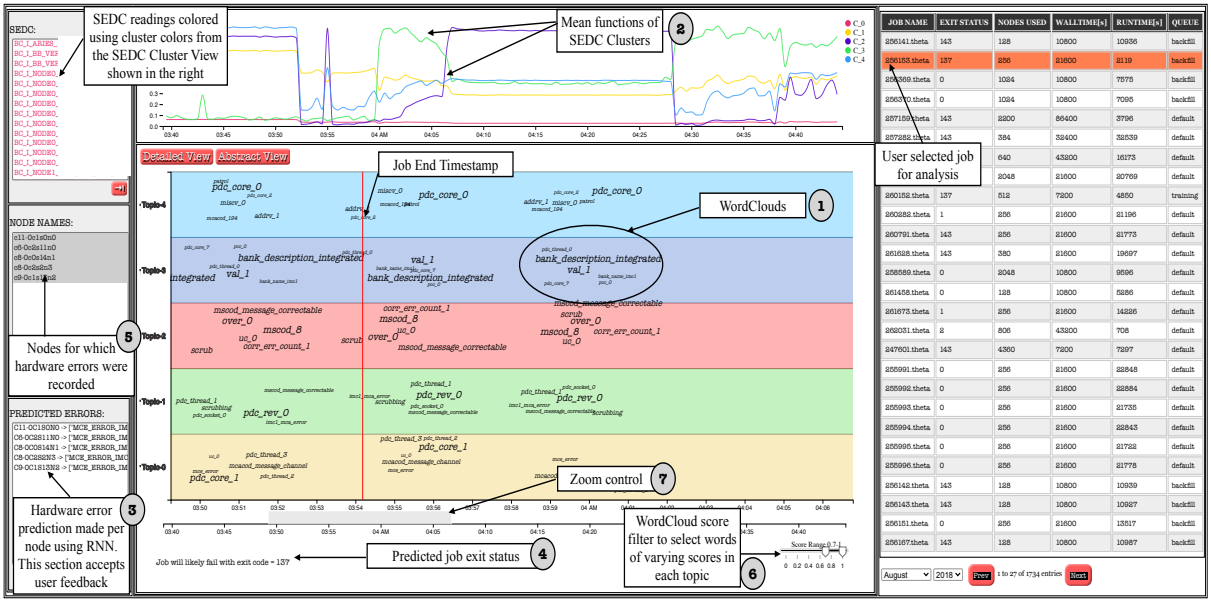

Figure 5.2: The figure shows the visualization front-end of the system. (1) ToT word clouds show the evolution or lack of change in the hardware error topics that have occurred in the selected nodes on which the job is running. (2) The plot shows the mean functions of the SEDC clusters. In total, 570 measurements have been grouped via Agglomerative Clustering in this example. (3) The hardware error RNN predictions and (4) two-stage job exit status predictions are shown on the left. (5) Nodes view showing nodes which reported hardware errors. (6) Word cloud score filter to select words of varying scores in each topic. (7) Synchronized timeline control bar.

5.4 Case Studies

We describe two detailed cases of job failures and present how our system aids in the identification of failure events from the large-scale and diverse logs. The case studies illustrate how to interpret the different layouts in the visualization and compare the processed results from each data type. In the second case study, we describe some of the user interaction mechanisms available in our system.

5.4.1 Case Study 1:

Fig. 5.2 depicts the hardware topic clouds view, using word clouds (Fig. 5.2①), for the case study 1 job. This job was expected to run for 6 hours on 256 nodes but ended at approximately 50 minutes with exit status 137. Using the selection view, we can identify the nodes where hardware errors have occurred (Fig. 5.2⑤). Out of the 256 nodes used by the job, there are 5 with hardware errors. Upon selecting these 5 nodes, the word

cloud view updates, showing the summarized distribution of the hardware error topics. The zoomed view of the word cloud (Fig. 5.2⑦) makes the error distribution over time more intelligible. We can investigate hardware errors using the word cloud view by zooming in further and adjusting the score range scale (Fig. 5.2⑥). We see that the nodes have encountered multiple machine check exception (MCE) errors. The field val_1 (Fig. 5.2①) means that a valid error has occurred. The UC set to 0 (shown in Topic-2) indicates that the processor fixed the error condition. The PCC bit set to zero indicates that the error condition did not corrupt the processor's state. However, this error summary is for all 5 selected nodes. We can select individual nodes (Fig. 5.2⑤) to view the summary of hardware errors at that precision. The environment log (SEDC) measurements are read for the selected nodes. We use Agglomerative Clustering to cluster time-series data for the selected job duration into groups of high similarity. In this example, we read 570 SEDC measurements and compute the mean function for each cluster. The resulting cluster means are shown in the SEDC cluster view (Fig. 5.2②). We see larger variations in clusters 2, 3 and 4. The RNN model is then used to predict the hardware errors likely to occur soon. In this example, the RNN predicts that the next 10 hardware errors are likely to be Machine Check Exception (MCE), likely caused by memory issues. Using up to 20 predicted hardware errors along with hardware errors that have already occurred during the job, we predict the job's exit status using the two-stage job prediction pipeline. The predicted job status is then shown in the visualization (Fig. 5.2④). The user can update this view if the prediction is wrong. In this example, 22 minutes before the actual failure, our system predicts that the job is likely to fail with exit status 137. This should provide sufficient lead time for proactive actions to prevent a potentially catastrophic event.

5.4.2 Case Study 2:

Fig. 5.3 shows the SEDC cluster view for the nodes of a selected job. This job was expected to run for 10 hours on 648 nodes but ended at approximately 9 hours with exit status 143. In this example, the job prediction pipeline uses the input from the RNN model, which predicts the hardware errors likely to occur in the near future. The job

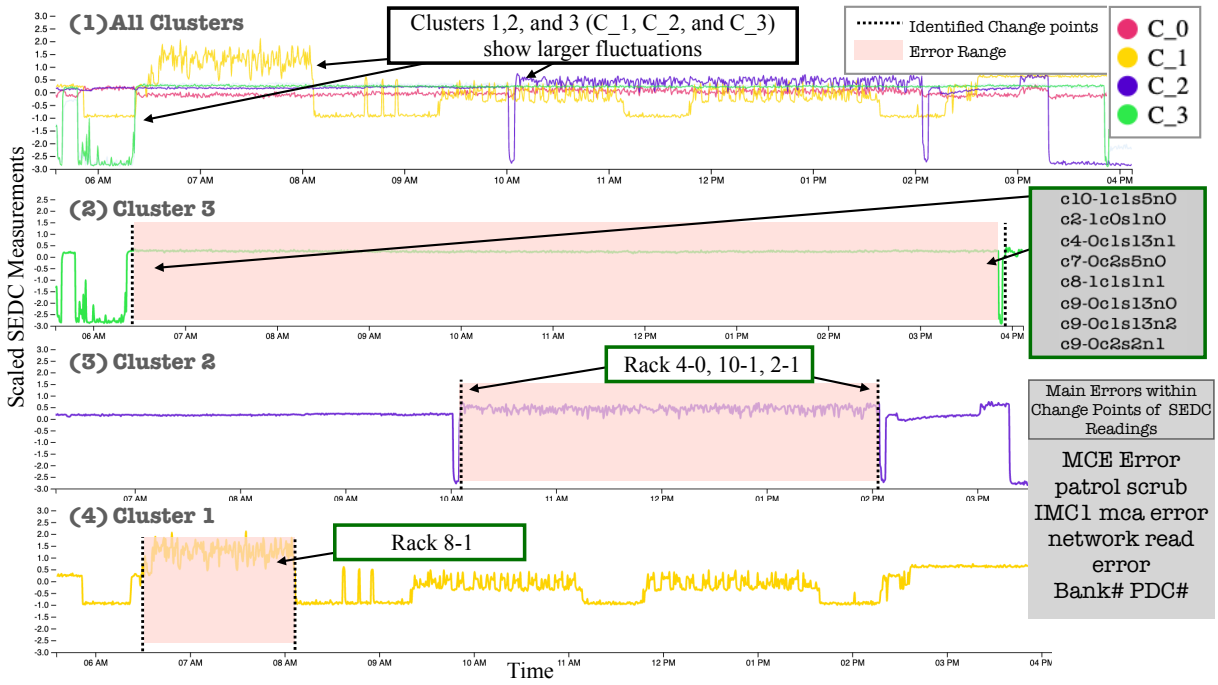

Figure 5.3: The figure shows the SEDC view for the nodes of a selected job. (1) shows the mean functions of all the clustered SEDC readings, (2)-(4) show the mean functions of the SEDC readings for clusters 3, 2, and 1 (C_3, C_2, C_1), respectively. The change points identified are shown in dotted black lines in (2)-(4). The cluster C_3 include readings pertaining to nodes that have machine check exception (MCE) errors with correctable memory errors in the error range identified by the change points. The identified change points within the SEDC readings give the time-ranges of interest which are shown in red rectangular overlay. In the figure the nodes and sets of nodes (racks) are shown in the green outlined box. In (2), the nodes with hardware errors show similar trends for the duration of the errors. Whereas, (3) & (4) show behavior of sets of nodes used by the job that did not report any hardware error (© 2022 IEEE).

prediction pipeline predicts that the job will likely fail with exit status 143, 32 minutes before the actual failure. This provides sufficient lead time for proactive actions and further investigation on problematic nodes to prevent potentially catastrophic events in the near future. In Fig. 5.3, the readings showing current (unit in amperes) are normalized between values −3 and 3. The identified change points (shown in vertical dashed black lines), help in filtering the time range (shown in red overlay) of the job run when these nodes reported the hardware errors. In Fig. 5.3①, the mean function of the readings belonging to the clusters C_1, C_2, and C_3 show larger fluctuations for the duration of the job run. By analyzing trends of SEDC readings belonging to individual clusters, a user can identify similar behaviors of individual nodes or sets of

nodes, i.e., racks. For example, in Fig. 5.3②, cluster 3 (C_3) shows peaks and troughs at specific time points (shown in vertical dashed black lines). This cluster contained SEDC readings from all the nodes reporting hardware errors for the selected job. Fig. 5.3③-④, capture the mean behavior of nodes from racks used by the job that reported *no* hardware errors. Here, the error range in each figure (Fig. 5.3③, 5.3④) isolates the region of hardware errors at nodes (in the same rack) that reported hardware errors, i.e., Fig. 5.3②. For example, in Fig. 5.3③, job nodes with no errors in racks 4-0, 10-1, and 2-1 show error regions for job nodes in the same racks (c4-0c1s13n1, c10-1c1s5n0, and c2-1c0s1n0) with trends shown in Fig. 5.3②. Similarly, Fig. 5.3④ show error region for job node c8-1c1s1n1 (Fig. 5.3②). Although not all time ranges identified by the change point detection [165] algorithm are error regions, the method gives an intuition of interesting time ranges for a long-running job and helps narrow down which nodes may be problematic.

5.4.3 Case Study 3:

Fig. 5.4 shows the abstract view of the hardware error messages. This view has two components: (1) the topic view and (2) the dimensionality reduction (DR) view. The user can use this abstraction of the errors to identify the error types, for example, if the error is at the node or network level. The topic view ① shows the top 23 error messages on the vertical axis and the topic categories on the horizontal axis for the selected job, nodes, and time range. Users can select clusters in the DR view ②, which highlights the corresponding error messages. In Fig. 5.4②, the points belonging to Topic-2 are selected, and the highlighted error messages in Fig. 5.4① are lcb (Link Control Block) errors. Points in topics 0 and 4 are closer to each other in the DR view, suggesting that these errors are related. This is evident in the topic view (①,③), since these topics share node-level errors, scrubbing, patrol etc., which are related to memory errors. Selecting points from Topic-4 in Fig. 5.4④ highlights node-level errors. Upon identifying the error types and the corresponding nodes and time range, the user can switch to the detailed view to inspect the error messages in more detail. Fig. 5.5 shows the hardware topic clouds view, using word clouds for a job that failed with exit status 1. The job ran

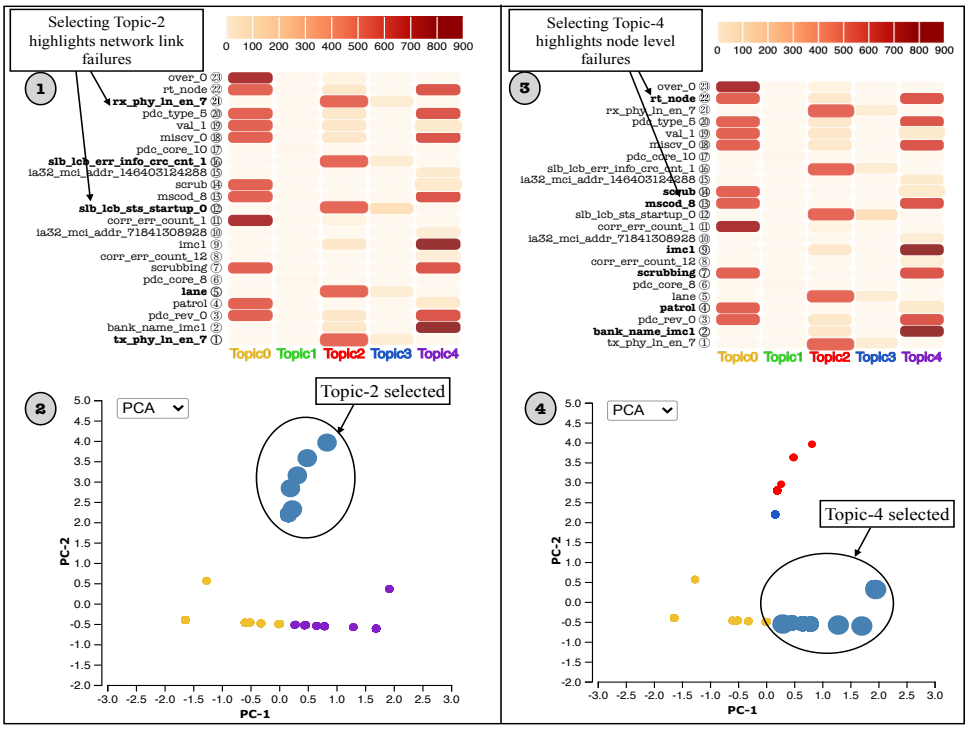

Figure 5.4: The figure shows the abstract view of the hardware error messages in a job. This view consists of two components, (1)(3) the topic view and, (2)(4) the dimensionality reduction view, (PCA, UMAP, t-SNE) (© 2022 IEEE).

on 256 nodes for a runtime of about 6 hours. The selection view shows that 8 of the 256 nodes have hardware errors. Upon selecting these 8 nodes, the word cloud view updates to show the distribution of hardware error topics. As we zoom in further on the word cloud view, we find that the nodes have encountered multiple machine check exception (MCE) errors with *val_1* showing that a valid error has occurred. The word cloud visualization also shows that there has been patrol *scrubbing*, a variant of memory scrubbing, throughout the job's duration. Memory scrubbing corrects bit errors and as it consists of read and write operations, it is known to increase power consumption. This increase in power consumption is shown in the SEDC cluster view (Fig. 5.5②) from 7:00 am to 11:00 am.

Fig. 5.6 shows the SEDC cluster view. In this example the readings showing current (I, unit in amperes) are normalized between values −3 and 3. In Fig. 5.6①, the mean function of the clusters C_2, C_3, and C_4 show larger fluctuations for the duration of the job run. In the word cloud view, a user is able to view all the errors for a user-selected

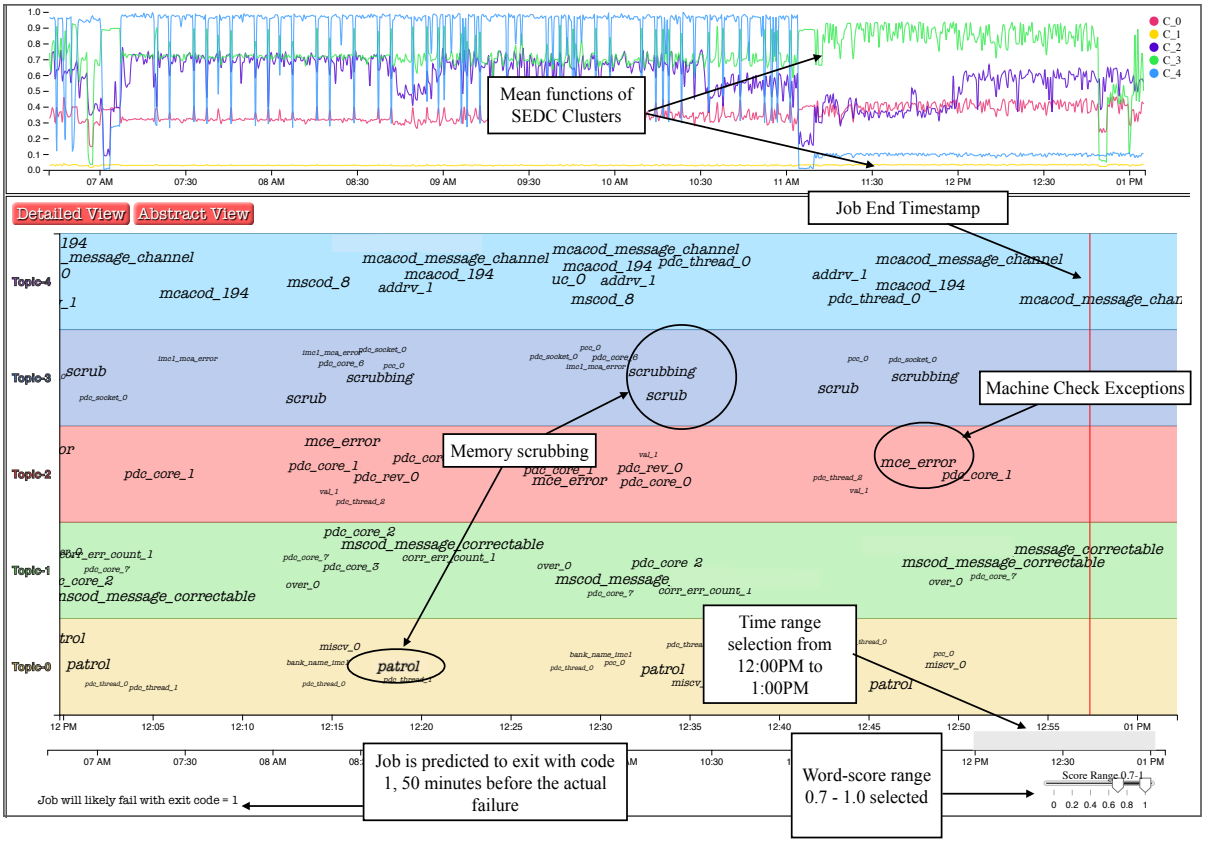

Figure 5.5: The figure shows the visualization front-end of the system. (1) ToT word clouds show the evolution or lack of change in the hardware error topics that have occurred in the selected nodes on which the job is running. (2) The timeline plot shows the means of the SEDC measurement clusters. In total, 683 measurements have been grouped via Agglomerative Clustering in this example. (3) The hardware error RNN predictions and two-stage job exit status predictions are shown on the right. This view shows the word clouds filtered by word score (6) to avoid clutter.

set of nodes and time duration. However, this view alone does not provide details about time durations or specific time points at which the individual nodes report the errors. By analyzing trends of SEDC readings belonging to individual clusters, a user can identify behaviors of individual nodes or sets of nodes. For example, in Fig. 5.6③, the cluster 4 (C_4) shows peaks and troughs at specific time points (shown in vertical dashed black lines). This cluster contained SEDC readings from node $c0$-$0c2s5n3$. Using these time points, identified by the change point detection [165] algorithm, a user can identify the errors reported by this node. Filtering the word cloud data for this node (using the nodes view in the selection view) showed that the node reported errors at time points identified using the SEDC cluster view and in the duration from 10 : 22AM - 10 : 57AM

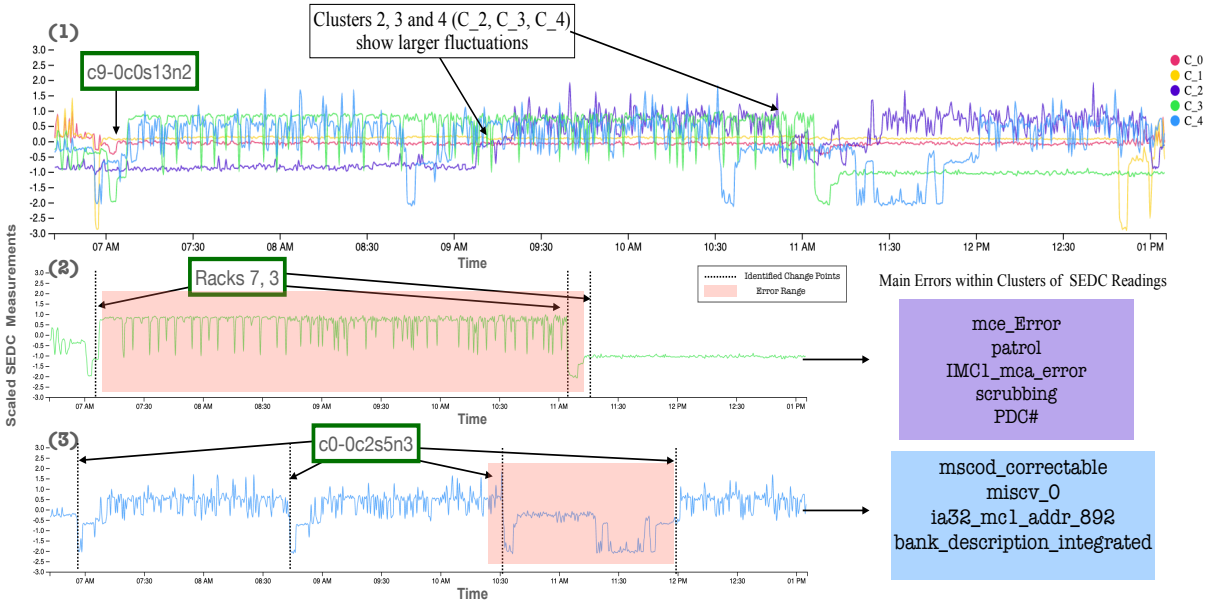

Figure 5.6: The figure shows the SEDC Cluster view. (1) shows the mean functions of all the clustered SEDC readings, (2) shows the mean function of the SEDC readings for cluster 3 (C_3), and (3) shows the mean function of the SEDC readings for cluster 4 (C_4). The change points identified are shown in dotted black lines in (2) and (3). These change points give the time-ranges of interest, which are shown in red rectangular overlay. The nodes or groups of nodes (racks) are shown in the green outlined box (© 2022 IEEE).

when the SEDC measurements show a drop in value. In Fig. 5.6②, nodes used by the job on racks 7 and 3 show similar SEDC trends. The identified change points (shown in vertical dashed black lines), again help in filtering the time range (shown in red overlay) of the job run when these nodes reported the hardware errors. Therefore, the SEDC cluster view helps filter specific temporal regions for which a node or groups of nodes behave uniquely. Thus, it helps augment the user's perception of errors laid out within the system. Some clusters also show SEDC readings measured at the rack level (each rack contains 192 nodes). When rack-level readings and node-level readings are clustered together, it shows that the job is affecting readings at the rack level even if the job doesn't utilize the entire rack.

The RNN model is then used to predict the hardware errors likely to occur soon. In this example, the RNN predicts that the next 10 hardware errors are likely to be Machine Check Exception (MCE) errors. Using this information and the hardware errors that have already occurred during the job run, we predict the exit status of the

job using the two-stage job prediction pipeline. In this example, our system predicts job failure with exit code 1, 52 minutes before the actual failure.

5.5 System Model Analysis

This section describes the prediction results, lead time, and some insights from our analysis. We analyze 91,217 jobs from 2018-2019. The error categories in the dataset can be broadly classified as informational, memory, transaction, and transient errors. The type of components that incur errors are either node or network (link control buffer, network interconnect etc.). The combined hardware job log data is split with 48% as a training set, 32% as a validation set, and 20% as a test set. To assess the quality of the prediction methods we use three metrics, precision (*pr*), recall (*rec*), and accuracy (*acc*):

$$pr = \frac{TP}{TP+FP} \quad rec = \frac{TP}{TP+FN}$$

$$acc = \frac{TP+TN}{TP+FP+FN+TN}$$

where TP is the number of true positives, i.e., the number of correctly predicted failures; FP is the number of false positives, i.e., the number of jobs wrongly predicted as failures; FN is the number of false negatives; and TN is the number of true negatives. Precision is the fraction of the jobs that were predicted as job failures that were correct. Recall is the fraction of the job failures that were correctly predicted. Accuracy is the ratio of correct predictions to the total number of predictions. Usually, an increase in precision leads to a decrease in recall and vice versa. Table 5.2 summarizes the accuracies of each stage in the hardware/job analysis pipeline.

Insight 1 - ***Capturing jobs with built-in fault tolerance is challenging***: Stage 3, the part of the job prediction pipeline that predicts if a job will fail, filters out jobs that may have built-in fault tolerance by predicting that they will succeed, which improves the prediction accuracy of Stage 4 (job exit status prediction). Jobs with built-in resilience may still exit with success (exit status = 0). This behavior is difficult to model and predict since these jobs process to completion despite anomalous behavior in the HPC system. Our proposed course of action is to collect more fine-grained data, including console

logs and syslogs. This would help build models with more detailed information about these jobs and their mechanisms to offload tasks when encountering system anomalies.

Insight 2 - ***Jobs with custom exit codes and stalled jobs need to be handled separately***: The jobs that are predicted to fail in stage 3 are then passed to stage 4, which predicts the job exit status. This stage has an accuracy of 92.3%. If all jobs, including those predicted to succeed, are passed on to stage 4, the accuracy drops to 85.1%. One reason is that stage 3 of the pipeline helps filter out jobs with built-in fault tolerance, jobs with custom exit status assigned to them, and jobs that have stalled. These jobs need to be separately handled. Part of our future work will include identifying such jobs and building a workflow to model their behavior. In our dataset an estimated 30% of the jobs could have fault tolerance built into their workflow. However, further investigation is needed here.

Insight 3 - ***Including data from multiple sources along with cabinet/cage/slot information improves the prediction accuracy***: Here, we discuss some data pre-processing insights that improved the prediction accuracy. Hardware error log data is duplicated for job failure events that occur close to each other; we consolidate or remove such repeated events if they occur within a predefined time-frame. The pre-processing of hardware error data for hardware event sequence prediction using the RNN model includes filtering duplicate events. Certain error messages occur very frequently and continually; we only use these events at predefined intervals. Job log data and hardware error log data are combined to form an input dataset for the two-stage job prediction pipeline. Hardware events occurring close to a job, i.e., 30 seconds - 2 minutes before or after, are added to that job in the combined dataset. This technique boosted the prediction accuracy from 81.2% to 84.6%. Nodes in HPC systems are often categorized as service or compute nodes. We processed the log data through the machine learning pipeline to identify how these categories affect the prediction accuracy. The job exit status prediction without cabinet/cage/slot information was 55% accurate. When we included cabinet/cage/slot information, the accuracy increased to 88%. This boost in prediction accuracy shows that each node still needs to be treated as an individual

entity, even in a homogeneous set of nodes. This method significantly increases the size of the dataset.

Insight 4 - ***Including environment logs in the analysis could help pinpoint anomalous system behaviors***: Steep fluctuations, such as large temperature fluctuations, can degrade disk and network performance. This could result in transient errors and device performance fluctuating between normal conditions and degradation. With the SEDC cluster view, we can visualize these measurement fluctuations. Fluctuations in SEDC readings are seen at the slot level and across slots on which jobs are running. This means that the processing of other jobs in the temporal and spatial vicinity could be affected. When we encountered peculiar fluctuations in temperature measurements, we investigated further. We found cases where nodes in the surrounding slots showed a continued increase in temperature even after the initial job had stopped. Such cases need to be investigated further. Although not frequent, do cases like these affect jobs running on nodes in the neighboring slot? Our visualization provides a high-level glimpse of such cases that are otherwise overlooked.

Table 5.1: Prediction results for stage 3, the job failure prediction stage, using various machine learning models. The random forest model output is passed onto stage 4, the job exit status prediction stage, and this gives an accuracy of 92.3% (© 2022 IEEE).

Model	**Accuracy**	**TP**	**TN**	**FP**	**FN**
Random Forest	65.4%	107250	9766	52820	4930
Decision Tree	58.2%	88620	7080	57320	11980
Neural Network	50.6%	75140	7360	68476	14024

Table 5.1 gives the confusion matrix with accuracy values for stage 3, the job failure prediction stage for the combined job-hardware log. We reach an accuracy of 65.4% for the Random Forest machine learning model with a precision of ~64% and a recall of ~95%.

Table 5.2: Prediction results for stages in hardware/job analysis pipeline (© 2022 IEEE).

Stages	**Description**	**Model**	**Accuracy**
Stage 2	Hardware error prediction	RNN	81.3%
Stage 3	Job failure prediction	Random Forest	65.4%
Stage 4	Job exit status prediction	Random Forest	92.3%

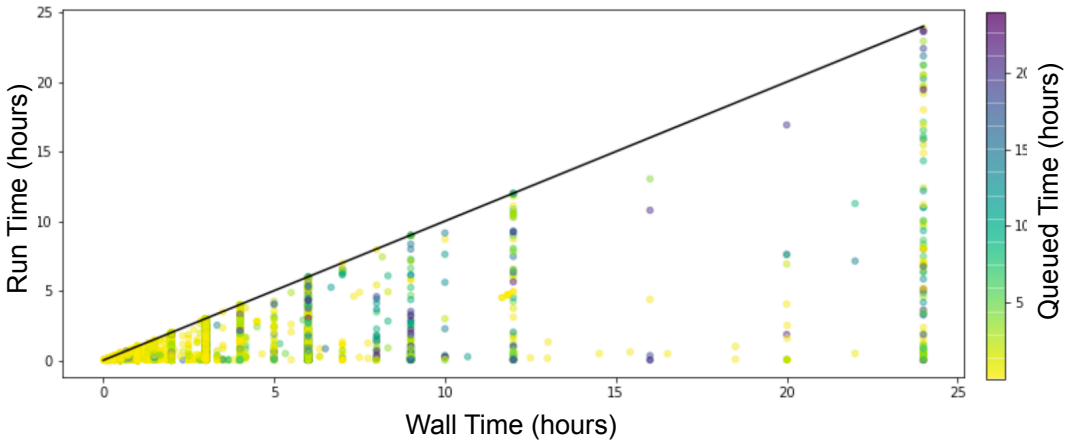

Figure 5.7: The figure shows the job execution time (runtime) versus the total time requested for a job (wall-time) in hours, colored by the time the job spent in the queue, queued wait time, for failed jobs.

Fig. 5.7 shows the job execution time (runtime) versus the total time requested for a job (wall-time) in hours. These jobs failed to execute successfully. The black line represents time when the runtime is equal to the wall-time, at which time the jobs terminate. The colors represent the time the jobs spent waiting in the queue before the resource allocation. Here we see that the shorter running jobs do not spend too much time in the queue as per the job scheduling policy. However, we also see jobs that spend well over five hours in the queue and fail to execute successfully, with some jobs having the runtime equal to the wall-time. In some cases these runtimes are close to 24 hours. Hence, early error detection to avoid such cases becomes necessary.

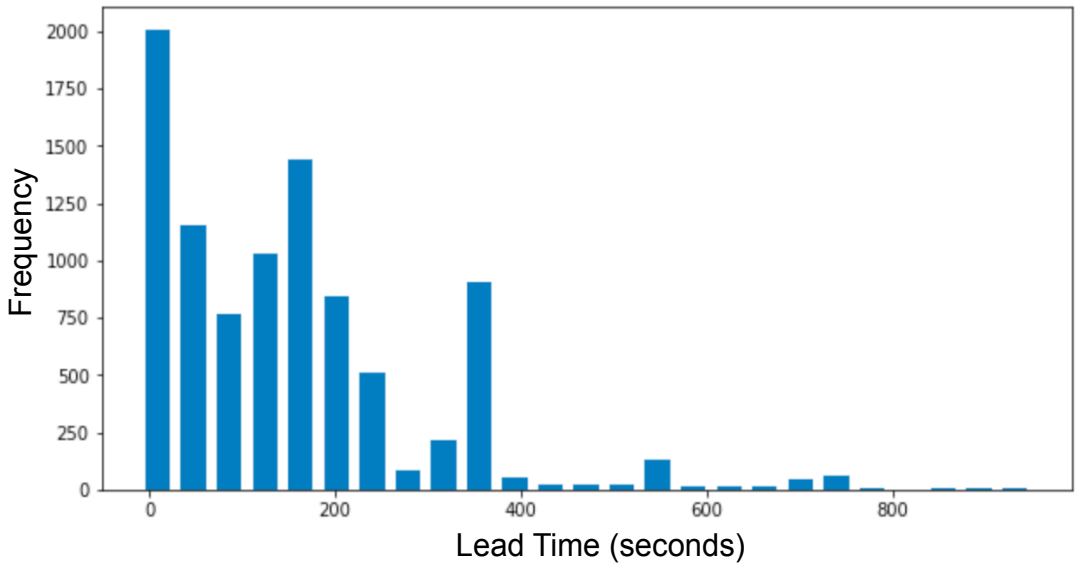

Figure 5.8: The figure shows the lead time of job log exit status prediction for $9,456$ jobs of the test set. The histogram shows the distribution of the number of correct predictions in the two-stage job prediction pipeline in terms of minutes ahead of the job termination (© 2022 IEEE).

Table 5.3: Description of Job Exit Code/Status (© 2022 IEEE)

Signal	**Value**	**Meaning**
SIGHUP	1	Catchall for general errors or death of controlling process
SIGINT	2	Misuse of shell built-ins or Interrupt from keyboard
SIGILL	132	Illegal Instruction
SIGABRT	134	Abort signal from abort(3)
SIGKILL	137	Kill signal
SIGTERM	143	Termination signal; killed by software or OS
SIGHUP	127	“command not found"

Figure 5.8 shows the distribution of the lead time of correct exit status predictions in terms of minutes ahead of the job termination. The results show that only 591

(7.34%) of the predictions occur within 30 minutes of the actual failures, and a total of 8761 (92.65%) of the predictions occur further in advance. This provides a reasonable amount of time for corrective measures to handle errors that cause an anomalous job exit. The trained RNN model provides an accuracy of 81.3%. The hardware error prediction stage gives us information about the types of hardware event sequences more likely to occur at the hardware component level. Fortunately, the addition of the RNN model in the pipeline does not reduce the lead time and in 1.2%–2% of cases the lead time increases. As an improvement we plan to use additional logs like console logs, syslogs etc. to get fine-grained error details to improve the trained RNN model.

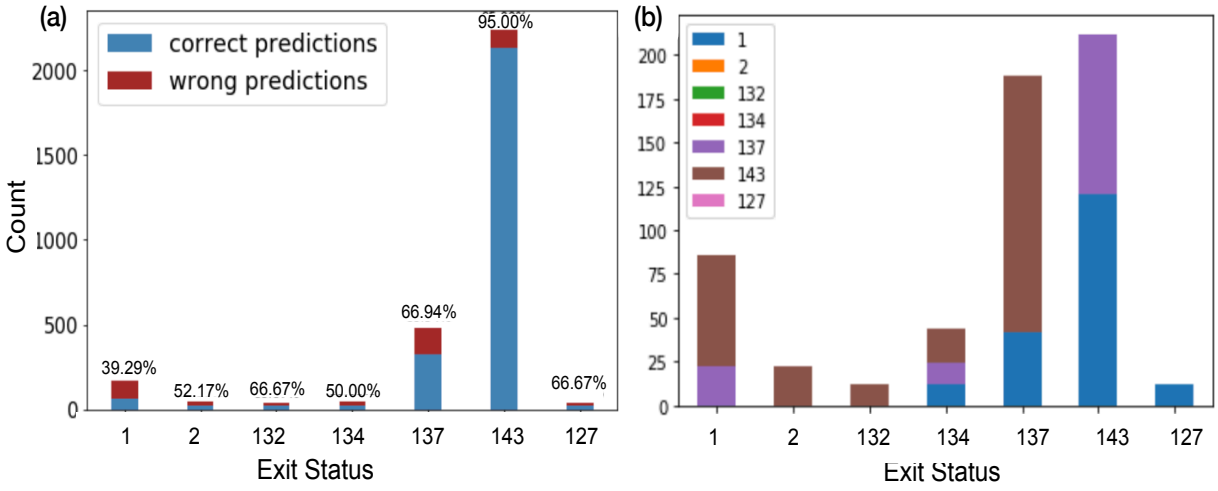

Figure 5.9: The figure (a) shows the number of correct and incorrect predictions per job exit status–annotated with percentage of correct predictions–for 3152 jobs for the selected job exit codes included in our test dataset, and (b) shows the job exit status predictions made by the model for cases in which the predictions were incorrect. Most of the predictions incorrectly assign the exit status of 1, 143 or 137 to jobs. Exit codes 137, 143, and 1 are usually associated with kill signals, and this could lead to some confusion in prediction (© 2022 IEEE).

The distribution of jobs per exit status is shown in Figure 5.9(a), along with the breakdown for each category between those correctly labeled and incorrectly labeled. For convenience, Table 5.3 lists common job exit statuses and a brief description for each exit code. The most common non-zero exit status in the log dataset is 143. Our model can label 92% of jobs with the correct exit status.

Figure 5.9(b) shows the incorrect model predictions of job exit status. Here we have selected the most commonly occurring exit codes, since the exit codes can range

from 0-255. We can see that most of the incorrect predictions assign the exit status of 1, 143 or 137 to jobs. The predictions with exit code 143 and 137, which signify a kill or termination signal, are usually confused by the model. We posit that these errors could be due to users providing their own exit codes when their application exits unexpectedly, the job timing out, or errors introduced by incorrect predictions by the RNN model from Stage 2.

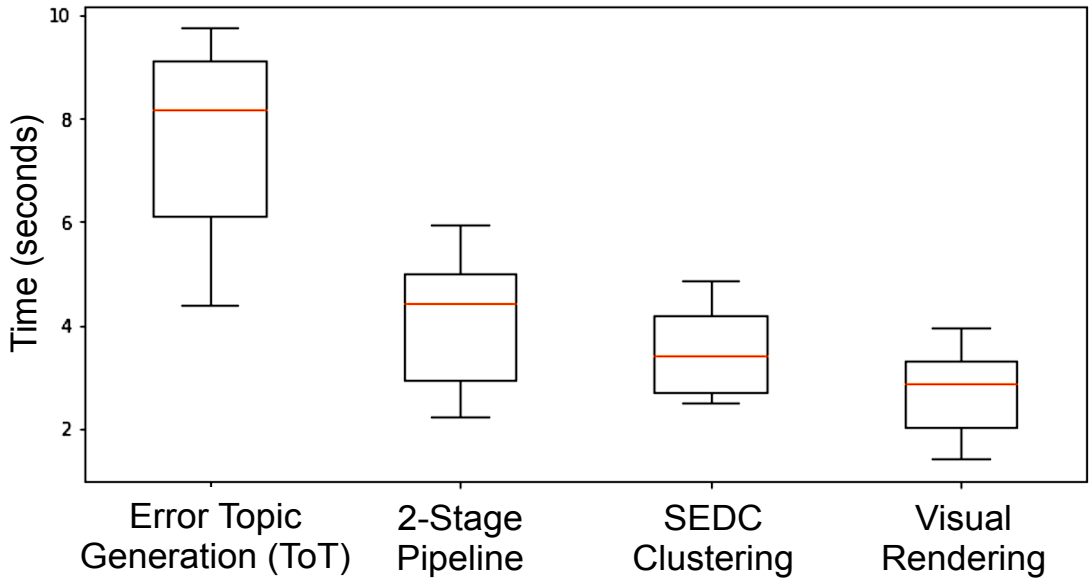

Figure 5.10: The figure shows the box plot of the user wait time in seconds for back-end and front-end computations to complete. The computation times in the analysis pipeline of our system can be split into sections: 1) error topic generation (ToT), 2) two-stage job prediction pipeline, 3) SEDC clustering, and 4) visualization rendering (© 2022 IEEE).

Figure 5.10 shows the box plots for the various computation costs in our system pipeline. We have identified four groups of computation times that contribute to the overall user wait time. These are error topic generation using ToT, the two-stage job prediction pipeline, SEDC clustering, and the visualization rendering. The topic generation (ToT) computations take up to 10 seconds of wait time. The wait time depends on parameters such as number of iterations, topics, and nodes. The two-stage job prediction pipeline and SEDC clustering result in at most 6 seconds of wait time each. The visual rendering is done asynchronously, and different parts of the visualization render independently on the screen. Therefore, there is not much overhead, i.e. up to 4 seconds, in the visualization when compared to the time spent in the other parts of

our system.

Since our machine learning models do not make perfect predictions, our system provides an option for feedback from users, such as the system administrators. The user in the loop is able to make changes and updates to the output which is then used as a feedback mechanism to update the model to be used in the future. We allow the user to update hardware event sequences predicted by RNN. The user-updated sequences are then passed through the two-stage job prediction pipeline to predict if the job passes or fails and the most likely job exit status. The user can also update the job exit status predicted by our system. The two-stage job prediction models are retrained using the updated information while they are also check-pointed separately.

5.6 Conclusion

In this work, we share insights gained from data processing and analysis which could be extended to be used with similar datasets. Our visual analytics tool analyzes the job logs and helps glean insights from their correspondence with the other logs (i.e., hardware error logs, environment logs, etc.) at various temporal and spatial resolution.

In the job prediction pipeline, we have an accuracy of 65.4% for predicting if a job will fail and an accuracy of 92.3% for predicting the job exit status. Predicting whether jobs with built-in fault tolerance would fail is a challenging task and the information in our dataset is not sufficient to make more accurate predictions. The RNN model in the hardware error prediction stage helps in identifying the next 10-20 hardware error patterns in the sequence (the number can be customized), and the model accuracy is 81.3%. Including the hardware error prediction stage improves the lead time in 1.2%−2% of the test cases and does not affect the lead times for the rest. The hardware error prediction and two-stage job prediction pipeline in our system identified 92.65% of the correctly-predicted jobs at least 30 minutes before failure. We analyze SEDC/environment logs, which are usually ignored during error log analysis due to their size, which can cross the gigabyte - terabyte range in a few weeks. Using these logs could help pinpoint faulty nodes which can be investigated using our tool. To

counter any erroneous predictions made by our system, we have a feedback mechanism that helps filter and update information which would help improve the deduction of error patterns in the data.

Our next goal is to incorporate root cause analysis in our visual analytics system with the help of additional logs such as syslogs and console logs.

Chapter 6
A Visual Analytics Approach for Multi-Scale Systemic Assessment of Multifidelity High Performance Computing Systems

6.1 Introduction

Maintenance and monitoring of supercomputer systems are crucial steps for facilitating their robustness and reliability, leading to advancements in numerous fields of scientific research. As supercomputing systems are growing in complexity with changing execution environments, increasing computation capabilities, and increasing frequency of updates and upgrades, the monitoring data collected at multiple fidelity levels and varying temporal resolutions is also increasing. Furthermore, high utilization of these systems is expected to accommodate applications and jobs that sometimes execute on these systems for weeks. Hence, any system errors and failures could compromise the integrity of the results or cause job failures, all of which lead to significant overhead in the already computationally and financially expensive research and development. Therefore, we study these diverse systems and subsystems' log data and use the knowledge acquired to derive patterns of behaviors that could help understand the underlying state of the system at any given time. In this work, we focus our efforts on the Cray XC40 system, specifically the Theta supercomputing system deployed at ALCF [5]. Theta is an 11.69 Petaflops system that was designed in collaboration with Intel and Cray. The nodes are interconnected with an Aries interconnect in a Dragonfly topology. We built a holistic analytical system that processes the massive log data,

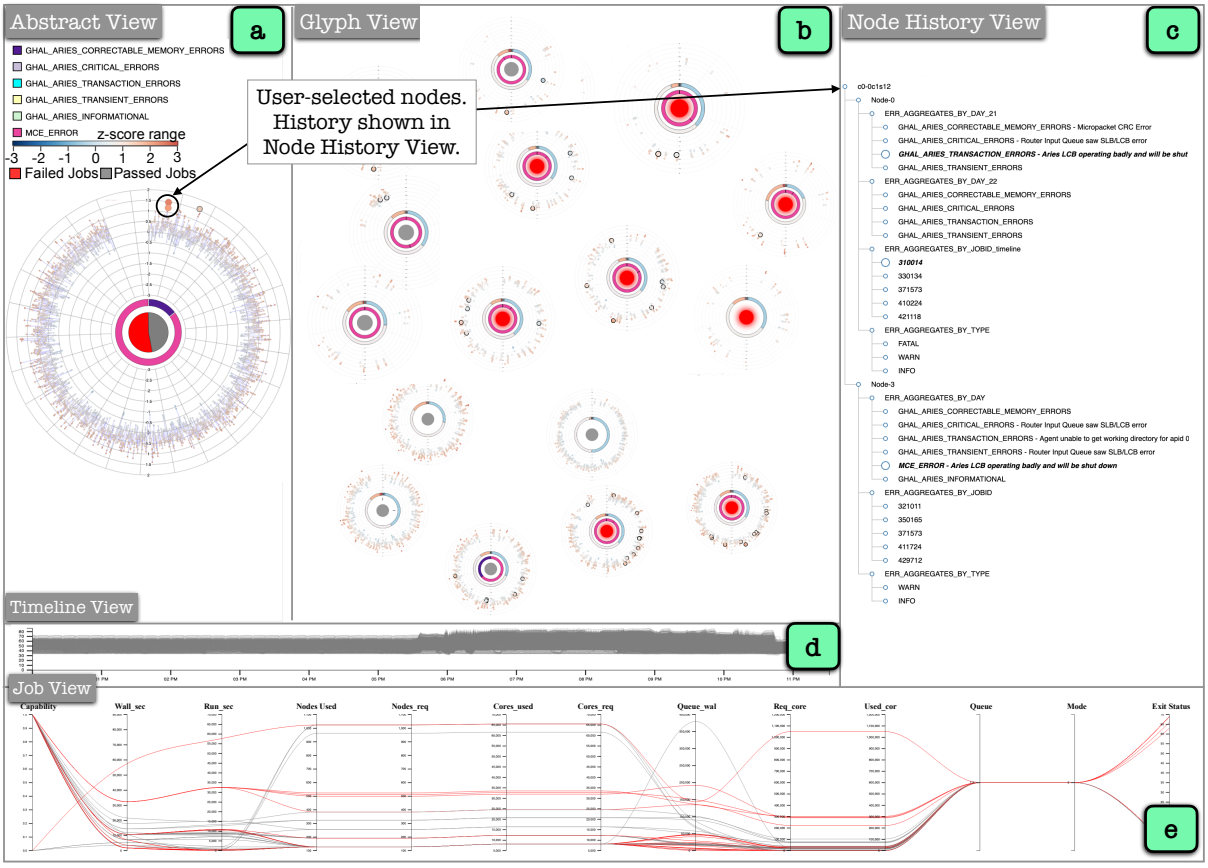

Figure 6.1: The UI of our visual analytics tool. (a) The abstract view shows the aggregated values of the log data. (b) The glyph view shows the log data information separated by granularities of jobs, users, projects, nodes used, and job exit codes. (c) The node history view shows the history of the nodes selected by lasso selection in the abstract and the glyph views. (d) The environment log data (power, temperature, voltage, current, etc.) is displayed in the timeline view. (e) The job view uses parallel coordinates to display the details of the job log information.

including the hardware logs (event sequence and text data), job logs (event sequence data), environment logs (also referred to as SEDC (System Environment Data Collections) logs), syslogs, console logs etc. SEDC is a Cray systems tool used to collect and report temporal data (readings collected from sensors housed in the compute nodes) in real-time, collected from disparate subsystems and components of a supercomputer system. This data collected from the predefined spatial localities and various temporal resolutions are unprocessed and raw. Therefore, we deal with multifidelity large-scale data from various sources within the HPC system. For example, the hardware error log contains information accumulated from different control systems and interlinked

subsystems with data size ranging in tens of GB. The environment log data are more frequently reported and collected at every 10-30 seconds interval, making the dataset size approach gigabyte (GB) to terabyte (TB) range every few weeks. The job log data contains information about the various applications utilizing the systems and their characteristics (i.e., nodes used, start and end times, etc.) with data sizes ranging in hundreds of megabytes (MB) a year. The kinds of logs range in size from 5 GB per day. It is a challenging task to analyze these volumes of data daily.

This work uses multiresolution Dynamic Mode Decomposition (mrDMD) [95] to succinctly represent the larger environment log data. We then display its correspondence with the additional preprocessed log data like the hardware and job log data. MrDMD is a technique that decomposes high-dimensional data into correlated spatial-temporal modes and is an improved version of Dynamic Mode Decomposition (DMD) [141, 144, 146]. DMD decomposes the data into spatial modes that correlate it over its spatial features (similar to principal component analysis (PCA)) and temporal modes that correlate the data across unique temporal Fourier modes. The DMD algorithm has been used to isolate and extract distinct sleep spindle networks from sub-dural electrode array recordings of human subjects using DMD spectrum analysis and baselines analysis [15]. Furthermore, mrDMD extracts the spatial and temporal features over multiple timescales by integrating space and time. The multiresolution analysis in mrDMD recursively subtracts low-frequency, or slower varying, dynamics from the data at each selected timescale, making it ideal for separating different timescale features or modes and analyzing them separately. MrDMD has been used for object feature tracking in videos over evolving timescales [95].

Our work combines the mrDMD analysis [95] with the work on frequency isolation using DMD spectrum analysis to generate the mrDMD spectrum [15]. We extract the high power mrDMD modes at various frequency ranges. We then identify custom baselines that are either system or user-specific to extract patterns that signify changes between the typical system state (specified by the baselines) and the current state.

We make the following contributions:

- MrDMD Baseline identification for system logs is based on two factors
  - system or sub-component specification
  - user system-usage trends
- Our mrDMD algorithm has the following improvements and features
  - Time range splits at mrDMD levels are determined by the start and end times of system jobs. This ensures that the results of the system usage analysis do not overlap between different jobs or applications utilizing the system.
  - MrDMD mean mode magnitudes are extracted at frequency ranges where the "power" of the corresponding mode is higher than a specified threshold, thus eliminating noise.
  - We process the time range splits asynchronously at each level. This improvement increases the speed of the algorithm.
- Two use scenarios on real-world datasets demonstrating the effectiveness of our approach.

Since the data is proprietary, compiling this diverse multifidelity dataset from other supercomputing facilities is challenging. Nevertheless, our stated contributions will apply to other large-scale systems reporting similar data types. In addition, it will provide insights to users and system administrators to better understand the system's state at any given time, present or past, and to perform system maintenance to reduce future system failures proactively.

6.2 Related Work

6.2.1 Error Log Analysis in Large Scale Systems

Substantial efforts have been directed towards pinpointing and forecasting failures to facilitate evasive actions for failure identification in large-scale HPC systems, especially when current technology and infrastructure are delivering the capacity to record systems' states faster. Past survey papers [67, 78, 142] on forecasting and classifying failures give an in-depth view of current methodologies and their drawbacks based on log data analysis. Pre-processing of log data is the first step in log analysis, and the steps are grouped into three main actions: data categorization, data filtering, and causality filtering [192]. Some past efforts include real-time clustering algorithms to concisely represent temporal and event-based data with evolving clusters of information in the log files [49, 51, 77, 150, 152]. Prior works also explore the influence of power, and temperature on device reliability, with the main focus on hard disk drives, solid state drives, GPUs, and CPUs [7, 33, 73, 126, 147, 157]. Other efforts employ visual analytics solutions for studying large-scale system behavior. Some previous works include developing a multi-coordinated visual analytics tool to investigate and optimize the network communication of a supercomputer [45, 102, 151, 152], developing node-link diagrams with matrix-based hierarchical aggregation [46], functional data analysis (FDA) to incrementally and progressively update the streaming time series data for identifying outliers by using FDA and FPCA [150], visualization framework for parallel discrete-event simulations (PDES) [99]. Visual analytic approaches have been proposed to help filter and analyze millions of records of such inconsistent text data [14, 87, 125, 189]. A few automated log analysis tools [31, 49, 51, 64, 114, 192] use text or event pattern-based correlations, signal processing, pattern recognition and mining, and spatiotemporal event-based analysis. These tools analyze a specific type of data or a combination of data types, e.g., numerical, text-based, or event-based data. Our log data contains all three data types [151, 152]; thus, the task becomes challenging to process these data types individually and in conjunction. The results of the analysis are displayed on our visual analytics tool.

Past efforts also include scalable and interactive visual analytics tools [88, 123, 151, 152] for analyzing multifarious log data; however, these tools lack the capability to analyze and visualize extensive temporal data in a concise representation. Legacy tools like IBM Blue Gene Navigator [97] with plain log statistical visualization are popular tools operational administrators use to monitor supercomputer health.

This work has a front-end visualization showing correlations between hardware log, job log, and environment log data with synchronized time. In order to provide a holistic insight into comprehending the complexity of large-scale systems, we need to process and analyze a multitude of log data. Past attempts typically process only one or two types of error log data, and most studies ignore the analysis of the larger and more computationally expensive environment logs. In our work, we aim to address these shortcomings.

6.2.2 Multiresolution Dynamic Mode Decomposition

As mentioned above, mrDMD is a variation of the DMD algorithm. The original DMD algorithm was applied to fluid dynamics. It works with the assumption that observations can be approximated by a linear dynamical system that closely models the characteristics of the observations [95]. Therefore, DMD and mrDMD extract information from a nonlinear dynamical system [95, 141, 144]. Inspired by multiresolution analyses and, specifically, wavelet methods and windowed Fourier transforms, mrDMD recursively analyzes timescales, selecting modes to remove from the data of interest [95].

In recent years both DMD and mrDMD have served as powerful mechanisms for studying the dynamics of the nonlinear systems in fields including fluid mechanics [95,144,167], financial analysis [111], control systems [134] neuroscience [15], streaming analysis [68, 132] and denoising [30, 146], and foreground and background separation in video analysis [95,112]. DMD with unsupervised clustering was used to uncover distinct sleep spindle networks using cortical distribution patterns, frequency, and duration [15]. DMD has also been extended to incorporate control and demonstrated on a model with applicability to infectious disease data analysis with mass vaccination [134].

MrDMD of elastic waves with image registration and Kullback Leibler (KL) divergence was used to diagnose and localize the surface microscale defects in the Lead Zirconate Titanate [84]. MrDMD has been used to study and detect the onset of seizures in scalp electroencephalography (EEG) signals [9]. In a streaming low-storage fluid dynamic setting, DMD results were updated as new data became available. Their methods included a "batch-processed" formulation and a compression step [68]. Another parallelized algorithm extracts DMD modes using the streaming method of snapshots singular value decomposition on a graphics processing unit (GPU). They use the native compressed format of many data streams to reduce data transfer costs from CPU to GPU [132]. A denoising DMD implementation to reduce the algorithm's bias to sensor noise used three steps, including correcting the determined bias using known noise properties, performing DMD forwards and backward in time, combining the results, and developing an algorithm based on least-square analysis [30].

Since DMD and its variations have numerous applications in various fields, we attempt to leverage its benefits by combining the mrDMD analysis [95] with the DMD spectrum analysis for isolating mrDMD modes at automated or user-specified frequency ranges [15]. We have improved the sampling rate selection and temporal split identification in the recursion steps of the mrDMD algorithm. At each level of the multiresolution analysis, we compute mrDMD modes asynchronously which further increases the speed of our algorithm. These improvements increase the accuracy and speed of computation. We also choose baselines that we define as system and user-specific to extract meaningful patterns that indicate changes between the typical system state (represented by baselines) and the current state (represented by data of interest). To the best of our knowledge, currently, no tools visualize the mrDMD analysis results. Our tool visualizes the z-scores or standard deviation of the mrDMD results of current readings from the mrDMD results of chosen baseline readings.

6.3 System Overview

This section describes the overview of our visual analytics tool and the back-end analysis pipeline.

6.3.1 Overall Organization

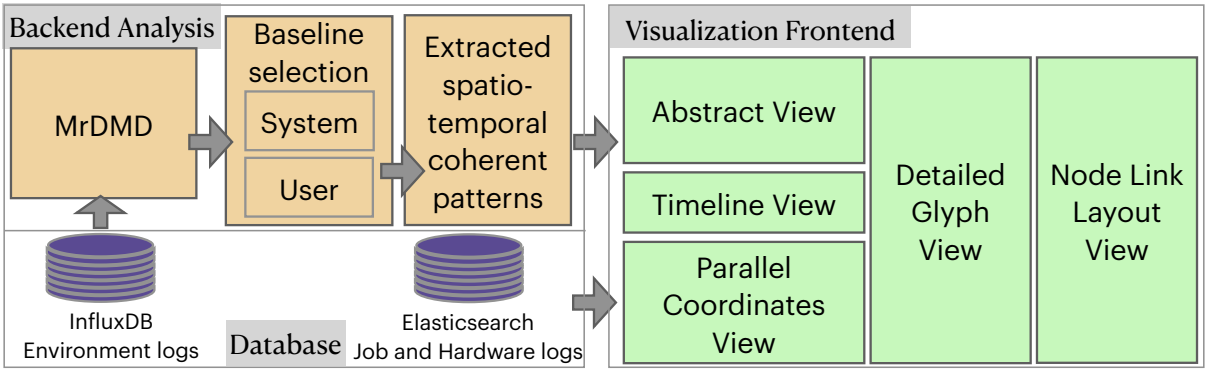

Figure 6.2: General organization of our visual analytics tool

Fig. 6.2 depicts the general organization of our system with three main sections: databases, back-end analysis, and front-end visualization. Our system operates on two 6-core, 2.4-GHz Intel E5–2620v3 processors (Intel Haswell architecture) and 256GB of DDR4 memory. We use InfluxDB [76], a time series database optimized for speedy storage and retrieval of time series data to store environment logs. The job and hardware logs are stored in an Elasticsearch database [55]. The data sizes are 3-4 GB per day, 1-2 GB per month, and a few hundred MB per year for the environment, hardware, and job logs. Our system operates on Python 3.8, a Flask server [57], and D3 [11] visualization. The source code will be made available on publication. The back-end analysis section is split into three groups consisting of the mrDMD analysis with our improvements, the baseline selection mechanism (where we choose between user-specific or system-specific baselines), and the extraction of the coherent spatiotemporal patterns based on the mrDMD power spectrum values. The visual analytics frontend consists of five sections with a built-in interaction mechanism (explained in detail in Fig. 6.1).

6.3.2 Backend Analysis

Jobs routinely utilize supercomputers for hours to weeks on thousands of nodes. A supercomputer node reports hardware and software errors along with information about jobs utilizing it. Each node also has multiple sensors that record power, temperature, current, memory, fan speed, etc., and are stored as environment logs. Our backend analysis pipeline mainly focuses on processing this large environment log dataset collected from sensors. This log data contains monitoring information from different sensors collected every 10-30 seconds. The resulting substantial data size poses a challenge and contributes to significant processing overhead. The supercomputer houses approximately 150 sensor readings per node. Furthermore, this dataset is usually ignored in log data analysis due to the size and lack of valuable insights extracted from popular methods.

6.3.2.1 Multiresolution Dynamic Mode Decomposition(mrDMD)

We will now briefly introduce the DMD algorithm [15, 95, 167]. Collecting K sensor readings from M snapshots in time, we may construct two raw data matrices $\mathbf{X}$ and $\mathbf{X}'$, in which $\mathbf{X}'$ columns are shifted by one time point from $\mathbf{X}$.

$$\mathbf{X} = \begin{bmatrix} | & | & | & & | \\ \mathbf{x}_1 & \mathbf{x}_2 & \mathbf{x}_3 & \dots & \mathbf{x}_{M-1} \\ | & | & | & & | \end{bmatrix}$$

$$\mathbf{X}' = \begin{bmatrix} | & | & | & & | \\ \mathbf{x}_2 & \mathbf{x}_3 & \mathbf{x}_4 & \dots & \mathbf{x}_{M} \\ | & | & | & & | \end{bmatrix}$$

The algorithm estimates the eigendecomposition for the best-fit matrix $\mathbf{A}$ such that

$$\mathbf{X}' = \mathbf{A}\mathbf{X}. \tag{6.1}$$

The DMD algorithm has the following steps:

1) Decompose the data matrix $\mathbf{X}$ with SVD [164]:

$$\mathbf{X} = \mathbf{U}\mathbf{\Sigma}\mathbf{V}^* \tag{6.2}$$

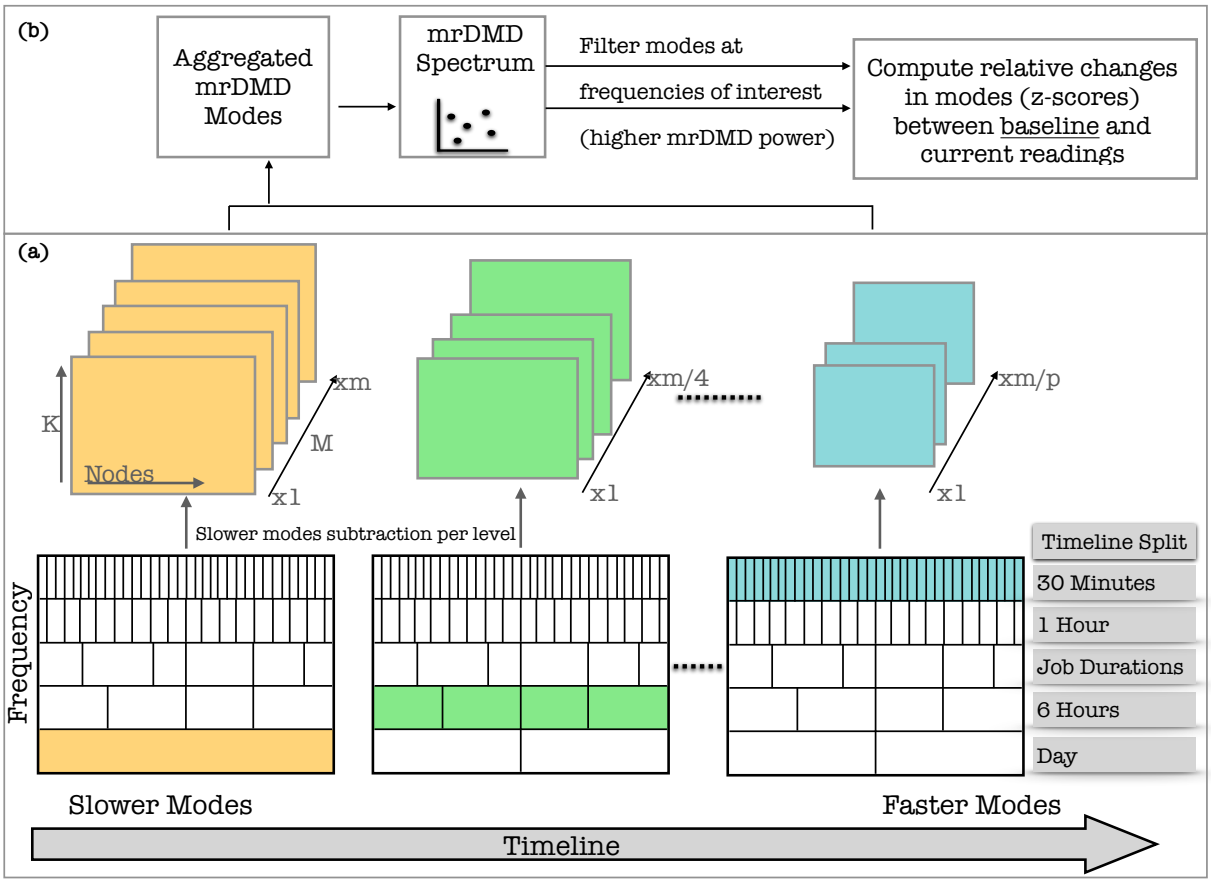

Figure 6.3: Representation of the multiresolution dynamic mode decomposition [95]

where $*$ is the conjugate transpose, $\mathbf{U} \in \mathbb{C}^{K \times r}$, $\mathbf{\Sigma} \in \mathbb{C}^{r \times r}$ and $\mathbf{V} \in \mathbb{C}^{M-1 \times r}$. Here r is the rank of the reduced SVD approximation to $\mathbf{X}$. The SVD rank reduction is computed using the optimal Singular Value Hard Threshold (SVHT [53]).

2) Compute the $r \times r$ projection of the full matrix $\mathbf{A}$ onto $\mathbf{U}$'s low-rank modes, $\mathbf{A'}$.

$$\mathbf{A'} = \mathbf{U}^* \mathbf{A} \mathbf{U} = \mathbf{U}^* \mathbf{X'} \mathbf{V} \mathbf{\Sigma}^{-1} \tag{6.3}$$

3) Compute eigendecomposition of $\mathbf{A'}$:

$$\mathbf{A'W} = \mathbf{W\Lambda} \tag{6.4}$$

where eigenvectors are found in the columns of $\mathbf{W}$ and the corresponding eigenvalues λ_i are in diagonal matrix $\mathbf{\Lambda}$.

4) Compute the DMD modes

$$\mathbf{\Phi} = \mathbf{X'V\Sigma^{-1}W} \tag{6.5}$$

Each column of $\mathbf{\Phi}$ contains a DMD mode ϕ_{i} corresponding to the ith eigenvalue. Knowing the low-rank approximation of the data in the form of eigenvalues and eigenvectors, the projected future result can be constructed for all time and is given by

$$\tilde{x}(t) = \sum_{\mathrm{i=1}}^{\mathrm{r}} \mathrm{b_i}(0)\phi_{\mathrm{i}}(\boldsymbol{\xi})\exp(\omega_{\mathrm{i}}\mathrm{t}) = \mathbf{\Phi}diag(exp(\omega t))\boldsymbol{b} \tag{6.6}$$

where $b_i(0)$ is the initial amplitude of each mode, $\mathbf{\Phi}$ is the matrix whose columns are the eigenvectors ϕ_i, $\omega_{\mathrm{i}} = \ln(\lambda_{\mathrm{i}})/\Delta\mathrm{t}$, where Δt is the time step, $diag(exp(\omega t))$ is a diagonal matrix whose entries are the eigenvalues $exp(\omega t)$, $\boldsymbol{\xi}$ are the spatial coordinates, and $\boldsymbol{b}$ is a vector of the coefficients b_i.

The mrDMD is a recursive algorithm that removes low-frequency content from snapshots of data collected over a period of time. Low-frequency content captures the slow-varying dynamics of the system. M is chosen so that it is large enough to represent the system's dynamics, i.e., enough high- and low-frequency components are present. At the initial recursion step, the slowest DMD modes (*m1*) are subtracted from the DMD result (Fig. 6.3(a)). The slower modes correspond to values below a threshold, computed by a user-defined maximum number of oscillations in the time series divided by the time range (i.e., the number of time points) at each split. We have chosen the maximum number of oscillations for our current dataset to be 4. We then split the dataset into segments along the timeline (as shown in Fig. 6.3(a)). DMD is once again separately performed on each split segment. The slowest modes are subtracted again, and the recursive algorithm continues until termination. In mrDMD, due to the subtraction of slower modes in the previous levels, at each level, the multiresolution features are represented by different spatiotemporal DMD modes. Therefore no single set of modes dominates the SVD and influences features at multiple levels or time scales.

The future solution for all time using the mrDMD low-rank approximation of the system as in Eq. 6.6 is given by [95]:

$$x_{mrDMD}(t) = \sum_{i=1}^{M} b_i(0)\phi_i^{(1)}(\xi)\exp(\omega_i t) \tag{6.7}$$

$$= \underbrace{\sum_{i=1}^{m_1} b_i(0)\phi_i^{(1)}(\xi)\exp(\omega_i t)}_{\textit{(slower modes)}} + \underbrace{\sum_{i=m_1+1}^{M} b_i(0)\phi_i^{(1)}(\xi)\exp(\omega_i t)}_{\textit{(fast modes)}} \tag{6.8}$$

here the $\phi_i^{(1)}(x)$ represent the DMD modes computed from the full M snapshots. The first term in Eq. 6.8 has the slower dynamics whereas the second term has the faster dynamics. The second term gives the fast scale data matrix that is sent forward in the recursion step.

At each level of the data split, our algorithm automatically determines the sampling rate. Since the slower modes are subtracted at the initial levels, we use a lower sampling rate of the time series data at these levels. As the levels get higher, we increase the sampling rate, hence processing more time points (i.e., extracting higher frequency modes). For the supercomputer log, we determine the sampling rate as four times the Nyquist limit to capture cycles [162]. This value is customizable. At each level of the mrDMD analysis, we asynchronously process each time range generated after the split, as the time range results within a level are independent of the other time ranges. This improvement reduces the computation time of the analysis (refer section 6.5 Fig. 6.17).

The jobs utilizing the supercomputer execute for multiple hours. The original mrDMD algorithm split the timeline at each level into equal parts [95]. However, when handling environment data collected from multiple nodes in a supercomputer, a single timeline split may contain time series from multiple jobs. Since each job may utilize the system differently, mrDMD modes computed at each split may contain results from the time series of multiple jobs. This overlap of mrDMD modes will result in erroneous results, especially if the user wants to filter results per job. Therefore, we improved the algorithm by taking into account the start and end times of all jobs utilizing the nodes from which the time series are processed. We determine a value at which we split the timeline resulting in the minimum overlap between job start and end times.

When analyzing time series data from multiple nodes, we calculate a time point for split where the distance between the start or the end times of jobs in the vicinity (±2 hours, this number depends on the level at which the job duration split (refer Fig. 6.3) is performed could range from a few minutes to hours) of the chosen time point is minimum. This improves the final results, as seen in section 6.4 (Fig. 6.9). We use job duration split at level 3 in our analysis (refer Fig. 6.3(a))). However, it is customizable to be included in other levels depending on the length of the jobs and the initial length of the time series.

6.3.2.2 Frequency Isolation of spatiotemporal modes using mrDMD spectrum

The mrDMD decomposition at each level results in spatial modes ϕ_i, each with a corresponding eigenvalue λ_i describing its temporal dynamics. The eigenvalue λ_i is complex-valued. The dynamics can be easily interpreted after the above transformation $\omega_i = \ln(\lambda_i)/\Delta t$; the sign of the real component of ω_i determines if the corresponding mode dynamics are growing (positive), or decaying (negative), and the imaginary component determines the frequency of oscillations. We compute the frequency of oscillation of mode ϕ_i in units of cycles per second as

$$f_i = \left| \frac{\text{imag}(\omega_i)}{2\pi} \right| \tag{6.9}$$

Analogous to the traditional power spectrum computed with the fast Fourier transform (FFT) algorithm [179], the mrDMD "power" spectrum is then computed at each frequency of oscillation f_i and is given by $P_i = \left|\phi_i\right|_2^2$ [15]. Our mrDMD algorithm uses mrDMD mode magnitudes where the DMD spectrum power (normalized to 1.0) is greater than 0.5. Our experiments have shown that this value helps adequately capture the low-rank representation of the dynamics of our current dataset. This improvement removes the noisy low power mode magnitudes and their corresponding frequencies and also helps reduce the computation time.

6.3.2.3 Baseline Selection

Once we extract the high-power mrDMD modes from frequency ranges using the mrDMD spectrum, we compute the absolute values of the extracted mrDMD modes

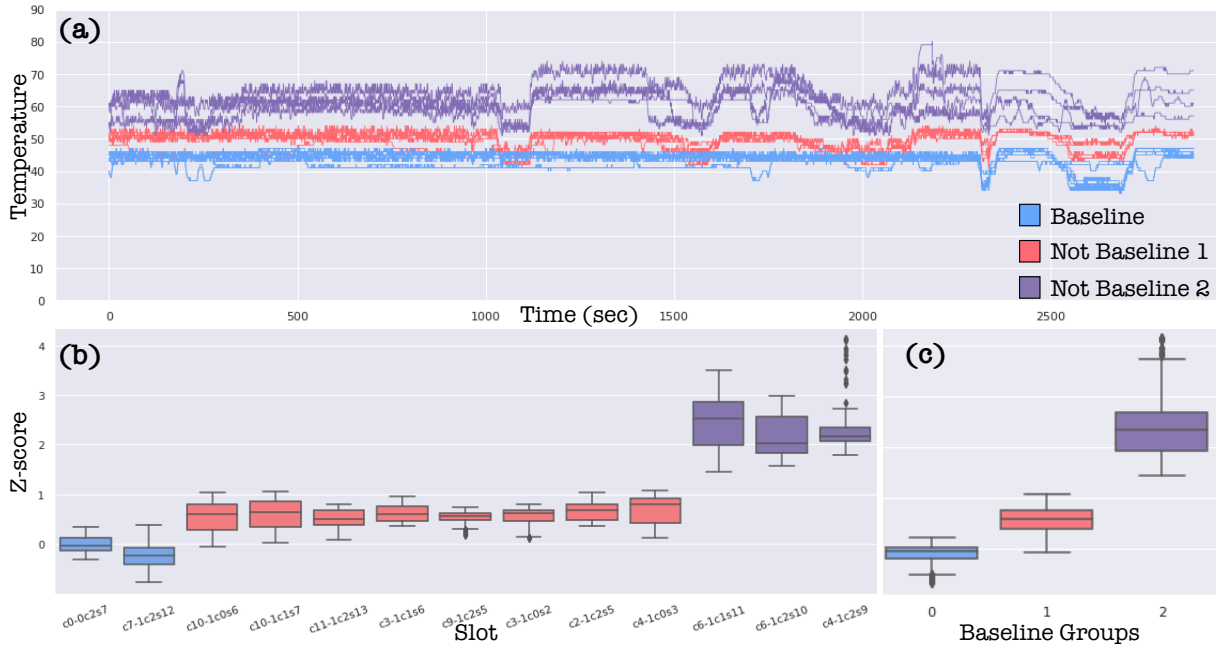

Figure 6.4: Analyzing (a) the supercomputer environment logs containing selected baseline readings(blue) and non-baseline readings(red, purple). (b) and (c) show the boxplots of the z-scores (change of non-baseline readings from baseline) versus slots(each slot contains 4 supercomputer nodes) or time series groups.

in the frequency ranges and average it to obtain the mean mode magnitudes. We do this separately for readings that we classify as baseline readings and non-baseline readings. Baseline readings constitute a subset of time series data that indicate an expected system functionality [15]. We choose the baseline readings based on (1) the system specifications that constitute normal functioning, e.g., readings within the normal temperature range without causing the supercomputer to overheat, or (2) the user's usage trends as seen in the environment logs of the nodes used by the user's jobs. We use the system manuals to identify the baseline value ranges for the first baseline type. For the second baseline type, we pick the environment log readings from nodes allocated to a user's job that has completed a successful execution.

We then compute the standard deviation of the non-baseline minus the baseline mean mode magnitudes at each reading (time series) for a trial set through bootstrapping. The trial set consists of time series following a typical scenario, e.g., passed job status, no hardware errors, etc. We then compute the z-scores of the difference from the

baseline at each reading (time series) using the bootstrapped calculations of the standard deviation [15]. Thus we obtain the relative changes in mean mode magnitudes for each new data using the previously computed results for data exhibiting normal behavior. For the system specification baseline, the z-scores capture the difference between what we identify as normal system (baseline) behavior and current system (non-baseline) behavior. For the user's usage trend baseline, the z-scores capture the difference between the utilization of the system by a user over time.

Fig. 6.4 shows how baseline and non-baseline readings transform into z-score values. Fig. 6.4(a) shows the temperature readings, grouped as baseline (blue) and non-baseline (purple and red) readings, for multiple nodes in the supercomputer. Fig. 6.4(b-c) shows the z-scores versus slots and the reading groups (groups 0, 1, and 2 constitute blue, red, and purple readings, respectively). The reading groups are chosen based on the differences in their magnitudes and variations over time. Here as expected, the z-score values for baseline readings are close to 0 (y-axis). However, for those readings away from the baseline, the readings with a smaller z-score change (< 1) are in red and those with a larger change (> 1.5) are purple. We further explain the use of these types of baselines in the case study section 6.4.

6.3.3 Visualization Frontend

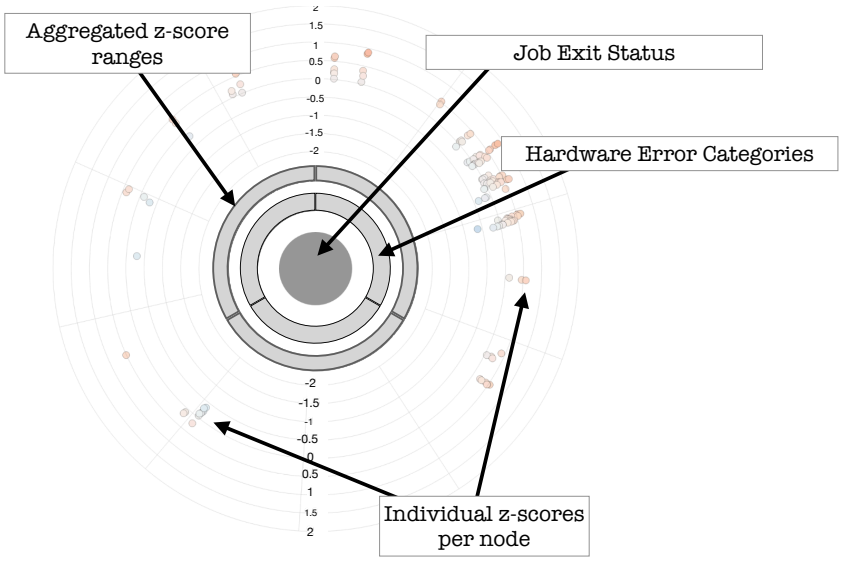

Figure 6.5: Abstract and glyph view layouts

Fig. 6.1 shows the UI of our visual analytics tool. The abstract view (Fig. 6.1(a)) shows the aggregated values of the log data. The glyph view (Fig. 6.1(b)) shows the log data information separated by granularities of jobs, users, projects, nodes used, and job exit codes. The glyph view uses the force directed layout algorithm [120]. The abstract and the glyph view sections (Fig. 6.5 and Fig. 6.1(a),(b)) contain a center circle specifying if the job has passed (gray) or failed (red). The innermost pie chart shows the hardware log error categories grouped by their counts for the job run duration. The glyph view (Fig. 6.1(b) and Fig. 6.5) has another concentric pie chart that shows the z-score values grouped by counts within one standard deviation range (which is customizable) from values -2 to 2. Note that z-score values can range from -10 to 10 in the supercomputer log dataset. The z-scores have a lighter white hue if they lie closer to the baseline, i.e., 0. We display the counts of nodes on the mouseover interaction over each arc of this pie chart. The outermost radial view shows the scatter plot of the z-scores for each node utilized by the job. Since there are $4,300$ nodes, the radial axis takes $4,300$ node numbers (IDs) in clockwise order. The node history view (Fig. 6.1(c)) shows the history of the nodes selected by lasso selection in the abstract and the glyph views (Fig. 6.1(a,b)). In the timeline view (Fig. 6.1(d)), we display environment log data (power, temperature, voltage, current, etc.). The timeline view has a built-in brush interaction using which one can refine the results of the node history view (Fig. 6.1(c)). The job view uses parallel coordinates to display the details of the job log information. On clicking each job in this view, the glyph and node history view results get updated. As a part of our future work, we plan to minimize the z-score overlap in the radial view through aggregations. Our current implementation provides scrolling interaction to zoom in to a section of the radial axis, thus eliminating the overlaps.

In Fig. 6.1(a), the user has selected nodes from the z-score radial view. The size of the nodes is proportional to the size of the fatal errors in the system for the time duration specified in the timeline view (Fig. 6.1(d)). This example shows an 11-hour duration data. Using the lasso selection, the user can select these nodes to view them in finer detail. On node selections, the glyph view (Fig. 6.1(b)) gets updated, and here

it shows the jobs that have been executed in this duration of our analysis. In this view, we can choose other granularities such as jobs, users, projects, nodes used, and job exit codes. The job details are plotted in the parallel coordinates view (Fig. 6.1(e)), giving more information about its specifications. The node history view (Fig. 6.1(c)) then shows the current errors and history of the errors in the selected nodes and the jobs that encountered these errors. Here Nodes 0 and 3 in slots $c0-0c1s12$ are the problematic nodes reporting transaction and MCE (machine check exception) errors.

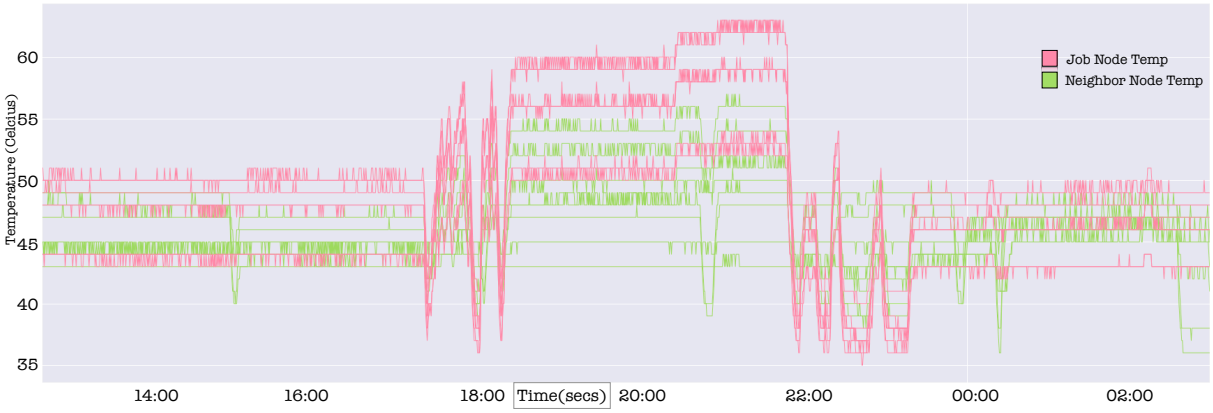

Figure 6.6: Temperature readings of the Cray HPC system where time series in light pink show temperature at nodes utilized by a job and time series in light green show temperatures in the neighboring nodes.

In this work, we analyze and process environment log readings and study their behavior and effects on the system and users. Fig. 6.6 shows an example of temperature readings from nodes in the theta supercomputer utilized by a job (light pink). We then plot the temperature readings of nodes in the spatial vicinity of these job nodes. We see that a spike in the job node readings from 18:00 to 22:00 is mirrored by the nodes in its vicinity. It is clearly evident that neighboring jobs affect each other's behaviors and could affect the node's overall performance.

We clustered environment log temperature readings recorded during user job executions using Agglomerative clustering algorithms, and we plotted the results in a 2D plain using PCA, as shown in Fig. 6.7 (bottom). Fourteen readings belonging to each cluster group are plotted in Fig. 6.7 (top). The clusters-(0,1,2) isolate mostly lower frequency readings, and cluster 3 comprises high-frequency readings. Although en-

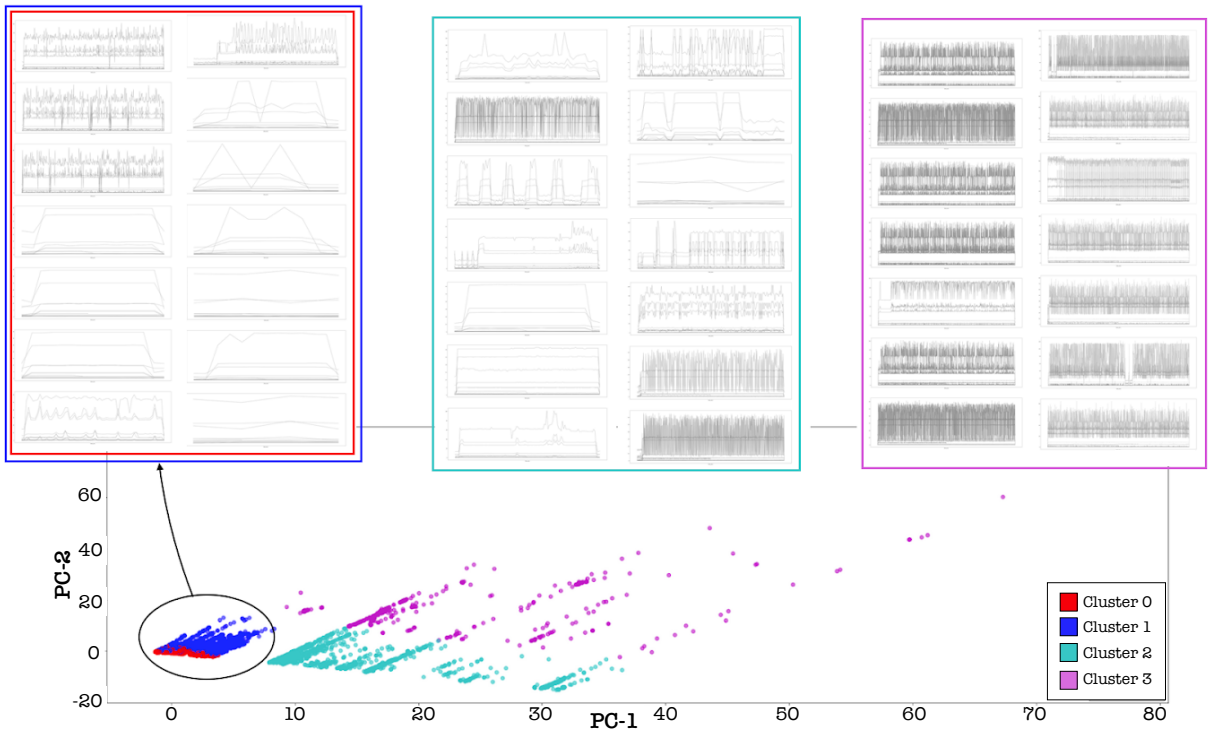

Figure 6.7: Analysis of environment log readings using Agglomerative clustering plotted on a 2-D plain using PCA (bottom). The time series of each cluster group is shown at the top.

vironment log data are traditionally ignored in log analysis, this analysis shows us that environment logs can be used to group and extract users' usage patterns and the corresponding system behaviors. The auxiliary views (Fig. 6.1) in our visual analytics tool can isolate and extract this information for further analysis.

6.4 Case Studies

6.4.1 Case Study 1

We describe two detailed cases of supercomputer utilization by user jobs and present how our system aids in identifying usage patterns from large-scale and diverse logs. We have shown an analysis example in Sec. 2.3. The case studies further demonstrate the mrDMD results using our visual analytics tool and compare the processed data from each log type mentioned previously. Our environment logs are collected from the Cray XC40 supercomputer and contain data from 24 compute racks with a total of 4,300 nodes. Sensors on each node collect data in 10-30 second intervals. We mainly

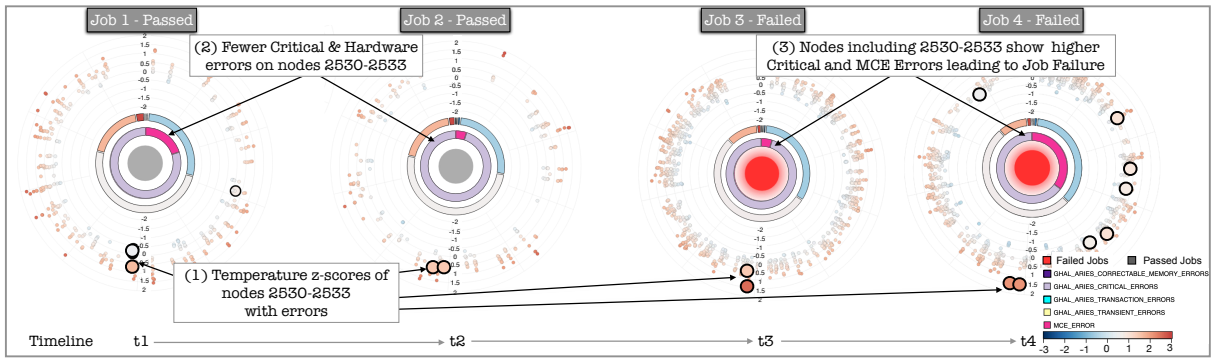

Figure 6.8: Glyph view of four jobs utilizing the supercomputer. The job glyphs are sorted in the ascending order of their start times. The first two jobs (1 and 2) have exited with a pass status and the last two (3 and 4) have failed. Nodes $2530-2533$ consistently report hardware (MCE and critical) errors. The z-score values, computed using the system specification baseline, increased for the nodes $2530-2533$ from job 1 to job 4 resulting in node 2533 shutdown along with job 4 failure.

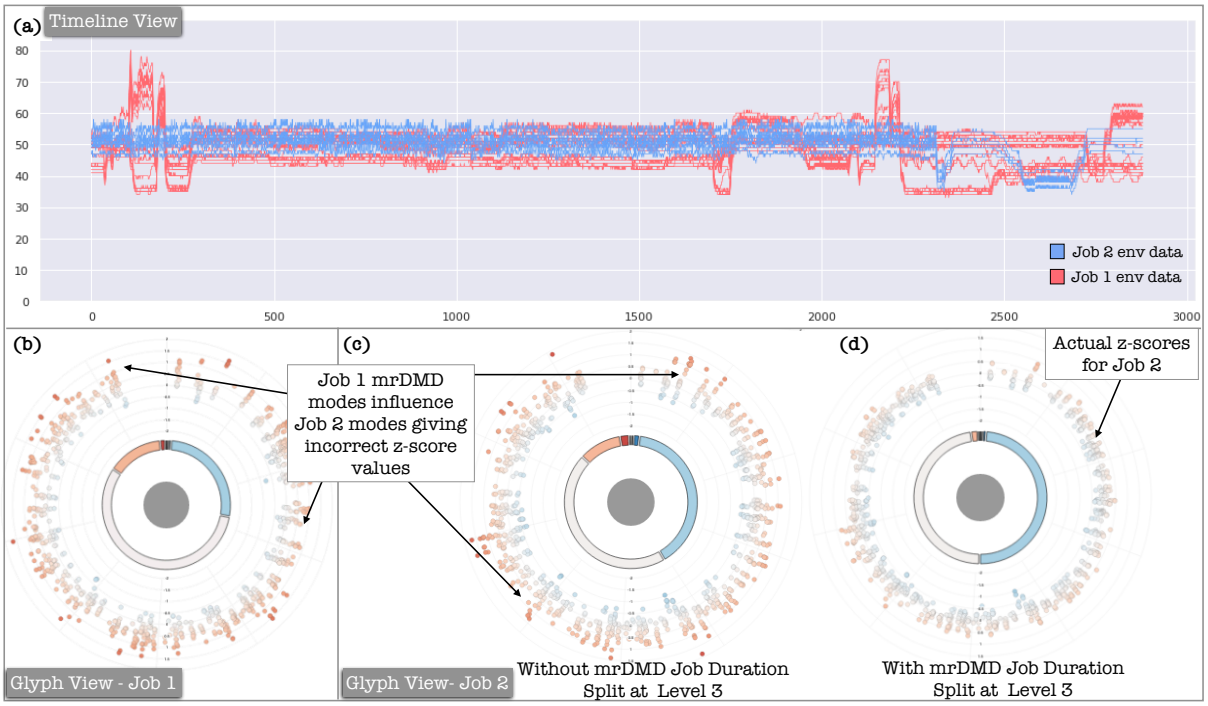

Figure 6.9: Analysis of two jobs utilizing the system in the same spatial and temporal locality. (a) shows a subset of the environment log readings for job 1 (red) and job 2 (blue), (b) shows job 1 glyph view, and (c)(d) shows glyph views for job 2 without and with the mrDMD timeline split for job duration at level 3, respectively.

analyze the temperature and current readings from the environment logs. There are 4 temperature and 4 current readings per node. We perform the mrDMD analysis and compute the z-scores separately for each type of reading.

The Fig. 6.8 shows the glyph view for four different jobs utilizing the system starting from time $t1$ to $t4$. Here we ordered the job glyphs in the ascending order of their start time, i.e., $t1 < t2 < t3 < t4$. Each glyph contains a center circle that specifies if the job has passed or failed. The first two jobs have passed (gray), and the last two have exited with a failed (red) status. The innermost pie chart shows the hardware log error categories grouped by their counts for the job run duration. Fig. 6.8 shows that the four jobs have both MCE (machine check exceptions) and critical errors throughout the job runs. The next concentric pie chart shows the z-score values grouped by counts within one count range from values -2 to 2. Note that z-score values can range from -10 to 10 in the supercomputer log dataset, but they vary from -2 to 2 z-score range for the current temperature examples. The z-scores have a lighter white hue if they lie closer to the baseline, i.e., 0. Here we see that most of the z-scores follow typical behavior and hence have a lighter hue. We display the counts of nodes within each z-score range on the mouseover interaction on each arc of the pie chart. The outermost radial view shows the scatter plot of the z-scores for each node utilized by the job. Since there are $4,300$ nodes, the radial axis takes $4,300$ node numbers (IDs) in clockwise order. The larger nodes are reporting hardware errors. We have the hover interaction, which gives the node IDs on the mouseover interaction. The user can also pan or zoom in to a region of the radial axis, which helps a user scan the visualization in more detail. The four jobs chosen ran during a 24-hour duration and utilized a set of nodes with IDs 2430-2533 that consistently reported hardware errors. We have used system specification baselines for all four jobs. We see from job 1 that the z-score move from being closer to the baseline (white, light blue hues) to being much higher than the baseline (orange hues) for job 4. Orange hues signify higher temperature on the nodes, and this temperature increase will affect the neighboring nodes as well (see Fig. 6.15). Also, the number of nodes reporting hardware errors increased significantly at job 4, resulting in a node failure of node 2533 and eventually the job 4 failure. Finally, the node history view (not shown in Fig. 6.8) summarized additional details of the errors as *Router input queue saw SLB error*, *Aries LCB operating badly and will be shut down*, and

Node ID 2533 unavailable. Our tool helps visualize these consistent patterns of failures for the first three job runs, indicating that these anomalous nodes need to be checked by system administrators and any hardware failures be immediately rectified before a node shuts down, as in this case. Another usefulness of utilizing z-scores from mrDMD analysis of environment logs is that nodes in the immediate vicinity also report similar hardware errors, although this may not always lead to node shutdown.

Fig. 6.9 shows the analysis of two jobs utilizing the system in the same spatial locality. Job 1 had executed before job 2 (Note that we have overlayed 3000 time points of both jobs along the x-axis for clarity). Fig. 6.9(a) shows a subset of the temperature readings reported by the nodes utilized by the jobs, job 1 (red) and job 2 (blue). Temperature readings for job 2 mostly follow close to the selected baseline with a range between 40°-60° Celsius. However, the readings greatly fluctuate for job 1, between 30°- 80° Celsius. Fig. 6.9(b) shows the glyph view for job 1, as expected, the z-scores are largely varying as it follows the time series pattern shown in Fig. 6.9(a). Fig. 6.9(c)(d) show the glyph views for job 2 without and with the mrDMD timeline split for job duration at level 3, respectively. Without the job duration split in the mrDMD analysis (Fig. 6.9(c)), the z-score patterns for job 1 and job 2 look similar even though that behavior is not reflected Fig. 6.9(a). The mrDMD analysis could give erroneous results due to overlapping time ranges between the end of job 1 and the beginning of job 2. This results in higher frequency modes between jobs to overlap and, in some cases, leads to erroneous results, as seen in Fig. 6.9.

6.4.2 Case Study 2

We utilize environment logs from the Cray XC40 supercomputer that contain data from 24 compute racks with a total of 4,300 nodes. In this case study, we filter jobs by a single user. The figure shows the glyph view for three different jobs from a single user utilizing the system starting from time $t1$ to $t3$. Here we ordered the job glyphs in the ascending order of their start time, i.e., $t1 < t2 < t3$. Each glyph contains a center circle that specifies if the job has passed or failed. The first two jobs have passed (gray), and the last one has exited with a failed (red) status. The innermost pie chart shows

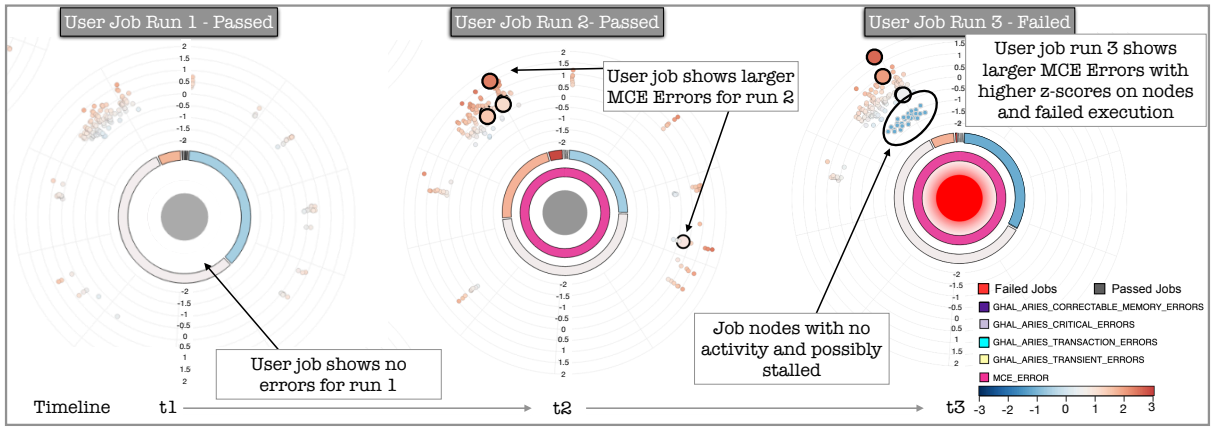

Figure 6.10: Glyph view of 3 jobs utilizing the supercomputer. The job glyphs are sorted in the ascending order of their start times. The first two jobs (1 and 2) have exited with a pass status and the last one (3) has failed. Few nodes consistently report hardware (MCE) errors. The z-score values, computed using user specific baseline, increased for the error nodes in jobs 2 to job 3. Job 3 showed a subset of nodes that were underutilized and were possibly stalled.

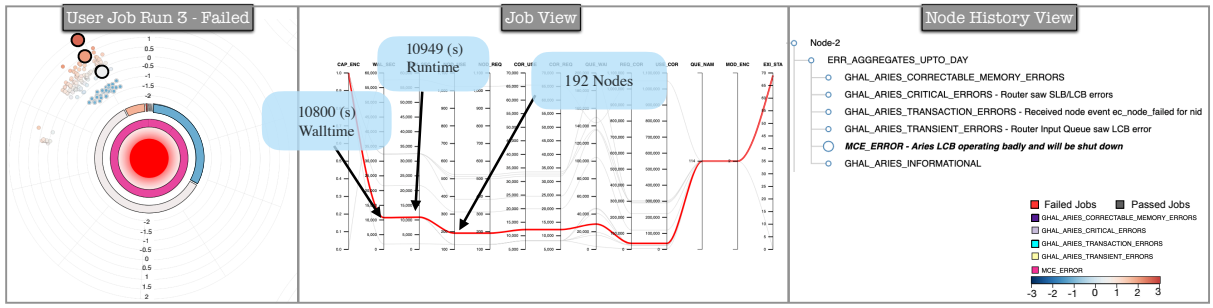

Figure 6.11: Analysis of job 3 from Fig. 6.10 showed that the job exceeded the allocated time and consistently reported errors in the duration of its runtime with possibly underutilized nodes.

the hardware log error categories grouped by their counts for the job run duration. Fig. 6.10 shows that two out of three jobs have predominantly reported MCE (machine check exceptions) errors throughout the job runs. Note that we have filtered out the informational type hardware log data. The next concentric pie chart shows the z-score values grouped by counts within one count range from values -2 to 2. The larger nodes are reporting hardware errors. The user can also pan or zoom in to a region of the radial axis, which helps a user scan the visualization in more detail and controls visual clutter. The three user jobs chosen ran during a 24-hour duration and utilized 192 nodes. User jobs 2 and 3 consistently reported a larger number of hardware MCE errors on three nodes.

In this case study, for the mrDMD analysis, we chose the user-specific baseline readings. We picked the baseline readings from user job 1. The user job 1 (Fig. 6.10) utilized 192 nodes with two readings per node, giving a total of 384 readings. Since we select baselines from user job 1 (one reading per node), the z-scores are mostly within the range (-1 to 1), i.e., they lie closer to the baseline value 0, with roughly one standard deviation variation on either side. However, for user jobs 2 and 3, the z-scores lie from -2 to 2. A reason could be that the user made improvements or changes to their code to utilize the system at varying capacities. In this case, user job 2 showed significant MCE errors in 4 nodes and increased z-score values on one node ID 4505. The next run, user job 3, showed MCE errors with much higher z-score values, particularly on one node ID 4507. However, we also see a small subset of nodes having a z-score much lesser than the baseline. This indicates that the nodes are not being utilized, as lower z-score values indicate a lesser current on these nodes compared to the previous runs. Also, low values of z-scores indicate a possible stalling where nodes are waiting without performing any computation for the duration of the job run.

In Fig. 6.11, we analyze job 3 further using the auxiliary views, i.e., the job view and the node history view. On selecting the job 3 in the glyph view, the job is highlighted in the job view, and we see in Fig. 6.11 that the runtime, i.e., the execution time of the job had exceeded the wall time (values visible through hover interaction), i.e., the maximum time allocated to the job. This scenario occurs when the application has stopped, but the job is still perceived to be running by the system. In such cases, the system resources could be in a stalled state and hence are unused, bringing down the usability of the overall supercomputer. Using this insight, the system admins could request a system recovery and repair in case of a consistent system errors across jobs. Furthermore, the system admins could inform the user to either fully utilize the allocated nodes or request fewer nodes on subsequent job runs. In this case study, the node history layout (Fig. 6.11) indicates consistent *MCE* and *Aries LCB errors on the network link*. However, the node history also revealed memory-related, critical, transient, and transaction errors when utilized by other jobs in the past 24 hours.

6.4.3 Case Study 3

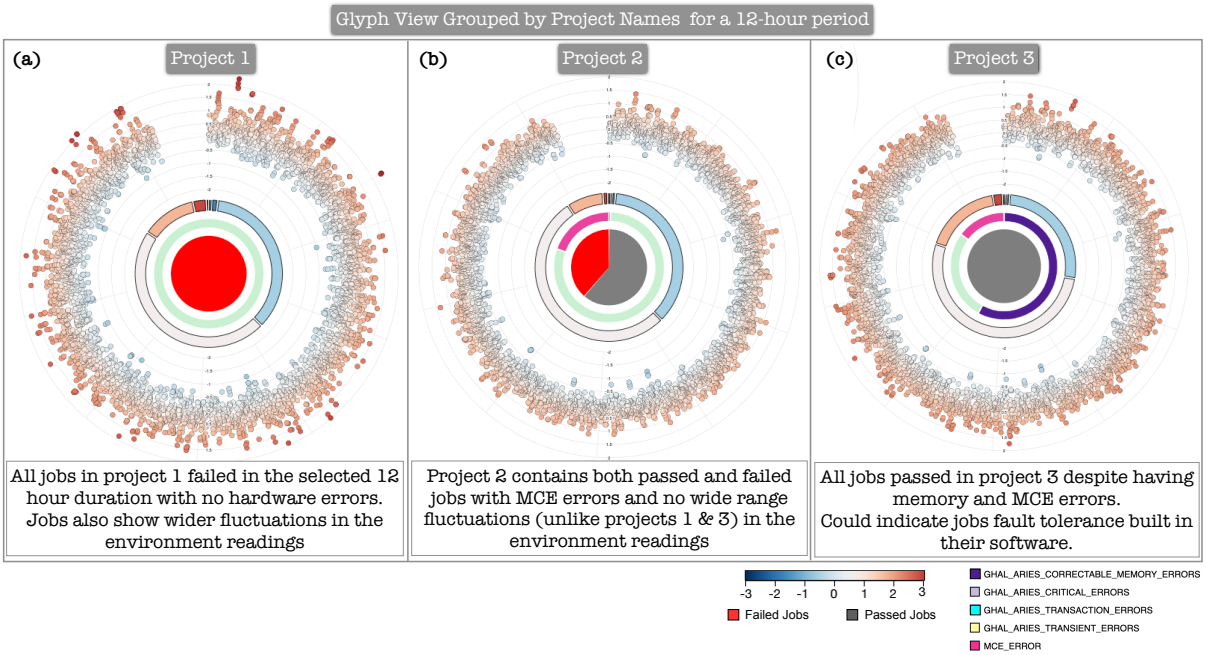

Figure 6.12: Glyph view of 3 scenarios of projects utilizing the supercomputer whose jobs are grouped over 12 hours. Project 1 (a) shows jobs that have all failed but report no hardware errors; however, they indicate larger fluctuations from the baseline. Project 2 (b) shows jobs that have both passed and failed jobs and report MCE errors; however, no wider fluctuations are seen from the baseline. Project 3 (c) shows jobs that have all passed but show memory and MCE errors and contain environment readings with wider fluctuations from the baseline.

In Fig. 6.12, we show glyph views of three peculiar scenarios for projects utilizing the supercomputer. Each project contains multiple users who execute multiple jobs on the supercomputer. Project 1, Fig. 6.12(a), shows jobs that have all failed but report no hardware errors; however, they indicate larger fluctuations from the baseline. On closer inspection, the jobs in the project are mostly capability jobs and are known to show high-temperature readings utilizing a larger number of supercomputer nodes. It is interesting to note that in the given time window of 12 hours, all jobs belonging to job-queues backfill and default had failed without hardware errors or useful informational messages; since we do not track software errors in this work, this could be due to software/code issues in the user applications. Project 2, Fig. 6.12(b), shows jobs that have both passed and failed jobs and report MCE errors; however, no wider fluctuations are seen from the baseline. These types of z-score fluctuations are seen when the errors

cause an immediate job failure and release of nodes to users on the waitlist and do not cause the overall node temperature to increase. Project 3, Fig. 6.12(c), shows jobs that have all passed but show memory and MCE errors and contain environment readings with wider fluctuations from the baselines. These are a particular set of jobs utilizing the supercomputer with fault tolerance built into their programming. The admins could use this information to help guide and inform users of other projects to take such preventative measures to avoid job failures. Using the above categories of projects, the system admins could explore the types of project jobs and their utilization trends to understand how the underlying system can be better utilized, increasing the system's usability and availability.

6.5 Discussion

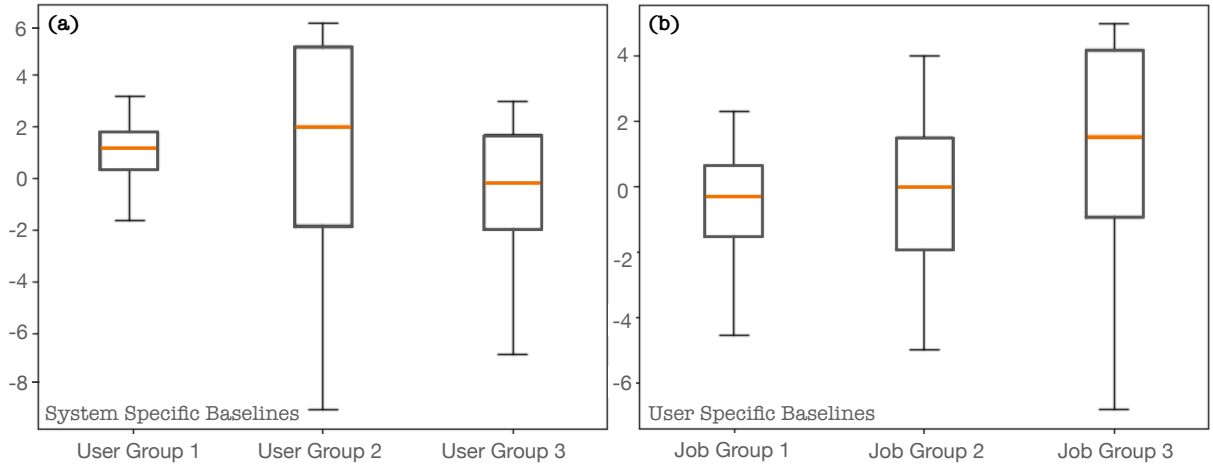

Figure 6.13: Analyzing the supercomputer log data using the (a) system-specific baseline and (b) user-specific baseline.

We have presented our approach's effectiveness in analyzing large-scale time series data and its relevance to processing supercomputer environment logs to draw meaningful insights regarding the system's functionality. We further discuss the strengths of our approach in this section in categorizing jobs and users of the supercomputers using temperature readings.

First, we applied our mrDMD analysis to environment logs. We store the z-score results in an Elasticsearch database. We then isolated 11,654 jobs spanning the years

2019-2020. Fig. 6.13(a) shows the boxplot of z-scores versus user groups for the supercomputer. Using the system-specific baseline, we were able to isolate particular groups of user utilization patterns on the supercomputer using temperature readings. The x-axis of Fig. 6.13(a) shows users' job usage patterns that are similar, i.e., there is no overlap of users along the x-axis groups. User group 1 mainly consists of small runtime jobs (< 2 hours) with z-scores slightly over the baseline. User group 2 consists of larger jobs, usually utilizing more than 256 nodes in the system. Most of the user group 2 jobs are high-capability jobs that fully use the memory and show higher temperature readings at most nodes throughout the job runtime, which was greater than 4 hours. User group 3 also consists of longer running jobs (> 2 hours). However, unlike user group 2, these jobs are not high-capability. Knowing this information about supercomputer usage patterns, the system admins can encourage the users to request fewer nodes when they do not fully utilize their allocated nodes for their jobs. For shorter-running jobs, the system admins can reduce the user's wait times for node allocation and hence improve their job scheduling strategies [131].

Fig. 6.13(b) shows the boxplot of z-scores versus job groups for the supercomputer. Using the user-specific baseline, we were able to isolate particular groups of job utilization patterns on the supercomputer using temperature readings. The x-axis of Fig. 6.13(b) shows job patterns that are similar, i.e., there could be an overlap of users along the x-axis groups. Here, for each user, we chose one job that had no hardware errors and executed with a pass status. We then used the standard deviation computed from the mrDMD mode magnitudes of the baselines for the users' subsequent job runs. We repeated this process for over 300 users utilizing the supercomputer. Fig. 6.13(b) shows the general categorization of user jobs when using user-specific baselines per user. The job group 1 consists of jobs with a lower number of hardware errors, and the environment logs follow the same z-score trend as the baseline jobs. The job group 2 and 3 both show a larger number of hardware errors. However, job group 3 consists mainly of high-capability jobs when compared to job group 2. From Fig. 6.13(a) and (b), using the mrDMD analysis, we can categorize users and jobs using z-score values, and

the system admins can track the system usage patterns compared to past utilization trends.

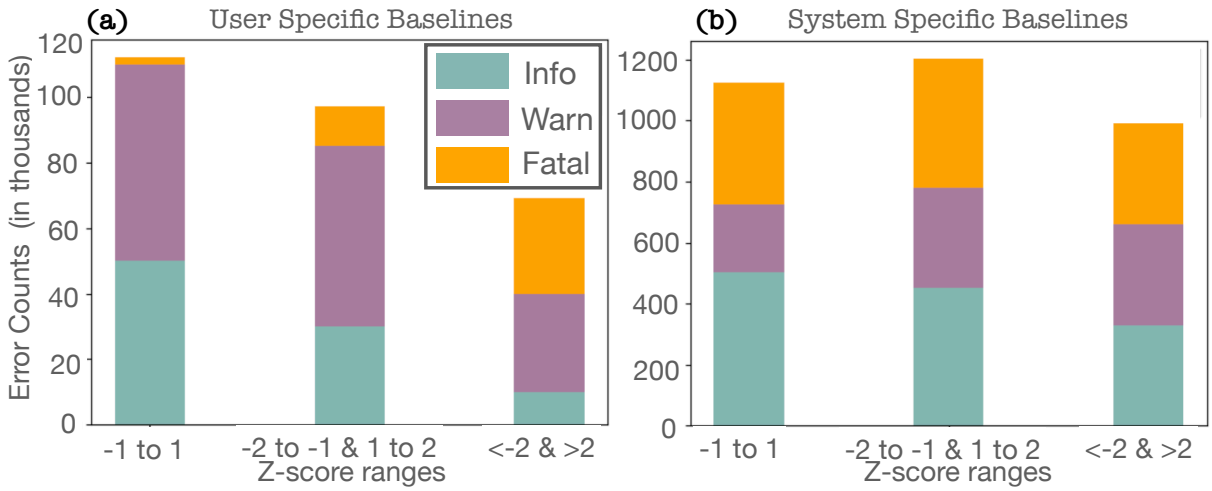

Figure 6.14: Analyzing the supercomputer log data using z-score ranges and plotting the counts of hardware errors at each range for (a) user-specific baselines and (b) system-specific baselines. (a) and (b) show the z-scores values before and after the system reported hardware errors, respectively.

Fig. 6.14 shows the plots for the hardware error counts versus z-score ranges. Here the hardware errors in the dataset are categorized into informational, warning, and fatal (errors linked with system failures) type error messages. Our goal was to check if the environment log information, displayed through z-scores, captured any hardware error trends. Fig. 6.14(a) shows the plot for a group of 150 jobs for 23 users in the system. We compute the z-scores using a user-specific baseline for passed job runs. Here we see that the fatal hardware error counts increase as the z-score values move away from the baseline. However, the informational and warning messages are higher in lower z-score ranges. Fig. 6.14(b) uses system-specific baselines to compute z-scores for 500 user jobs. However, when we attempted to find patterns of z-scores linking hardware error categories to specific groups of users or jobs, we could not derive a link. Fig. 6.14(b) shows that the hardware error categories are roughly equal across all z-score values when using system-specific baselines for all user jobs. This demonstrates the usefulness of using user-specific baselines as they help identify and link hardware errors with user jobs.

Fig. 6.15 shows the plots of z-score values versus slots (4 nodes) in the supercomputer

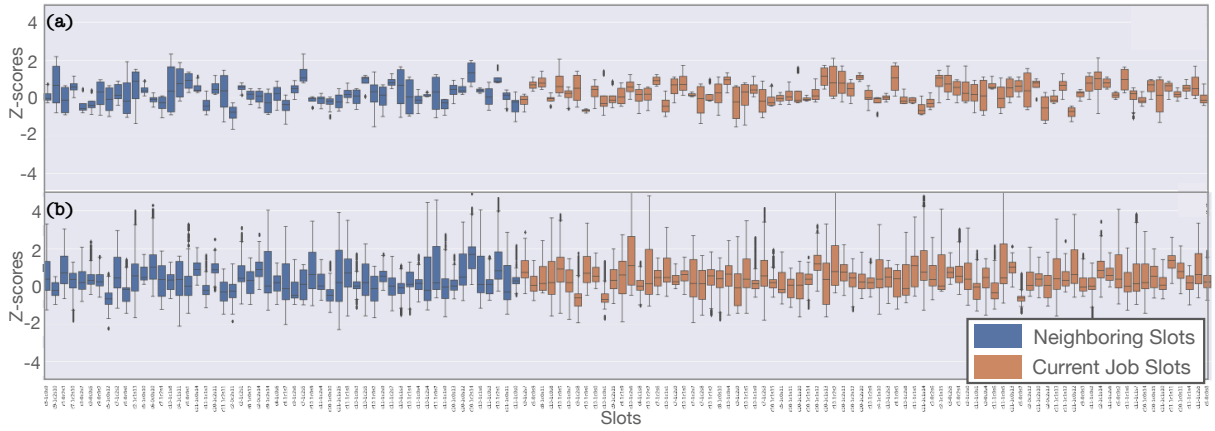

Figure 6.15: Analyzing the supercomputer log data showing z-score values versus slots (4 nodes) for one job run. The top (a) and bottom (b) figures show the z-scores for job slot readings and neighboring slot readings before and after the job run reported hardware errors, respectively.

for one job run. The orange-colored slots are allocated to this job, and the blue belong to the neighboring slots not utilized by the job. Fig. 6.15(a) and Fig. 6.15(b) show the z-scores values before and after the system reported hardware errors, respectively. We can clearly see that z-scores show larger variations (−4 to 4) in values after the job slots reported hardware errors. These hardware errors were memory and critical errors, and the resulting temperature spike affected not only the job slots but also the neighboring slots that reported no errors. Similar trends are visible across multiple jobs and further analysis is needed to identify countermeasures to avoid system heating or overheating.

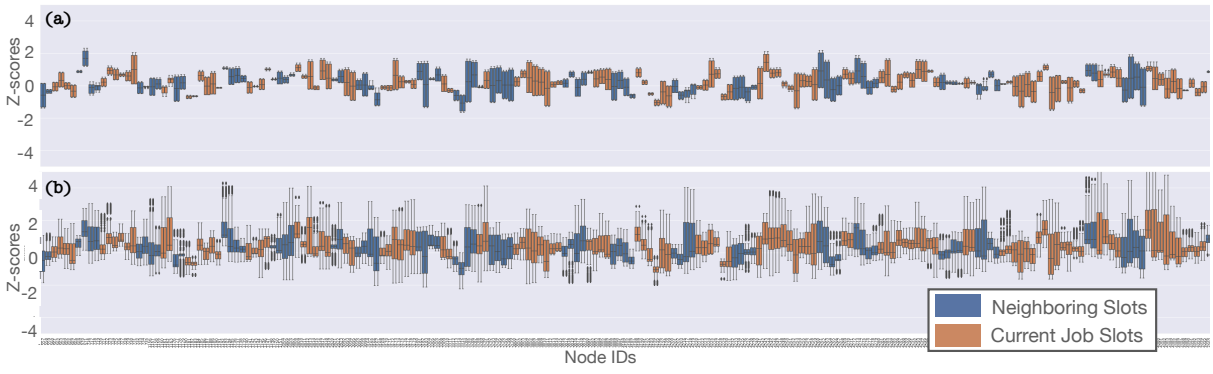

Figure 6.16: Analyzing the supercomputer log data showing z-score values versus nodes for one job run. The top (a) and bottom (b) figures show the z-scores for a subset of job node readings and neighboring node readings before and after the job run reported hardware errors, respectively. The nodes along the x-axis are sorted by their node IDs

Fig. 6.16 shows the plots of z-score values versus nodes in the supercomputer for

the job run shown in Fig. 6.15. The orange-colored nodes are allocated to this job, and the blue belong to the neighboring nodes not utilized by the job. Fig. 6.16(a) and Fig. 6.16(b) show the z-scores values for a subset of job and neighboring nodes before and after the system reported hardware errors, respectively. We sorted the nodes along the x-axis by their node IDs. Similar to Fig. 6.15, z-scores show larger values variations (-4 to 4) after the job nodes reported hardware errors. When we view the z-score values at the finer node granularity, the temperature fluctuations of neighboring nodes show similar variations. Hence, any unprecedented temperature rise in a node may affect its neighbors and damage a large portion of the system.

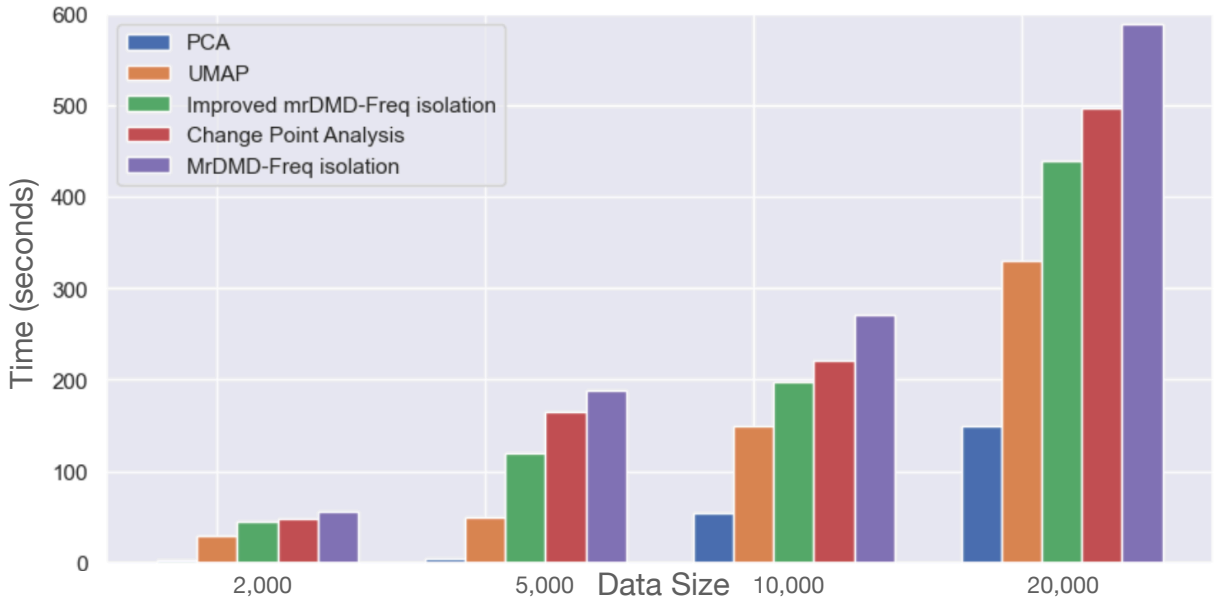

Figure 6.17: A comparison of completion times, showing how performance scales with the data size for $4,300$ readings.

The sampling rate at each level of our mrDMD depends on the level. Lower levels use lower sampling rates, and higher frequency levels sample data at a higher frequency. Since we do not extract high-frequency modes at lower levels, we can use this improvement without changing the final result. At each level of the mrDMD analysis, the algorithm subtracts the slower modes and splits the timeline. We process each of these splits asynchronously during the recursive process, adding a performance improvement, as shown in Fig. 6.17. For this analysis, we have picked some popular time series analysis and clustering methods, including PCA [82], UMAP [115](n_neighbors = 4,

min_dist = 0.1, and metric="euclidean"), change point analysis [166](model="rbf", jump=3, min_size=2, breakpoints=20). We then compare the computation times using 4,300 time series of varying length data size, including $4,300 \times 2000$, $4,300 \times 5000$, $4,300 \times 10000$, and $4,300 \times 20,000$. We performed the experiments on the Cray XC40 supercomputer with two 6-core, 2.4-GHz Intel E5–2620 v3 processors (Intel Haswell architecture) and 256GB of DDR4 memory. The results show that although our method does not outperform PCA and UMAP, it does outperform the change point detection algorithm and the mrDMD algorithm without our performance improvements.

6.6 Conclusion

With significant achievements in computer engineering and the advent of exascale systems, there is a significant increase in monitoring data collected at multiple fidelity levels and varying temporal resolutions. Our work aims to build a holistic visual analytical tool that processes this massive monitoring data, including the hardware logs, job logs, environment logs, syslogs, console log, etc., collected from disparate subsystems and components of a supercomputer system. The tool provides an abstract overview of the systems underlying state and behaviors. The main goal of our work is to consolidate and provide analysis results from multiple types of log data. Environment log data is usually ignored in log analysis mainly because of its massive size and lack of a mechanism to draw interesting patterns promptly. Our work aims to bridge this gap through our modified mrDMD analysis and visual analytics tool. With our improvements to the multiresolution dynamic mode decomposition (mrDMD) algorithm, we are able to promptly extract supercomputer usage and error patterns at varying temporal and spatial resolutions. We identified two types of baselines, representing a normal system behavior, and used them to extract groups of users and jobs that follow similar supercomputer usage patterns. Our data-driven analysis can be easily applied to other large-scale system log data. We plan to extend our work to support the analysis of streaming multivariate time series to deliver results in real time.

Chapter 7
Conclusion

This dissertation has identified critical challenges in error log analysis when applied to visual analytics and has exemplified solutions for several tangible topics. This dissertation also introduces mechanisms that employ multiple error logs to provide a holistic representation of large-scale systems. Faster implementation of functional data analysis algorithms in conjunction with other functional data preprocessing mechanisms enhances log data analysis results' interpretability, usability, and flexibility. The multi-stage analysis of the multifidelity log data using machine learning algorithms helps predict when an application would end with a failure status and the likely error message. In the meantime, auxiliary visualizations provide details about the system environment measurements like power, temperature, etc., indicating the overall impact of the error on the system. The improvements and customization of multiresolution dynamic mode decomposition (mrDMD) analysis with mrDMD spectrum analysis guide the multi-scale analysis by extracting information at multiple resolutions in time. Furthermore, the methodologies are data-driven and applied across systems and domains, providing useful insights and interpretations.

As the large-scale computing systems approach exascale capabilities coupled with increasing demands for faster, more efficient, robust, and reliable platforms to execute mission-critical applications, research is focused on studying large-scale system logs to understand and capture the implicit mechanics of error propagation. Recent works in log analysis have made significant progress in identifying and revealing the traces of error propagation. However, a gap remains to be bridged regarding providing a holistic system view. In our attempt, we harness the capabilities of visual analytics reinforced by robust backend mechanisms through functional data analysis, machine learning,

deep learning, and multi-scale analysis to provide a holistic picture of these large-scale systems and convey the results to the user visually. In addition, our visualization tools incorporate multiple auxiliary views representing various system elements and user feedback in their pipeline. While most error log analyses utilize one or two types of system logs, our analyses utilize multiple system logs, up to five types, collected at varying levels of resolutions both within the system and across time. These considerations have enabled us to better understand the system's functionalities spatially across the system hardware and temporally across days or months. The work in this dissertation helps motivate and pave the way for future research and development toward designing multifidelity machine log analysis through visual analytics.

Appendix A
Online Supplementary Materials

The links below include the repositories of implementations, demo videos, and examples used for each corresponding chapter.

- Chapter 3: `https://github.com/sshilpika/streaming-ms-plot`
- Chapter 4: `https://github.com/argonne-lcf/MELA`
- Chapter 5 and 6: `https://github.com/sshilpika/error-log-analysis`

Appendix B
Appendix for Chapter 6

B.1 Using mrDMD for extraction of spatio-temporal patterns at isolated frequencies ranges

DMD method provides a spatio-temporal decomposition of data into a group of dynamic modes extracted from measurements of a given system in time. DMD can be thought of as a combination of spatial dimensionality reduction techniques such as principal component analysis [82] or proper orthogonal decomposition (POD) [18], and Fourier transforms in time.

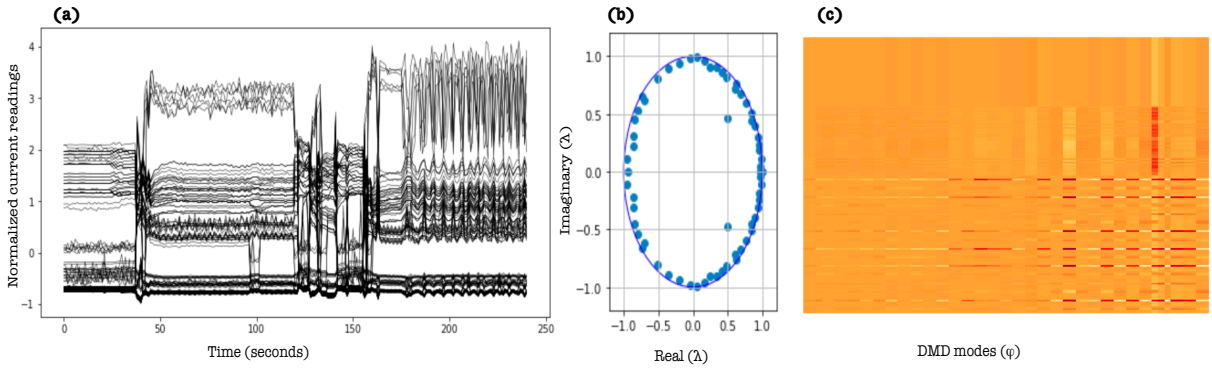

Figure B.1: The figure shows sample of the environment log data (a), specifically the current readings which are z-score normalized, (b) the eigenvalues λ of the decomposition are shown as complex values, and (c) show the DMD modes for the readings in (a)

Consider measurements taken from p observable locations at times Δt, where each of m snapshots of measurements form a column vector x with p elements each. The Fig. B.1(a), shows z-score normalized environment readings for current (original unit in amperes) along the y-axis. These measurements are obtained at sample rate of 10 seconds totaling $m = 250$ snapshots of $p = 160$ measurements each.

We can construct two $p \times (m-1)$ raw data matrix from m snapshots in time:

$$\mathbf{X} = \begin{bmatrix} | & | & | & & | \\ \mathbf{x}_1 & \mathbf{x}_2 & \mathbf{x}_3 & \dots & \mathbf{x}_{m-1} \\ | & | & | & & | \end{bmatrix} \tag{B.1}$$

$$\mathbf{X}' = \begin{bmatrix} | & | & | & & | \\ \mathbf{x}_2 & \mathbf{x}_3 & \mathbf{x}_4 & \dots & \mathbf{x}_m \\ | & | & | & & | \end{bmatrix} \tag{B.2}$$

Here, X and X' have largely overlapping data, with columns of X' being shifted one Δt from those in X. Assuming there exists an unknown linear operator A giving: $X' = AX$.

The eigendecomposition of A gives the dynamic mode decomposition of the data matrix pair X and X'.

In our examples of environment measurements for a data matrix X has $n << m$. We have fewer measurements than the snapshots in time. Therefore, SVD of X produces k non-zero singular values, where k is the smaller of n and $m-1$. Thus restricting the maximum number of DMD modes and eigenvalues to a much smaller number given by n in our case. Therefore, making it unable to capture the dynamics over the larger value of m snapshots in time [15, 61]. We augment the resulting matrix by appending the time-shifted versions of the measurements' snapshots to solve this issue. This technique is inspired by the Hankel matrix as constructed in the Eigenvalue Realization Algorithm [61, 83]. If we use $k-1$ time-shifted snapshots, the augmented X matrix looks as follows:

$$\mathbf{X}_{aug} = \begin{bmatrix} | & | & & | \\ \mathbf{x}_1 & \mathbf{x}_2 & \dots & \mathbf{x}_{m-k} \\ | & | & & | \\ | & | & & | \\ \mathbf{x}_2 & \mathbf{x}_3 & \dots & \mathbf{x}_{m-k+1} \\ | & | & & | \\ . & & & \\ . & & & \\ | & | & & | \\ \mathbf{x}_h & \mathbf{x}_{h+1} & \dots & \mathbf{x}_{m-1} \\ | & | & & | \end{bmatrix} \tag{B.3}$$

and we obtain $Xaug'$ matrix similarly. DMD is applied on $Xaug$ and $Xaug'$ instead of X and X'. The spatial DMD modes $\mathbf{\Phi}$ is now a $kn \times m$ matrix. Since the new data matrices are time-shifted stacks, the results also reflect this property giving stacks of h repeated DMD modes. Fig. B.1(b) and (c), give the eigenvalues and DMD mode results. Fig. B.1(b), shows the real and imaginary values plotted along the x-axis and y-axis, respectively. The values along the unit circle represent stable modes, and the ones within and outside represent the rate of decay and growth, respectively.

The multiresolution dynamic mode decomposition (mrDMD) [95] is an extension to DMD using which multi-scale spatio-temporal features are extracted and approximate dynamical models are formulated. In other words, using the mrDMD approach, we can implement principled decomposition of the data at multiple temporal scales. Here, the temporal scales are split at predefined intervals. In our examples, we split it at intervals starting from years, months, weeks, days, job duration, hours, and ending at minutes. After each temporal split, we perform DMD extraction and remove the slower DMD modes from the extracted DMD modes. The slower modes or background modes are DMD eigenvalues that fall below a predefined user-chosen threshold radius. Once the slower DMD modes are removed, the filtered DMD results are used for the next DMD extraction at a smaller or shorter temporal scale.

Step 1: We use the mrDMD power spectrum to identify frequency ranges where the magnitude of the mrDMD power represent high energy. The power of each DMD mode $\mathbf{\Phi_i}$ is given by:

$$\mathbf{P_i} = |\mathbf{\Phi_i}|_2^2 \tag{B.4}$$

Each spatial mrDMD mode $\mathbf{\Phi_i}$ is associated with a corresponding eigenvalue $\lambda_\mathbf{i}$, where the magnitude and phase component of $\lambda_\mathbf{i}$ represents the growth/decay, and frequency of oscillation respectively.

$$\mathbf{f_i} = \left| \frac{\mathbf{imag}(\omega_\mathbf{i})}{2\pi} \right| \tag{B.5}$$

where $\omega_\mathbf{i} = \log(\lambda_i)/\Delta t$, where t is the time. The usefulness of extracting data frequencies based on the DMD spectrum has been demonstrated in the works by Brunton et al. [15] on electrodes recordings of brain activity over minutes to hours. However, the work employs traditional DMD, and we use mrDMD to systematically and recursively removes slower temporal or spatial DMD modes from the data [29, 94]. At each temporal scale of the mrDMD computation, the DMD spectrum is computed using Eq. B.4. In work by Brunton et al., the frequency ranges of interest and the abnormal DMD power magnitudes are known before analysis. We extract the mrDMD modes in the frequency ranges where the DMD power shows high energy compared to that of the chosen baseline measurements representing normal operation. Fig. B.2, show the mrDMD spectrum for a selected set of baseline readings (shown in blue) and readings that deviate from the baseline. The example chosen here has 3000 time points and 48 environment readings, 22 of which represent baseline readings. The time ranges chosen for multiresolution splits are 24 hours, every 6 hours, and every hour. We can see that the mrDMD power extracted at each time resolution shows larger mrDMD power for readings that deviate from the baseline.

Step 2: Once the mrDMD is computed, we can analyze a specific frequency range. The mrDMD modes lying within a frequency range are averaged to extract the mean mode magnitude. The mean mode is subtracted from the baseline readings. Then the z-scores of relative changes in mean mode magnitude is computed. Fig. B.3, shows the z-scores of relative changes in the mean mode magnitudes computed for frequency

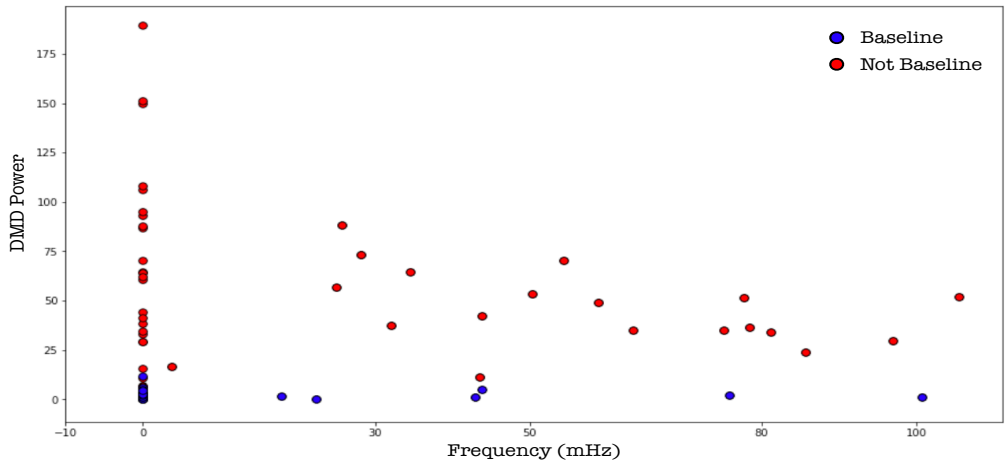

Figure B.2: The figure shows the mrDMD spectrum power versus the frequency (mHz). The red dots represent environment log readings other than the baseline and the blue dots represent the baseline readings.

range $20-60$ mHz (shown in pink), and > 80 mHz (shown in cyan). The figure shows that the environment readings representing normal jobs, Fig. B.3(a),(b), have z-scores that are closer to the baseline. The environment readings corresponding to the job that failed to execute, Fig. B.3(c), shows z-scores with larger deviations from the baseline.

Step 3: Our visual analytics front-end allows user to customize the frequency range, mrDMD threshold range, and sampling rate selections. The mrDMD spectrum results can be plotted on the abstract view or the glyph layout as in Chapter 6. By doing so, we identify peculiar temporal and spatial ranges that cross the baseline setting of normal system operation. A challenge in the implementation of this method would be the computation cost. To solve this problem, we reduce the sampling rate of the time-series data for macro-scale resolution (years) and increase it as we approach micro-scale resolution (minutes) as suggested in [95].

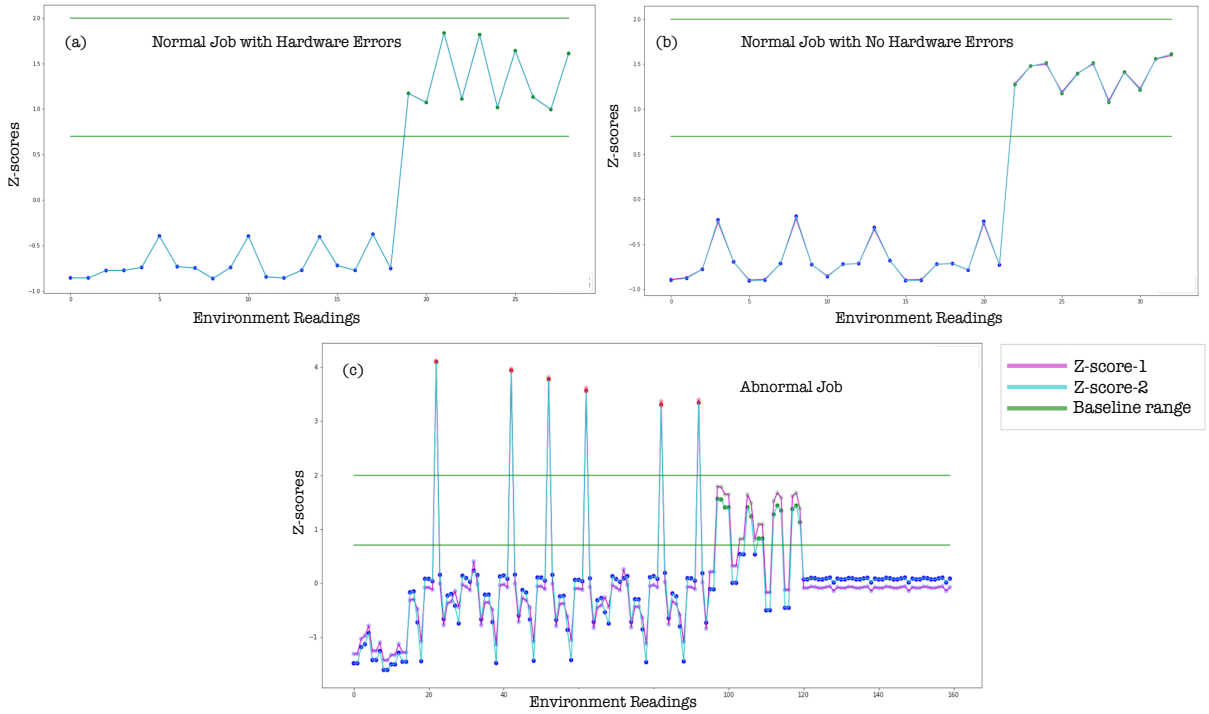

Figure B.3: The figure shows the z-scores of relative changes in the mean mode magnitudes computed for frequency range $20-60$ mHz (shown in pink) and >80mHz (shown in cyan). The green line represents the value-range of the baseline readings. (a) and (b), show z-scores for environment readings corresponding to nodes that executed jobs successfully, and (c), shows z-scores for the environment readings corresponding to a node where the job failed. From the environment readings representing normal jobs ((a) and (b)) we see z-scores are closer to the baseline. The job that failed to execute (c), shows environment readings with z-scores having larger deviations from the baseline.

References

[1] J. Ah-Pine. An efficient and effective generic agglomerative hierarchical clustering approach. *J. Mach. Learn. Res.*, 19(1):1615–1658, Jan. 2018.

[2] M. Ali, A. Alqahtani, M. W. Jones, and X. Xie. Clustering and classification for time series data in visual analytics: A survey. *IEEE Access*, 7, 2019.

[3] G. Aneiros, R. Cao, R. Fraiman, C. Genest, and P. Vieu. Recent advances in functional data analysis and high-dimensional statistics. *J. multivariate analysis*, 170:3–9, 2019.

[4] Argonne Leadership Computing Facility, Argonne National Laboratory. Status of theta supercomputer. `https://status.alcf.anl.gov/theta/activity`, 2021. Accessed: 2021-4-19.

[5] Argonne National Laboratory. Argonne Leadership Computing Facility (ALCF) Theta Supercomputer. `https://www.alcf.anl.gov/theta`. Accessed: January 23, 2021.

[6] A. Avizienis, J.-C. Laprie, B. Randell, and C. Landwehr. Basic concepts and taxonomy of dependable and secure computing. *IEEE Trans. Dependable Secur. Comput.*, 1(1):11–33, jan 2004. doi: 10.1109/TDSC.2004.2

[7] L. N. Bairavasundaram. *Characteristics, Impact, and Tolerance of Partial Disk Failures*. PhD thesis, University of Wisconsin at Madison, USA, 2008. AAI3348701.

[8] D. A. Bhanage, A. V. Pawar, and K. Kotecha. It infrastructure anomaly detection and failure handling: A systematic literature review focusing on datasets, log preprocessing, machine & deep learning approaches and automated tool. *IEEE Access*, 9:156392–156421, 2021. doi: 10.1109/ACCESS.2021.3128283

[9] M. Bilal, M. Rizwan, S. Saleem, M. M. Khan, M. S. Alkatheir, and M. Alqarni. Automatic seizure detection using multi-resolution dynamic mode decomposition. *IEEE Access*, 7:61180–61194, 2019. doi: 10.1109/ACCESS.2019.2915609

[10] D. M. Blei, A. Y. Ng, and M. I. Jordan. Latent dirichlet allocation. *J. Mach. Learn. Res.*, 3(null):993–1022, Mar. 2003.

[11] M. Bostock, V. Ogievetsky, and J. Heer. D3 data-driven documents. *IEEE Transactions on Visualization and Computer Graphics*, 17(12):2301–2309, 2011. doi: 10.1109/TVCG.2011.185

[12] N. Bourassa, W. Johnson, J. Broughton, D. M. Carter, S. Joy, R. Vitti, and P. Seto. Operational data analytics: Optimizing the national energy research scientific computing center cooling systems. In *Proceedings of the 48th International Conference on Parallel Processing: Workshops*, ICPP 2019. Association for Computing Machinery, New York, NY, USA, 2019. doi: 10.1145/3339186.3339210

[13] A. Brown, A. Tuor, B. Hutchinson, and N. Nichols. Recurrent neural network attention mechanisms for interpretable system log anomaly detection. In *Proceedings of the First Workshop on Machine Learning for Computing Systems*, MLCS'18. Association for Computing Machinery, New York, NY, USA, 2018. doi: 10.1145/3217871.3217872

[14] M. Brundage, S. Chandrasegaran, X. Zhang, and K.-L. Ma. Using text visualization to aid analysis of machine maintenance logs, 2020-04-30 2020.

[15] B. W. Brunton, L. A. Johnson, J. G. Ojemann, and J. N. Kutz. Extracting spatial–temporal coherent patterns in large-scale neural recordings using dynamic mode decomposition. *Journal of Neuroscience Methods*, 258:1–15, 2016. doi: 10.1016/j.jneumeth.2015.10.010

[16] R. B. Cattell. The scree test for the number of factors. *Multivariate Behav. Res.*, 1(2):245–276, 1966.

[17] T. Chalermarrewong, T. Achalakul, and S. C. W. See. Failure prediction of data centers using time series and fault tree analysis. In *2012 IEEE 18th International Conference on Parallel and Distributed Systems*, pp. 794–799, 2012. doi: 10.1109/ICPADS.2012.129

[18] A. Chatterjee. An introduction to the proper orthogonal decomposition. *Current Science*, 78(7):808–817, 2000.

[19] Y. Chen, X. Yang, Q. Lin, H. Zhang, F. Gao, Z. Xu, Y. Dang, D. Zhang, H. Dong, Y. Xu, H. Li, and Y. Kang. Outage prediction and diagnosis for cloud service systems. In *The World Wide Web Conference*, WWW '19, p. 2659–2665. Association for Computing Machinery, New York, NY, USA, 2019. doi: 10.1145/3308558.3313501

[20] E. Chuah, A. Jhumka, J. C. Browne, B. Barth, and S. Narasimhamurthy. Insights into the diagnosis of system failures from cluster message logs. In *2015 11th European Dependable Computing Conference (EDCC)*, pp. 225–232, 2015. doi: 10.1109/EDCC.2015.19

[21] E. Chuah, S.-h. Kuo, P. Hiew, W.-C. Tjhi, G. Lee, J. Hammond, M. T. Michalewicz, T. Hung, and J. C. Browne. Diagnosing the root-causes of failures from cluster log files. In *2010 International Conference on High Performance Computing*, pp. 1–10, 2010. doi: 10.1109/HIPC.2010.5713159

[22] T. Crnovrsanin, J. Chu, and K.-L. Ma. An incremental layout method for visualizing online dynamic graphs. *J. Graph Algorithms and Applications*, 21(1):55–80, 2017.

[23] W. Dai and M. Genton. Directional outlyingness for multivariate functional data. *Computational Statistics & Data Analysis*, 03 2019.

[24] W. Dai and M. G. Genton. Multivariate functional data visualization and outlier detection. *J. Computational and Graphical Statistics*, 27(4):923–934, 2018.

[25] A. Das, F. Mueller, P. Hargrove, E. Roman, and S. Baden. Doomsday: Predicting which node will fail when on supercomputers. In *SC18: International Conference for High Performance Computing, Networking, Storage and Analysis*, pp. 108–121, 2018. doi: 10.1109/SC.2018.00012

[26] A. Das, F. Mueller, and B. Rountree. Aarohi: Making real-time node failure prediction feasible. In *2020 IEEE International Parallel and Distributed Processing Symposium (IPDPS)*, pp. 1092–1101, 2020. doi: 10.1109/IPDPS47924.2020.00115

[27] A. Das, F. Mueller, C. Siegel, and A. Vishnu. Desh: Deep learning for system health prediction of lead times to failure in hpc. In *Proceedings of the 27th International Symposium on High-Performance Parallel and Distributed Computing*, HPDC '18, p. 40–51. Association for Computing Machinery, New York, NY, USA, 2018. doi: 10.1145/3208040.3208051

[28] A. Dasgupta, D. L. Arendt, L. R. Franklin, P. C. Wong, and K. A. Cook. Human factors in streaming data analysis: Challenges and opportunities for information visualization. *Computer Graphics Forum*, 37(1):254–272, 2018.

[29] I. Daubechies. *Ten Lectures on Wavelets*. Society for Industrial and Applied Mathematics, 1992. doi: 10.1137/1.9781611970104

[30] S. T. M. Dawson, M. S. Hemati, M. O. Williams, and C. W. Rowley. Characterizing and correcting for the effect of sensor noise in the dynamic mode decomposition. *Experiments in Fluids*, 57:1–19, 2014.

[31] S. Di, R. Gupta, M. Snir, E. Pershey, and F. Cappello. Logaider: A tool for mining potential correlations of hpc log events. In *Proceedings of the 17th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing*, CCGrid '17, p. 442–451. IEEE Press, 2017. doi: 10.1109/CCGRID.2017.18

[32] M. Du, F. Li, G. Zheng, and V. Srikumar. Deeplog: Anomaly detection and diagnosis from system logs through deep learning. In *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*, CCS '17, p. 1285–1298. Association for Computing Machinery, New York, NY, USA, 2017. doi: 10.1145/3133956.3134015

[33] N. El-Sayed, I. A. Stefanovici, G. Amvrosiadis, A. A. Hwang, and B. Schroeder. Temperature management in data centers: Why some (might) like it hot. *SIGMETRICS Perform. Eval. Rev.*, 40(1):163–174, jun 2012. doi: 10.1145/2318857.2254778

[34] N. El-Sayed, H. Zhu, and B. Schroeder. Learning from failure across multiple clusters: A trace-driven approach to understanding, predicting, and mitigating job terminations. In *2017 IEEE 37th International Conference on Distributed Computing Systems (ICDCS)*, pp. 1333–1344, 2017. doi: 10.1109/ICDCS.2017.317

[35] O. ElTayeby and W. Dou. A survey on interaction log analysis for evaluating exploratory visualizations. In *Proceedings of the Sixth Workshop on Beyond Time and Errors on Novel Evaluation Methods for Visualization*, BELIV '16, p. 62–69. Association for Computing Machinery, New York, NY, USA, 2016. doi: 10.1145/2993901.2993912

[36] F. Chollet. Keras. `https://keras.io`. Accessed: January 23, 2021.

[37] F. Shilpika. Toward an In-Depth Analysis of Multifidelity System Logs. `https://youtu.be/exo6aim4uRI`. Accessed: January 23, 2021.

[38] F. Ferraty and P. Vieu. *Nonparametric functional data analysis: Theory and practice*. Springer Science & Business Media, 2006.

[39] D. D. Fong, A. Knoesen, M. Motamedi, T. O'Neill, and S. Ghiasi. Recovering the fetal signal in transabdominal fetal pulse oximetry. *Smart Health*, 9:23–36, 2018.

[40] A. L. N. Fred and A. K. Jain. Combining multiple clusterings using evidence accumulation. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 27(6):835–850, 2005. doi: 10.1109/TPAMI.2005.113

[41] K. P. F.R.S. Liii. on lines and planes of closest fit to systems of points in space. *The London, Edinburgh, and Dublin Philosophical Magazine and Journal of Science*, 2(11):559–572, 1901. doi: 10.1080/14786440109462720

[42] X. Fu, R. Ren, J. Zhan, W. Zhou, Z. Jia, and G. Lu. Logmaster: Mining event correlations in logs of large-scale cluster systems. In *2012 IEEE 31st Symposium on Reliable Distributed Systems*, pp. 71–80, 2012. doi: 10.1109/SRDS.2012.40

[43] T. Fujiwara, J. Chou, S. Shilpika, P. Xu, L. Ren, and K. Ma. An incremental dimensionality reduction method for visualizing streaming multidimensional data. *IEEE Trans. on Visualization and Computer Graphics*, 26(1):418–428, 2020.

[44] T. Fujiwara, O.-H. Kwon, and K.-L. Ma. Supporting analysis of dimensionality reduction results with contrastive learning. *IEEE Trans. Vis. Comput. Graph*, 26(1):45–55, 2020.

[45] T. Fujiwara, J. K. Li, M. Mubarak, C. Ross, C. D. Carothers, R. B. Ross, and K.-L. Ma. A visual analytics system for optimizing the performance of large-scale networks in supercomputing systems. *Visual Informatics*, 2(1):98–110, 2018. Proceedings of PacificVAST 2018. doi: 10.1016/j.visinf.2018.04.010

[46] T. Fujiwara, P. Malakar, K. Reda, V. Vishwanath, M. E. Papka, and K.-L. Ma. A visual analytics system for optimizing communications in massively parallel applications. In *2017 IEEE Conference on Visual Analytics Science and Technology (VAST)*, pp. 59–70, 2017. doi: 10.1109/VAST.2017.8585646

[47] T. Fujiwara, Shilpika, N. Sakamoto, J. Nonaka, K. Yamamoto, and K. Ma. A visual analytics framework for reviewing multivariate time-series data with dimensionality reduction. *IEEE Trans. Vis. Comput. Graph*, 27(2):1601–1611, 2021.

[48] A. Gainaru, F. Cappello, J. Fullop, S. Trausan-Matu, and W. Kramer. Adaptive event prediction strategy with dynamic time window for large-scale hpc systems. In *Managing Large-Scale Systems via the Analysis of System Logs and the Application of Machine Learning Techniques*, SLAML '11. Association for Computing Machinery, New York, NY, USA, 2011. doi: 10.1145/2038633.2038637

[49] A. Gainaru, F. Cappello, and W. Kramer. Taming of the shrew: Modeling the normal and faulty behaviour of large-scale hpc systems. In *Proceedings of the 2012 IEEE 26th International Parallel and Distributed Processing Symposium*, IPDPS '12, p. 1168–1179. IEEE Computer Society, USA, 2012. doi: 10.1109/IPDPS.2012.107

[50] A. Gainaru, F. Cappello, M. Snir, and W. Kramer. Fault 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*, pp. 1–11, 2012. doi: 10.1109/SC.2012.57

[51] A. Gainaru, F. Cappello, S. Trausan-Matu, and B. Kramer. Event log mining tool for large scale hpc systems. In *Proceedings of the 17th International Conference on Parallel Processing - Volume Part I*, Euro-Par'11, p. 52–64. Springer-Verlag, Berlin, Heidelberg, 2011.

[52] S. Ganguly, A. Consul, A. Khan, B. Bussone, J. Richards, and A. Miguel. A practical approach to hard disk failure prediction in cloud platforms: Big data model for failure management in datacenters. In *2016 IEEE Second International Conference on Big Data Computing Service and Applications (BigDataService)*, pp. 105–116, 2016. doi: 10.1109/BigDataService.2016.10

[53] M. Gavish and D. L. Donoho. The optimal hard threshold for singular values is $4/\sqrt{3}$. *IEEE Transactions on Information Theory*, 60(8):5040–5053, 2014. doi: 10.1109/TIT.2014.2323359

[54] A. Geist and D. A. Reed. A survey of high-performance computing scaling challenges. *Int. J. High Perform. Comput. Appl.*, 31(1):104–113, jan 2017. doi: 10.1177/1094342015597083

[55] C. Gormley and Z. Tong. *Elasticsearch: The Definitive Guide*. O'Reilly Media, Inc., 1st ed., 2015.

[56] Government of Canada. Canadian Weather Historical Data. `https://climate.weather.gc.ca/historical_data/search_historic_data_e.html`. Accessed: August 22, 202.

[57] M. Grinberg. *Flask web development: developing web applications with python*. " O'Reilly Media, Inc.", 2018.

[58] H. S. Gunawi, R. O. Suminto, R. Sears, C. Golliher, S. Sundararaman, X. Lin, T. Emami, W. Sheng, N. Bidokhti, C. McCaffrey, D. Srinivasan, B. Panda, A. Baptist, G. Grider, P. M. Fields, K. Harms, R. B. Ross, A. Jacobson, R. Ricci, K. Webb, P. Alvaro, H. B. Runesha, M. Hao, and H. Li. Fail-slow at scale: Evidence of hardware performance faults in large production systems. *ACM Trans. Storage*, 14(3), oct 2018. doi: 10.1145/3242086

[59] H. Guo, S. Di, R. Gupta, T. Peterka, and F. Cappello. La VALSE: Scalable Log Visualization for Fault Characterization in Supercomputers. In H. Childs and F. Cucchietti, eds., *Eurographics Symposium on Parallel Graphics and Visualization*. The Eurographics Association, 2018. doi: 10.2312/pgv.20181099

[60] L. Guo, D. Li, I. Laguna, and M. Schulz. Fliptracker: Understanding natural error resilience in hpc applications. In *SC18: International Conference for High Performance Computing, Networking, Storage and Analysis*, pp. 94–107, 2018. doi: 10.1109/SC.2018.00011

[61] J. H. Tu, C. W. Rowley, D. M. Luchtenburg, S. L. Brunton, and J. Nathan Kutz. On dynamic mode decomposition: Theory and applications. *Journal of Computational Dynamics*, 1(2):391–421, 2014. doi: 10.3934/jcd.2014.1.391

[62] M. Halkidi, Y. Batistakis, and M. Vazirgiannis. On clustering validation techniques. *J. Intell. Inf. Syst.*, 17(2–3):107–145, Dec. 2001. doi: 10.1023/A:1012801612483

[63] P. Hall and M. Hosseini-Nasab. Theory for high-order bounds in functional principal components analysis. In *Mathematical Proc. Cambridge Philosophical Society*, vol. 146, p. 225, 2009.

[64] H. Hamooni, B. Debnath, J. Xu, H. Zhang, G. Jiang, and A. Mueen. Logmine: Fast pattern recognition for log analytics. In *Proceedings of the 25th ACM International on Conference on Information and Knowledge Management*, CIKM '16, p. 1573–1582. Association for Computing Machinery, New York, NY, USA, 2016. doi: 10.1145/2983323.2983358

[65] K. Harms, T. Leggett, B. Allen, S. Coghlan, M. Fahey, C. Holohan, G. McPheeters, and P. Rich. Theta: Rapid installation and acceptance of an XC40 KNL system. *Concurrency and Computation: Practice and Experience*, 30(1), 2018. e4336 cpe.4336.

[66] A. N. Harutyunyan, A. V. Poghosyan, N. M. Grigoryan, and M. A. Marvasti. Abnormality analysis of streamed log data. In *2014 IEEE Network Operations and Management Symposium (NOMS)*, pp. 1–7, 2014. doi: 10.1109/NOMS.2014.6838292

[67] S. He, P. He, Z. Chen, T. Yang, Y. Su, and M. R. Lyu. A survey on automated log analysis for reliability engineering. *ACM Comput. Surv.*, 54(6), jul 2021. doi: 10.1145/3460345

[68] M. S. Hemati, M. O. Williams, and C. W. Rowley. Dynamic mode decomposition for large and streaming datasets. *Physics of Fluids*, 26:111701, 2014.

[69] R. Higdon. Generalized additive models. In W. Dubitzky, O. Wolkenhauer, K.-H. Cho, and H. Yokota, eds., *Encyclopedia of Systems Biology*, pp. 814–815. Springer, 2013.

[70] S. L. Ho and M. Xie. The use of arima models for reliability forecasting and analysis. *Comput. Ind. Eng.*, 35(1–2):213–216, Oct. 1998. doi: 10.1016/S0360-8352(98)00066-7

[71] S. Hochreiter and J. Schmidhuber. Long short-term memory. *Neural Comput.*, 9(8):1735–1780, Nov. 1997. doi: 10.1162/neco.1997.9.8.1735

[72] L. Horváth and P. Kokoszka. *Inference for Functional Data with Applications*. Springer, 2012.

[73] A. A. Hwang, I. A. Stefanovici, and B. Schroeder. Cosmic rays don't strike twice: Understanding the nature of dram errors and the implications for system design. *SIGARCH Comput. Archit. News*, 40(1):111–122, mar 2012. doi: 10.1145/2189750.2150989

[74] R. J. Hyndman and M. Shahid Ullah. Robust forecasting of mortality and fertility rates: A functional data approach. *Computational Statistics & Data Analysis*, 51(10):4942–4956, 2007.

[75] R. J. Hyndman and H. L. Shang. Rainbow plots, bagplots, and boxplots for functional data. *J. Computational and Graphical Statistics*, 19(1):29–45, 2010.

[76] InfluxData. InfluxDB Documentation. `https://docs.influxdata.com/influxdb`. Accessed: January 23, 2021.

[77] S. Jain, I. Singh, A. Chandra, Z.-L. Zhang, and G. Bronevetsky. Extracting the textual and temporal structure of supercomputing logs. In *2009 International Conference on High Performance Computing (HiPC)*, pp. 254–263, 2009. doi: 10.1109/HIPC.2009.5433202

[78] D. Jauk, D. Yang, and M. Schulz. Predicting faults in high performance computing systems: An in-depth survey of the state-of-the-practice. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, SC '19. Association for Computing Machinery, New York, NY, USA, 2019. doi: 10.1145/3295500.3356185

[79] D. Jauk, D. Yang, and M. Schulz. Predicting faults in high performance computing systems: An in-depth survey of the state-of-the-practice. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, SC '19. Association for Computing Machinery, New York, NY, USA, 2019. doi: 10.1145/3295500.3356185

[80] T. Jia, P. Chen, L. Yang, Y. Li, F. Meng, and J. Xu. An approach for anomaly diagnosis based on hybrid graph model with logs for distributed services. In *2017 IEEE International Conference on Web Services (ICWS)*, pp. 25–32, 2017. doi: 10.1109/ICWS.2017.12

[81] T. Jia, L. Yang, P. Chen, Y. Li, F. Meng, and J. Xu. Logsed: Anomaly diagnosis through mining time-weighted control flow graph in logs. In *2017 IEEE 10th International Conference on Cloud Computing (CLOUD)*, pp. 447–455, 2017. doi: 10.1109/CLOUD.2017.64

[82] I. Jolliffe. *Principal Component Analysis*, pp. 1094–1096. Springer Berlin Heidelberg, Berlin, Heidelberg, 2011. doi: 10.1007/978-3-642-04898-2_455

[83] J.-N. Juang and R. S. Pappa. An eigensystem realization algorithm for modal parameter identification and model reduction. *Journal of Guidance, Control, and Dynamics*, 8(5):620–627, 1985. doi: 10.2514/3.20031

[84] N. M. M. Kalimullah, A. Shelke, and A. Habib. Multiresolution dynamic mode decomposition (mrdmd) of elastic waves for damage localisation in piezoelectric ceramic. *IEEE Access*, 9:120512–120524, 2021. doi: 10.1109/ACCESS.2021.3108440

[85] K. Karhunen. *Zur Spektraltheorie stochastischer Prozesse*. Annales Academiae scientiarum Fennicae. A.1, Mathematica-physica. 1946.

[86] S. Katragadda, R. Gottumukkala, S. Venna, N. Lipari, S. Gaikwad, M. Pusala, J. Chen, C. W. Borst, V. Raghavan, and M. Bayoumi. Vastream: A visual analytics system for fast data streams. In *Proceedings of the Practice and Experience in Advanced Research Computing on Rise of the Machines (Learning)*, PEARC '19. Association for Computing Machinery, New York, NY, USA, 2019. doi: 10.1145/3332186.3332256

[87] S. Kesavan, H. Bhatia, A. Bhatele, S. Brink, O. Pearce, T. Gamblin, P.-T. Bremer, and K.-L. Ma. Scalable comparative visualization of ensembles of call graphs. *IEEE Transactions on Visualization and Computer Graphics*, pp. 1–1, 2021. doi: 10.1109/TVCG.2021.3129414

[88] S. P. Kesavan, T. Fujiwara, J. K. Li, C. Ross, M. Mubarak, et al. A visual analytics framework for reviewing streaming performance data. In *Proc. PacificVis*, pp. 206–215, 2020.

[89] S. Khan, A. Gani, A. W. A. Wahab, M. A. Bagiwa, M. Shiraz, S. U. Khan, R. Buyya, and A. Y. Zomaya. Cloud log forensics: Foundations, state of the art, and future directions. *ACM Comput. Surv.*, 49(1), may 2016. doi: 10.1145/2906149

[90] J. Klinkenberg, C. Terboven, S. Lankes, and M. S. Müller. Data mining-based analysis of hpc center operations. In *2017 IEEE International Conference on Cluster Computing (CLUSTER)*, pp. 766–773, 2017. doi: 10.1109/CLUSTER.2017.23

[91] A. Kneip and K. J. Utikal. Inference for density families using functional principal component analysis. *J. the American Statistical Association*, 96(454):519–542, 2001.

[92] H.-K. Ko, J. Jo, and J. Seo. Progressive Uniform Manifold Approximation and Projection. In *Proc. EuroVis*, pp. 133–137, 2020.

[93] M. Krstajic and D. A. Keim. Visualization of streaming data: Observing change and context in information visualization techniques. In *Proc. BigData*, pp. 41–47, 2013.

[94] J. N. Kutz, S. L. Brunton, B. W. Brunton, and J. L. Proctor. *Dynamic mode decomposition: data-driven modeling of complex systems*. SIAM, 2016.

[95] J. N. Kutz, X. Fu, and S. L. Brunton. Multiresolution dynamic mode decomposition. *SIAM Journal on Applied Dynamical Systems*, 15(2):713–735, 2016. doi: 10.1137/15M1023543

[96] A. N. Laboratory. Argonne leadership computing facility (alcf) theta supercomputer, 2021.

[97] G. Lakner and B. Knudson. *IBM System Blue Gene Solution: Blue Gene/Q System Administration*. IBM Redbooks, 2007.

[98] M. Landauer, F. Skopik, M. Wurzenberger, and A. Rauber. System log clustering approaches for cyber security applications: A survey. *Computers & Security*, 92:101739, 2020. doi: 10.1016/j.cose.2020.101739

[99] J. K. Li, T. Fujiwara, S. P. Kesavan, C. Ross, M. Mubarak, C. D. Carothers, R. B. Ross, and K.-L. Ma. A visual analytics framework for analyzing parallel and distributed computing applications. In *2019 IEEE Visualization in Data Science (VDS)*, pp. 1–9, 2019. doi: 10.1109/VDS48975.2019.8973380

[100] J. K. Li and K. Ma. P5: portable progressive parallel processing pipelines for interactive data analysis and visualization. *IEEE Trans. Vis. Comput. Graph.*, 26(1):1151–1160, 2020. doi: 10.1109/TVCG.2019.2934537

[101] J. K. Li and K.-L. Ma. P5: Portable progressive parallel processing pipelines for interactive data analysis and visualization. *IEEE Transactions on Visualization and Computer Graphics*, 26(1):1151–1160, 2020. doi: 10.1109/TVCG.2019.2934537

[102] J. K. Li, M. Mubarak, R. B. Ross, C. D. Carothers, and K.-L. Ma. Visual analytics techniques for exploring the design space of large-scale high-radix networks. In *2017 IEEE International Conference on Cluster Computing (CLUSTER)*, pp. 193–203, 2017. doi: 10.1109/CLUSTER.2017.26

[103] Y. Liang, Y. Zhang, H. Xiong, and R. Sahoo. An adaptive semantic filter for blue gene/l failure log analysis. In *2007 IEEE International Parallel and Distributed Processing Symposium*, pp. 1–8, 2007. doi: 10.1109/IPDPS.2007.370635

[104] Q. Lin, K. Hsieh, Y. Dang, H. Zhang, K. Sui, Y. Xu, J.-G. Lou, C. Li, Y. Wu, R. Yao, M. Chintalapati, and D. Zhang. Predicting node failure in cloud service systems. In *Proceedings of the 2018 26th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, ESEC/FSE 2018, p. 480–490. Association for Computing Machinery, New York, NY, USA, 2018. doi: 10.1145/3236024.3236060

[105] X. Lin, P. Wang, and B. Wu. Log analysis in cloud computing environment with hadoop and spark. In *2013 5th IEEE International Conference on Broadband Network & Multimedia Technology*, pp. 273–276, 2013. doi: 10.1109/ICBNMT.2013.6823956

[106] S. Liu, J. Yin, X. Wang, W. Cui, K. Cao, and J. Pei. Online visual analytics of text streams. *IEEE Trans. on Visualization and Computer Graphics*, 22(11):2451–2466, 2016.

[107] M. Loève. Fonctions aléatoires à décomposition orthogonale exponentielle. *La Revue Scientifique*, 84:159–162, 1946.

[108] LOGPAI. A platform for log analytics powered by AI. `https://www.logpai.com`. Accessed: April 3, 2023.

[109] J. Lu, F. Li, L. Li, and X. Feng. Cloudraid: Hunting concurrency bugs in the cloud via log-mining. In *Proceedings of the 2018 26th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, ESEC/FSE 2018, p. 3–14. Association for Computing Machinery, New York, NY, USA, 2018. doi: 10.1145/3236024.3236071

[110] S. Lu, B. Rao, X. Wei, B. Tak, L. Wang, and L. Wang. Log-based abnormal task detection and root cause analysis for spark. In *2017 IEEE International Conference on Web Services (ICWS)*, pp. 389–396, 2017. doi: 10.1109/ICWS.2017.135

[111] J. Mann and J. N. Kutz. Dynamic mode decomposition for financial trading strategies. *Quantitative Finance*, 16:1643 – 1655, 2015.

[112] K. Manohar, E. Kaiser, S. L. Brunton, and J. N. Kutz. Optimized sampling for multiscale dynamics. *Multiscale Modeling & Simulation*, 17(1):117–136, 2019. doi: 10.1137/17M1162366

[113] A. M. Martinez and A. C. Kak. Pca versus lda. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 23(2):228–233, 2001. doi: 10.1109/34.908974

[114] C. D. Martino, S. Jha, W. Kramer, Z. Kalbarczyk, and R. K. Iyer. Logdiver: A tool for measuring resilience of extreme-scale systems and applications. In *Proceedings*

of the 5th Workshop on Fault Tolerance for HPC at EXtreme Scale, FTXS '15, p. 11–18. Association for Computing Machinery, New York, NY, USA, 2015. doi: 10.1145/2751504.2751511

[115] L. McInnes, J. Healy, and J. Melville. UMAP: Uniform manifold approximation and projection for dimension reduction. *arXiv:1802.03426*, 2018.

[116] L. McInnes, J. Healy, N. Saul, and L. Großberger. Umap: Uniform manifold approximation and projection. *Journal of Open Source Software*, 3(29):861, 2018. doi: 10.21105/joss.00861

[117] L. Millán-Roures, I. Epifanio, and V. Martínez. Detection of anomalies in water networks by functional data analysis. *Mathematical Problems in Engineering*, 2018:5129735, 2018.

[118] H. Miyazaki, Y. Kusano, N. Shinjou, F. Shoji, M. Yokokawa, and T. Watanabe. Overview of the K computer system. *Fujitsu Scientific & Technical Journal*, 48(3):302–309, 2012.

[119] C. Muelder, B. Zhu, W. Chen, H. Zhang, and K. Ma. Visual analysis of cloud computing performance using behavioral lines. *IEEE Trans. Vis. Comput. Graph.*, 22(6):1694–1704, 2016. doi: 10.1109/TVCG.2016.2534558

[120] C. Mueller, D. Gregor, and A. Lumsdaine. Distributed force-directed graph layout and visualization. In *Proceedings of the 6th Eurographics Conference on Parallel Graphics and Visualization*, EGPGV '06, p. 83–90. Eurographics Association, Goslar, DEU, 2006.

[121] K. Nagaraj, C. Killian, and J. Neville. Structured comparative analysis of systems logs to diagnose performance problems. In *Proceedings of the 9th USENIX Conference on Networked Systems Design and Implementation*, NSDI'12, p. 26. USENIX Association, USA, 2012.

[122] N. Nakka, A. Agrawal, and A. Choudhary. Predicting node failure in high performance computing systems from failure and usage logs. In *2011 IEEE International Symposium on Parallel and Distributed Processing Workshops and Phd Forum*, pp. 1557–1566, 2011. doi: 10.1109/IPDPS.2011.310

[123] A. Netti, M. Müller, C. Guillen, M. Ott, D. Tafani, G. Ozer, and M. Schulz. Dcdb wintermute: Enabling online and holistic operational data analytics on hpc systems. In *Proceedings of the 29th International Symposium on High-Performance Parallel and Distributed Computing*, HPDC '20, p. 101–112. Association for Computing Machinery, New York, NY, USA, 2020. doi: 10.1145/3369583.3392674

[124] T. T. Neves, R. M. Martins, D. B. Coimbra, K. Kucher, A. Kerren, et al. Xtreaming: An incremental multidimensional projection technique and its application to streaming data. *arXiv:2003.09017*, 2020.

[125] H. T. Nguyen, A. Bhatele, N. Jain, S. P. Kesavan, H. Bhatia, T. Gamblin, K.-L. Ma, and P.-T. Bremer. Visualizing hierarchical performance profiles of parallel codes using callflow. *IEEE Transactions on Visualization and Computer Graphics*, 27(4):2455–2468, 2021. doi: 10.1109/TVCG.2019.2953746

[126] B. Nie, J. Xue, S. Gupta, C. Engelmann, E. Smirni, and D. Tiwari. Characterizing temperature, power, and soft-error behaviors in data center systems: Insights, challenges, and opportunities. In *2017 IEEE 25th International Symposium on Modeling, Analysis, and Simulation of Computer and Telecommunication Systems (MASCOTS)*, pp. 22–31, 2017. doi: 10.1109/MASCOTS.2017.12

[127] A. Oliner and J. Stearley. What supercomputers say: A study of five system logs. In *37th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN'07)*, pp. 575–584, 2007. doi: 10.1109/DSN.2007.103

[128] B. H. Park, Y. Hui, S. Boehm, R. A. Ashraf, C. Layton, and C. Engelmann. A big data analytics framework for HPC log data: Three case studies using the titan supercomputer log. In *Proc. CLUSTER*, pp. 571–579, 2018.

[129] B. H. Park, S. Hukerikar, R. Adamson, and C. Engelmann. Big data meets HPC log analytics: Scalable approach to understanding systems at extreme scale. In *Proc. CLUSTER*, pp. 758–765, 2017.

[130] B.-H. Park, G. Kora, and A. Geist. RAVEN: RAS data analysis through visually enhanced navigation. Proceedings of Cray User Group Conference, 2010.

[131] T. Patel, Z. Liu, R. Kettimuthu, P. Rich, W. Allcock, and D. Tiwari. Job characteristics on large-scale systems: Long-term analysis, quantification, and implications. In *SC20: International Conference for High Performance Computing, Networking, Storage and Analysis*, pp. 1–17, 2020. doi: 10.1109/SC41405.2020.00088

[132] S. D. Pendergrass, J. N. Kutz, and S. L. Brunton. Streaming gpu singular value and dynamic mode decompositions. *ArXiv*, abs/1612.07875, 2016.

[133] N. Pezzotti, B. P. Lelieveldt, L. van der Maaten, T. Höllt, E. Eisemann, and A. Vilanova. Approximated and user steerable tSNE for progressive visual analytics. *IEEE Trans. on Visualization and Computer Graphics*, 23(7):1739–1752, 2017.

[134] J. L. Proctor, S. L. Brunton, and J. N. Kutz. Dynamic mode decomposition with control. *SIAM Journal on Applied Dynamical Systems*, 15(1):142–161, 2016. doi: 10.1137/15M1013857

[135] J. O. Ramsay. Functional data analysis. In *Encyclopedia of Statistical Sciences*. American Cancer Society, 2006.

[136] J. O. Ramsay and C. J. Dalzell. Some tools for functional data analysis. *J. Royal Stat. Society: Series B (Methodological)*, 53(3):539–561, 1991.

[137] J. O. Ramsay and B. W. Silverman. *Functional Data Analysis*, p. 428. Springer-Verlag New York, 2005.

[138] C. R. Rao. Some statistical methods for comparison of growth curves. *Biometrics*, 14(1):1–17, 1958.

[139] C. A. C. Rincón, J.-F. Pâris, R. Vilalta, A. M. K. Cheng, and D. D. E. Long. Disk failure prediction in heterogeneous environments. In *2017 International Symposium on Performance Evaluation of Computer and Telecommunication Systems (SPECTS)*, pp. 1–7, 2017. doi: 10.23919/SPECTS.2017.8046776

[140] D. A. Ross, J. Lim, R.-S. Lin, and M.-H. Yang. Incremental learning for robust visual tracking. *Int. J. Computer Vision*, 77(1-3):125–141, 2008.

[141] C. W. ROWLEY, I. MEZIĆ, S. BAGHERI, P. SCHLATTER, and D. S. HENNINGSON. Spectral analysis of nonlinear flows. *Journal of Fluid Mechanics*, 641:115–127, 2009. doi: 10.1017/S0022112009992059

[142] F. Salfner, M. Lenk, and M. Malek. A survey of online failure prediction methods. *ACM Comput. Surv.*, 42(3), mar 2010. doi: 10.1145/1670679.1670680

[143] M. Sato. The supercomputer "Fugaku" and Arm-SVE enabled A64FX processor for energy-efficiency and sustained application performance. In *Proc. ISPDC*, pp. 1–5, 2020.

[144] P. J. SCHMID. Dynamic mode decomposition of numerical and experimental data. *Journal of Fluid Mechanics*, 656:5–28, 2010. doi: 10.1017/S0022112010001217

[145] P. J. Schmid. Dynamic mode decomposition of numerical and experimental data. *Journal of Fluid Mechanics*, 656(August):5–28, 2010. doi: 10.1017/s0022112010001217

[146] P. J. Schmid, L. Li, M. P. Juniper, and O. Pust. Applications of the dynamic mode decomposition. *Theoretical and Computational Fluid Dynamics*, 25(1):249–259, Jun 2011. doi: 10.1007/s00162-010-0203-9

[147] B. Schroeder and G. A. Gibson. Disk failures in the real world: What does an mttf of 1, 000, 000 hours mean to you? In *USENIX Conference on File and Storage Technologies*, 2007.

[148] B. Seraphim, S. Palit, K. Srivastava, and E. Poovammal. A survey on machine learning techniques in network intrusion detection system. In *2018 4th International Conference on Computing Communication and Automation (ICCCA)*, pp. 1–5, 2018. doi: 10.1109/CCAA.2018.8777596

[149] H. Shang. A survey of functional principal component analysis. *Advances in Statistical Analysis*, 98:121–142, 2014.

[150] Shilpika, T. Fujiwara, N. Sakamoto, J. Nonaka, and K.-L. Ma. A visual analytics approach for hardware system monitoring with streaming functional data analysis. *IEEE Transactions on Visualization and Computer Graphics*, pp. 1–1, 2022. doi: 10.1109/TVCG.2022.3165348. © 2022 IEEE.

[151] F. Shilpika, B. Lusch, M. Emani, V. Vishwanath, M. E. Papka, and K.-L. Ma. Mela: A visual analytics tool for studying multifidelity hpc system logs. In *2019 IEEE/ACM Industry/University Joint International Workshop on Data-center Automation, Analytics, and Control (DAAC)*, pp. 13–18, 2019. doi: 10.1109/DAAC49578.2019.00008. © 2019 IEEE.

[152] S. Shilpika, B. Lusch, M. Emani, F. Simini, V. Vishwanath, M. E. Papka, and K.-L. Ma. Toward an in-depth analysis of multifidelity high performance computing systems. In *2022 22nd IEEE International Symposium on Cluster, Cloud and Internet Computing (CCGrid)*, pp. 716–725, 2022. doi: 10.1109/CCGrid54584.2022.00081. © 2022 IEEE.

[153] S. Silalahi, R. M. Ijtihadie, T. Ahmad, and H. Studiawan. A survey on logging in distributed system. In *2022 1st International Conference on Information System & Information Technology (ICISIT)*, pp. 7–12, 2022. doi: 10.1109/ICISIT54091.2022.9873095

[154] P. Skibinski and J. Swacha. Fast and efficient log file compression. In *ADBIS Research Communications*, 2007.

[155] F. Skopik, M. Landauer, and M. Wurzenberger. Online log data analysis with efficient machine learning: A review. *IEEE Security & Privacy*, 20(3):80–90, 2022. doi: 10.1109/MSEC.2021.3113275

[156] M. Soualhia, F. Khomh, and S. Tahar. Predicting scheduling failures in the cloud: A case study with google clusters and hadoop on amazon emr. In *2015 IEEE 17th International Conference on High Performance Computing and Communications, 2015 IEEE 7th International Symposium on Cyberspace Safety and Security, and 2015 IEEE 12th International Conference on Embedded Software and Systems*, pp. 58–65, 2015. doi: 10.1109/HPCC-CSS-ICESS.2015.170

[157] V. Sridharan, J. Stearley, N. DeBardeleben, S. Blanchard, and S. Gurumurthi. Feng shui of supercomputer memory: Positional effects in dram and sram faults. In *2013 SC - International Conference for High Performance Computing, Networking, Storage and Analysis*, SC '13. Association for Computing Machinery, New York, NY, USA, 2013. doi: 10.1145/2503210.2503257

[158] C. D. Stolper, A. Perer, and D. Gotz. Progressive visual analytics: User-driven visual exploration of in-progress analytics. *IEEE Trans. on Visualization and Computer Graphics*, 20(12):1653–1662, 2014.

[159] A. Strehl and J. Ghosh. Cluster ensembles — a knowledge reuse framework for combining multiple partitions. *J. Mach. Learn. Res.*, 3(null):583–617, Mar. 2003. doi: 10.1162/153244303321897735

[160] J. Svacina, J. Raffety, C. Woodahl, B. Stone, T. Cerny, M. Bures, D. Shin, K. Frajtak, and P. Tisnovsky. On vulnerability and security log analysis: A systematic literature review on recent trends. In *Proceedings of the International Conference on Research in Adaptive and Convergent Systems*, RACS '20, p. 175–180. Association for Computing Machinery, New York, NY, USA, 2020. doi: 10.1145/3400286.3418261

[161] B. C. Tak, S. Tao, L. Yang, C. Zhu, and Y. Ruan. Logan: Problem diagnosis in the cloud using log-based reference models. In *2016 IEEE International Conference on Cloud Engineering (IC2E)*, pp. 62–67, 2016. doi: 10.1109/IC2E.2016.12

[162] L. Tan and J. Jiang. Chapter 2 - signal sampling and quantization. In L. Tan and J. Jiang, eds., *Digital Signal Processing (Third Edition)*, pp. 13–58. Academic Press, third edition ed., 2019. doi: 10.1016/B978-0-12-815071-9.00002-6

[163] Y. Tanahashi, C.-H. Hsueh, and K.-L. Ma. An efficient framework for generating storyline visualizations from streaming data. *IEEE Trans. on Visualization and Computer Graphics*, 21(6):730–742, 2015.

[164] L. N. Trefethen and D. Bau. *Numerical Linear Algebra*. SIAM, Philadelphia, 1997.

[165] C. Truong, L. Oudre, and N. Vayatis. Selective review of offline change point detection methods. *Signal Processing*, 167:107299, Feb 2020. doi: 10.1016/j.sigpro.2019.107299

[166] C. Truong, L. Oudre, and N. Vayatis. Selective review of offline change point detection methods. *Signal Processing*, 167:107299, 2020. doi: 10.1016/j.sigpro.2019.107299

[167] J. H. Tu, C. W. Rowley, D. M. Luchtenburg, S. L. Brunton, and J. N. Kutz. On dynamic mode decomposition: Theory and applications. *Journal of Computational Dynamics*, 1(2):391–421, 2014. doi: 10.3934/jcd.2014.1.391

[168] J. W. Tukey. Mathematics and the picturing of data. In *Proc. ICM*, vol. 2, pp. 523–531, 1975.

[169] C. Turkay, E. Kaya, S. Balcisoy, and H. Hauser. Designing progressive and interactive analytics processes for high-dimensional data analysis. *IEEE Trans. on Visualization and Computer Graphics*, 23(1):131–140, 2017.

[170] C. Turkay, N. Pezzotti, C. Binnig, H. Strobelt, B. Hammer, D. A. Keim, J.-D. Fekete, T. Palpanas, Y. Wang, and F. Rusu. Progressive data science: Potential and challenges. *arXiv:1812.08032*, 2018.

[171] L. Van Der Maaten. Accelerating t-sne using tree-based algorithms. *J. Mach. Learn. Res.*, 15(1):3221–3245, Jan. 2014.

[172] Y. Vardi and C.-H. Zhang. The multivariate L1-median and associated data depth. *PNAS*, 97(4):1423–1426, 2000.

[173] R. Viviani, G. Grön, and M. Spitzer. Functional principal component analysis of fmri data. *Human Brain Mapping*, 24(2):109–129, 2005.

[174] J.-L. Wang, J.-M. Chiou, and H.-G. Müller. Functional data analysis. *Annual Review of Statistics and Its Application*, 3(1):257–295, 2016.

[175] Q. Wang, S. Zheng, A. Farahat, S. Serita, and C. Gupta. Remaining useful life estimation using functional data analysis. In *Proc. ICPHM*, pp. 1–8, 2019.

[176] X. Wang and A. McCallum. Topics over time: A non-markov continuous-time model of topical trends. In *Proceedings of the 12th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '06, p. 424–433. Association for Computing Machinery, New York, NY, USA, 2006. doi: 10.1145/1150402.1150450

[177] Y. Watanabe, H. Otsuka, M. Sonoda, S. Kikuchi, and Y. Matsumoto. Online failure prediction in cloud datacenters by real-time message pattern learning. In *4th IEEE International Conference on Cloud Computing Technology and Science Proceedings*, pp. 504–511, 2012. doi: 10.1109/CloudCom.2012.6427566

[178] G. M. Weiss, K. Yoneda, and T. Hayajneh. Smartphone and smartwatch-based biometrics using activities of daily living. *IEEE Access*, 7:133190–133202, 2019.

[179] P. Welch. The use of fast fourier transform for the estimation of power spectra: a method based on time averaging over short, modified periodograms. *IEEE Transactions on audio and electroacoustics*, 15(2):70–73, 1967.

[180] K. Xu, Y. Wang, L. Yang, Y. Wang, B. Qiao, S. Qin, Y. Xu, H. Zhang, and H. Qu. CloudDet: Interactive visual analysis of anomalous performances in cloud computing systems. *IEEE Trans. Vis. Comput. Graph*, 26(01):1107–1117, 2020.

[181] K. Xu, M. Xia, X. Mu, Y. Wang, and N. Cao. EnsembleLens: Ensemble-based visual exploration of anomaly detection algorithms with multidimensional data. *IEEE Trans. Vis. Comput. Graph*, 25(1):109–119, 2019.

[182] R. B. Yadav, P. S. Kumar, and S. V. Dhavale. A survey on log anomaly detection using deep learning. *2020 8th International Conference on Reliability, Infocom Technologies and Optimization (Trends and Future Directions) (ICRITO)*, pp. 1215–1220, 2020.

[183] Y. Yang and K. Chen. Temporal data clustering via weighted clustering ensemble with different representations. *IEEE Transactions on Knowledge and Data Engineering*, 23(2):307–320, 2011. doi: 10.1109/TKDE.2010.112

[184] Y. Yu, X. Si, C. Hu, and J. Zhang. A review of recurrent neural networks: Lstm cells and network architectures. *Neural Comput.*, 31(7):1235–1270, July 2019. doi: 10.1162/neco_a_01199

[185] M. Zaharia, R. S. Xin, P. Wendell, T. Das, M. Armbrust, A. Dave, X. Meng, J. Rosen, S. Venkataraman, M. J. Franklin, A. Ghodsi, J. Gonzalez, S. Shenker, and I. Stoica. Apache spark: A unified engine for big data processing. *Commun. ACM*, 59(11):56–65, oct 2016. doi: 10.1145/2934664

[186] T. S. Zaman, X. Han, and T. Yu. Scminer: Localizing system-level concurrency faults from large system call traces. In *2019 34th IEEE/ACM International Conference on Automated Software Engineering (ASE)*, pp. 515–526, 2019. doi: 10.1109/ASE.2019.00055

[187] L. Zeng, Y. Xiao, H. Chen, B. Sun, and W. Han. Computer operating system logging and security issues: A survey. *Sec. and Commun. Netw.*, 9(17):4804–4821, nov 2016. doi: 10.1002/sec.1677

[188] S. Zhang, Y. Liu, W. Meng, Z. Luo, J. Bu, S. Yang, P. Liang, D. Pei, J. Xu, Y. Zhang, Y. Chen, H. Dong, X. Qu, and L. Song. Prefix: Switch failure prediction in datacenter networks. *Proc. ACM Meas. Anal. Comput. Syst.*, 2(1), apr 2018. doi: 10.1145/3179405

[189] X. Zhang, S. Chandrasegaran, and K.-L. Ma. Conceptscope: Organizing and visualizing knowledge in documents based on domain ontology. In *Proceedings of the 2021 CHI Conference on Human Factors in Computing Systems*, CHI '21. Association for Computing Machinery, New York, NY, USA, 2021. doi: 10.1145/3411764.3445396

[190] X. Zhang, Y. Xu, Q. Lin, B. Qiao, H. Zhang, Y. Dang, C. Xie, X. Yang, Q. Cheng, Z. Li, J. Chen, X. He, R. Yao, J.-G. Lou, M. Chintalapati, F. Shen, and D. Zhang. Robust log-based anomaly detection on unstable log data. In *Proceedings of the 2019 27th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, ESEC/FSE 2019, p. 807–817. Association for Computing Machinery, New York, NY, USA, 2019. doi: 10.1145/3338906.3338931

[191] Z. Zheng, Z. Lan, R. Gupta, S. Coghlan, and P. Beckman. A practical failure prediction with location and lead time for blue gene/p. In *2010 International Conference on Dependable Systems and Networks Workshops (DSN-W)*, pp. 15–22, 2010. doi: 10.1109/DSNW.2010.5542627

[192] Z. Zheng, Z. Lan, B.-H. Park, and A. Geist. System log pre-processing to improve failure prediction. *2009 IEEE/IFIP International Conference on Dependable Systems & Networks*, pp. 572–577, 2009.

[193] X. Zhou, X. Peng, T. Xie, J. Sun, C. Ji, D. Liu, Q. Xiang, and C. He. Latent error prediction and fault localization for microservice applications by learning from system trace logs. In *Proceedings of the 2019 27th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, ESEC/FSE 2019, p. 683–694. Association for Computing Machinery, New York, NY, USA, 2019. doi: 10.1145/3338906.3338961

[194] Y. Zuo, H. Cui, X. He, et al. On the Stahel-Donoho estimator and depth-weighted means of multivariate data. *The Annals of Statistics*, 32(1):167–188, 2004.