The Terminator: An AI-Based Framework to Handle Dependability Threats in Large-Scale Distributed Systems

by

Khalid Ayed Budayai Alharthi

Thesis

Submitted to the University of Warwick

for the Degree of

Doctor of Philosophy in Computer Science

Department of Computer Science

May 2023

Contents

List of Tables

List of Figures

Acknowledgments

In the name of ALLAH, the most gracious, the most merciful (Bism ALLAH Alrahman Alrahim)

I am grateful to many people who have assisted me in one way or another during my PhD studies at the University of Warwick. I am honoured to acknowledge them in this thesis.

First and foremost, I would like to glorify the Almighty ALLAH, the Most Gracious and the Most Merciful, for the blessings HE has given to me during my PhD journey and the completion of this thesis. May ALLAH's blessings reach HIS final Prophet Mohammad (peace be upon him), his family, and his companions.

Second, I would like to devote this PhD thesis to my parents (Ayedh and Amarah) for their utmost love, support, sacrifices, and warm prayers. "Ya ALLAH„ have mercy upon them as they raised me when I was a child." (Quran 17:24). Ya ALLAH, grant my parents the highest place in Jannah. Also, I am forever grateful to my family (brothers (Abdullah, Mohammed, Budayai, Nemshan, Faisal, Nasser, and Ahmed), and sisters (Khazma, Hajda, Ghyfa, Waeila, Saliha, Badriya, Fatima, Khayria, and Shaykha), nephews, nieces, and all relatives and friends) back home in Saudi Arabia who motivated me throughout this journey and for their love and prayers.

I am profoundly grateful to my supervisor, Dr. Arshad Jhumka, for his tenacious support and inspiration, which lead me to strive for excellence throughout my PhD study. His invaluable guidance, suggestions, and motivation were and will continue to be pivotal in shaping my current and future academic and professional growth. His proficiency in providing incalculable feedback and helpful advice has enhanced the quality of my work and fostered my capability as a researcher. I am truly honoured to have had the opportunity to work under his supervision and I am very thankful for his lasting impact on my career. Thank you for your support during the loss of my father and brother.

I am also immensely grateful to Dr. Sheng Di for mentoring me during my research visit at Argonne National Laboratory, the University of Chicago. I sincerely appreciate his assistance in providing the Mira data logs for my research work and his invaluable contribution and insightful feedback on my research papers. Moreover, I would like to thank Prof. Franck Cappello for providing valuable comments on my research papers.

I would like to warmly thank: Dr. Anwesha Das (Stanford University) for answering my questions about the Cray supercomputer system, Dr. Edward Chuah (University of Aberdeen) and Dr. Nentawe Gurumdimma for answering my questions about Ranger and Lonestar4 clusters.

Furthermore, I would like to thank: (i) Argonne National Laboratory- the University of Chicago, for providing the IBM Blue Gene/Q logs, (ii) Texas Advanced Computing Center at UT-Austin for providing the Ranger and Lonestar4 data logs, and (iii) Sandia National Laboratories (SNL) and Los Alamos National Laboratory (LANL) for providing the Cray system logs.

I would also like to thank Prof. Simon McIntosh-Smith(Bristol University) for hosting me during my one-year research visit at Alan Turing Institute (ATI) and the Jean Golding Institute (JGI). I would also like to thank the teams from the student services and AI & Data Science at ATI for the wonderful administrative support they have given to me during my research visit.

I would like to thank Dr. Lin Gui (King's College London) for his invaluable feedback and fruitful research collaboration. Also, I am grateful to Prof. Yulan He (King's College London) and Dr. Gabriele Pergola and the team of the Natural Language Processing (NLP) Research Group at Warwick University for the opportunity to join the group and learn from them.

I would also like to thank Sharon Howard, Maria Ferreiro, and Kelly Chennery for all the administrative support they gave me at the Computer Science Department, the University of Warwick.

I owe much gratitude to my master's thesis advisors at Kent State University, USA, Prof. Austin Melton and Prof. Angela Guercio, for helping me take my first steps in the world of research.

I would like to thank my examiners, Prof. Gihan Mudalige and Prof. Karim Djemame, for their insightful feedback and recommendation which helped improve my final thesis. I would also like to thank my Viva advisor, Prof. Sara Kalvala, for her guidance and support throughout my Viva.

Many thanks to my friends (Ghurom Saad, Mohammed Buti, Ali Mohammed Alqahtani, Saad Sharaf, Mohammed GhuromALLAH,and AbduLLAH Ahmed).

Additionally, I would like to thank my lab-mates— Mansour Aldawood (For his exceptional guidance throughout my PhD journey), Mohammed Maray Alqahtani (For his and other friends' support during the loss of my father and brother and his valuable advices during my PhD), Ali Maray Alqahtani, and Majed Albarrak, for their tremendous support and friendship during my studies. Also, I want to thank all of my friends in the CS department.

Special thanks to my dear neighbours (Omar AlSaeed and Nasser Alhammad) for their brotherhood, advices, and support. Also, I am thankful to all of my friends I have met during my PhD study.

My ULTIMATE thanks are dedicated to my beloved wife (Afnan), and my daughters (Elaf and Mehaf), for their unwavering love, which played a crucial role during my PhD journey. They genuinely deserve endless appreciation. Thank you for being my most significant source of inspiration; I am forever thankful for having you by my side.

Declarations

The author confirms that this thesis has not been submitted for a degree at another university. In addition, the main contributions of this thesis have been previously published by the author in the following flagship, top-tier, and leading conferences:

[34] Khalid Ayedh Alharthi, Arshad Jhumka, Sheng Di, Franck Cappello, and Edward Chuah. **Sentiment analysis based error detection for large-scale systems**. In 2021 51st Annual IEEE/IFIP International Conference on Dependable Systems and Networks (**DSN'2021**), pages 237–249. IEEE, 2021. [Chapter 5] (GitHub [5]).

[35] Khalid Ayedh Alharthi, Arshad Jhumka, Sheng Di, and Franck Cappello. **Clairvoyant: a log-based transformer-decoder for failure prediction in large-scale systems**. In Proceedings of the 36th ACM International Conference on Supercomputing (**ICS'2022**), pages 1–14, 2022. [Chapter 6] (GitHub [4]).

[36] Khalid Ayedh Alharthi, Arshad Jhumka, Sheng Di, Lin Gui, Franck Cappello, and Simon McIntosh-Smith. **Time machine: Generative real-time model for failure (and lead time) prediction in HPC systems**. In 2023 53st Annual IEEE/IFIP International Conference on Dependable Systems and Networks (**DSN'2023**). IEEE, 2023. [Chapter 7] (GitHub [23]).

Abstract

With the advent of resource-hungry applications such as scientific simulations and artificial intelligence (AI), the need for high-performance computing (HPC) infrastructure is becoming more pressing. HPC systems are typically characterised by the scale of the resources they possess, containing a large number of sophisticated HW components that are tightly integrated. This scale and design complexity inherently contribute to sources of uncertainties, i.e., there are dependability threats that perturb the system during application execution. During system execution, these HPC systems generate a massive amount of log messages that capture the health status of the various components. Several previous works have leveraged those systems' logs for dependability purposes, such as failure prediction, with varying results. In this work, three novel AI-based techniques are proposed to address two major dependability problems, those of (i) error detection and (ii) failure prediction.

The proposed error detection technique leverages the sentiments embedded in log messages in a novel way, making the approach HPC system-independent, i.e., the technique can be used to detect errors in any HPC system. On the other hand, two novel *self-supervised* transformer neural networks are developed for failure prediction, thereby obviating the need for labels, which are notoriously difficult to obtain in HPC systems. The first transformer technique, called *Clairvoyant*, accurately predicts the location of the failure, while the second technique, called *Time Machine*, extends Clairvoyant by also accurately predicting the lead time to failure (LTTF). Time Machine addresses the typical regression problem of LTTF as a novel multi-class classification problem, using a novel oversampling method for online time-based task training. Results from six real-world HPC clusters' datasets show that our approaches significantly outperform the state-of-the-art methods on various metrics.

Acronyms

AI Artificial Intelligence.

ANN Artificial Neural Network.

BERT Bidirectional Encoder Representations from Transformers.

Bi-LSTM Bidirectional Long Short-term Memory.

CFG Control Flow Graph.

CNN Convolutional Neural Network.

DL deep learning.

FNN Feedforward Neural Network.

GAN Generative Adversarial Networks.

GPT-2 Generative Pre-trained Transformer-2.

GRU Gated Recurrent Unit.

HDFS Hadoop Distributed File System.

HMM Hidden Markov Model.

HPC High-Performance Computing.

KNN K-Nearest Neighbor.

LR Logistic Regression.

LSTM Long Short-Term Memory.

MCC Matthew’s Correlation Coefficient.

ML machine learning.

Multinomial NB Multinomial Naive Bayes.

NLG Natural Language Generation.

NLP Natural Language Processing.

PCA Principal Component Analysis.

RAS Reliability, Availability and Serviceability.

RF Random Forest.

RNN Recurrent Neural Networks.

SGD Stochastic Gradient Descent.

SMART Self-Monitoring Analysis and Reporting Technology.

SVM Support Vector Machine.

TACC Texas Advanced Computing Center.

TCN Temporal Convolutional Network.

TF-IDF Term Frequency - Inverse Document Frequency.

XGBoost Extreme Gradient Boosting.

Chapter 1

Introduction

Growing demand has driven expansion in the large-scale distributed systems market since the first supercomputer, Colossus, was introduced in the 1940s. Since then, the performance of clusters/supercomputers has been improved for decades to fulfil the high-end of High-Performance Computing (HPC) application requirements. Most of today's supercomputers are capable of accomplishing computations at the petascale (i.e., computing at least one quadrillion 10^{15} per second) and the shift towards supercomputers is accelerated as exascale computing proliferates, especially due to the fact that the exascale computation's barrier (i.e., computing at least one quintillion 10^{18} calculations per second) has been broken recently [101]. This is achieved by unveiling the world's first exascale cluster, Frontier, in 2021 [3, 233] which opened for operations in 2022 [3], and soon-to-be-released supercomputers, El Capitan [22] and Aurora [1]. This evolution requires re-examining various fundamentals where exascale services are expected to become a part of the application pipeline across multiple domains. Exascale computing can unlock possibilities that are not feasible by petascale clusters. It is predicted that exascale supercomputers will be able to construct a realistic model of the human brain and other complex neuromorphic tasks. Large-scale high-precision HPC applications in different sectors such as healthcare, climate, manufacturing, energy, unseen science materials discovery and much more can be put into practice on exascale supercomputers. Moreover, large-scale distributed systems, in general, are becoming increasingly commonplace as Artificial Intelligence (AI) recreates a more prominent role in most academia, industry, and business computing tasks.

For instance, Amazon, IBM, and Nvidia are just a few vendors building their own supercomputers to handle complex tasks that require massive amounts of high resources computing.

Today's industry, engineering, and scientific applications require to be executed on HPC environments such as supercomputers or data centres, which are made of many complicated hardware/software components. One use-case is the training of AI models (e.g., computer version, Natural Language Processing (NLP)). Gigantic corporations such as Microsoft and Google have invested billions of dollars to build cutting-edge supercomputers for training AI models. Supercomputers are also used for weather forecasting and climate-change models, which manipulate giant weather data and predict hazard situations such as hurricanes and tornadoes. Healthcare research, such as developing and tracking new disease drugs and understanding genome sequences, also requires supercomputers. Another use is astronomy simulations, such as cosmological simulations of the galaxy. Cryptanalysis, which involves understanding cryptosystems and analysing all aspects of cryptography tasks, also benefits from using supercomputers. Other applications that necessitate supercomputers include fusion reactor simulation, chemical compounds modelling, molecular dynamics simulation, and fluid dynamics simulation [250] just to name a few. All these application executions substantially depend on the supercomputers, because they all project resource-intensive computations and massive data to deal with.

The demand for HPC power is becoming a significant trend and ubiquitous in manipulating massive data volumes, also leading to ever-increasing requirements on dependability and resilience. Also, exascale systems' applications are expected to continue with evolving rates of performance improvements through parallelism, leading to programming challenges for these systems [249]; hence, bringing out an additional layer of reliability challenges. In fact, over the coming decades, exascale computing will significantly affect our daily lives, specifically with the spread of AI technologies, data centres, and increased reliance on cloud computing across all sectors, from manufacturing, research purposes, and social networks to predictive models and posting market growth. For example, the global cluster market is estimated to reach \$50 billion by 2026, according to Hyperion Research in 2022 [6]. The scale and complexity of these sophisticated systems have grown significantly in response to the ever-increasing demands, so that they are facing unprecedented reliability and resilience challenges in the regard of inevitable recurring failures on their components (e.g., compute nodes).

1.1 Motivation

Faults are commonplace in computer systems, including hardware faults, software defects, design flaws, implementation/operation errors, and other factors such as environment faults, limited memory, or network interruptions which may lead to system failure ultimately. The large-scale HPC systems tend to encounter serious resilience problems because of unexpected costly failures. Particularly, the growing complexity of their components has added a new layer of difficulty in handling these failures. The HPC systems' failures result in a drastic computational overhead, thus severely impacting applications execution, leading to waste of resources, high costs of financial losses, service quality degradation, and significant amount of administrators' time to resolve these problems.

There are various approaches to handle failures and make large-scale systems more tolerant such as error detection and failure prediction. Error detection aims to identify the system errors to run a recovery mechanism before these errors propagate to cause failures. On the other hand, failure prediction seeks to predict potential failures before they occur, and this approach often requires predicting the lead time to the predicted failures. Predicting the failures and lead times allows the triggering of proactive management solutions (e.g., checkpointing, job migration), which can effectively prevent the failures from occurring or reduce their impacts. An efficient failure predictor may forecast the failure occurrence accurately, which can effectively resolve all the above issues in turn.

- An efficient failure prediction can significantly increase users' satisfaction with their job execution on supercomputers and cloud services. This is because increasing failure prediction accuracy can effectively improve proactive solutions (i.e., checkpointing). Note that today's exascale applications generally require a long time for checkpointing because of vast volumes of data involved in the checkpointing. For example, Hardware/Hybrid Accelerated Cosmology Code (HACC) [151] may generate 220 TB data per snapshot when simulating trillions of particles [186]. Since there were no effective predictors for the failures and their lead times, the state-of-the-art checkpointing solutions [49] are designed to set checkpoints periodically in case the failures may happen at any moment during the execution, even though the failures actually may happen very rarely in practice. Hence, with an accurate prediction of the failure occurrence moment, periodic checkpoints can be taken with a target at the right place and right time.

- An efficient failure predictor can effectively suppress error propagation throughout the system, which may enhance the system's dependability in turn. According to studies [100], any fatal event may bring out a long chain of succeeding other fatal events, which may involve many different components in large systems. For instance, there are 2.6 million fatal messages in the five years of the Mira [11] system logging period while there are only 1,255 source fatal events after the spatio-temporal filtering. That is, each fatal event may cause about 2000 succeeding fatal messages on average in the system. Without an accurate prediction of failure lead time, the fatal event may not be fixed in time, which may easily induce severe problems to other parts of the system.

- An efficient failure predictor can significantly reduce the system maintenance cost. With accurate failure prediction, the maintenance requirement can be mitigated considerably because the precautionary solutions can be taken in time to avoid unexpected components failures, substantially improving the system's reliability. Specifically, developing accurate failure predictors can minimize the breakdowns, outages, and financial losses in large systems that provide services of healthcare, emergency, education, and social media application (e.g., Whatsapp, Twitter). For instance, social media companies, such as Facebook, lost millions of dollars for just a few hours of outage [2].

1.2 Challenges and Opportunities

Main Challenges The HPC systems generate a large amount of log messages that capture the health state of their components. Log files contain detailed runtime information about the systems, and they are the first source for the system administrators to identify the system anomalies and failures. Nonetheless, it is difficult to detect the system errors and predict potential failures by simply observing their log messages. This can be attributed to three key factors. (i) The rapidly developing scale of large contemporary systems causes logs to blow up excessively and become a significant challenge to handle. (ii) Log messages may come from different system logs [98] corresponding to different systems, leading to distinct logging methods and styles. (iii) Most log messages are unlabelled, redundant and incomplete, and only a small proportion of the log events are pertinent to system failures [150].

Unfortunately, the effectiveness of existing anomaly detection and failure prediction approaches is still insufficient, which may significantly affect the dependability of the complex systems. Thus, a more efficient online failure predictor is necessitated, which is expected to flag impending failures and their lead-times with high prediction accuracy and speed and with less computational overhead. This can ensure the proactive failure solutions would be scheduled at the optimal time (not sooner or later), mitigating the impact of system failures, minimising downtime, and reducing costs.

Opportunities We discuss key opportunities which we can take advantage of to develop efficient error detection or failure prediction models. On the one hand, the occurrences of the log events usually emerge in sequence patterns which indicate the occurrence of the corresponding failures. In simpler terms, consecutive log events preceding and leading up to a specific failure serve as symptoms of that failure. On the other hand, the log messages are lines of textual data which contain multiple pieces of information (e.g., node ID, timestamp, job ID, message, etc.). Thus, contemporary Artificial Intelligence (AI) and Natural Language Processing (NLP) approaches have been revolutionised and built based on the breakthroughs in the machine and deep learning techniques, and they are well-suited for processing and analysing log messages and developing models to detect and predict the system's failures with significant improvements in efficiency, accuracy, and automation.

The modern AI techniques have proven their superiority in various NLP tasks such as text generation and sentiment analysis. Handling complex tasks and large volumes of data with higher accuracy and less human intervention (e.g., no need for manual labels, automation) are key motivations for utilising the modern NLP techniques to resolve the complexity of log data. This thesis focuses on utilising the log data generated by the components of HPC systems and NLP approaches developed based on new AI techniques, to propose novel real-time models that can predict failures and their lead-times in HPC systems (i.e., failure prediction) accurately. It also aims to deliver an approach that can effectively detect errors and identify faulty components (i.e., error detection).

1.3 Basic Concepts About Dependability

This section aims to introduce some essential terminologies and concepts related to the notion of dependability and fault tolerance to pave the way for our approach and the challenges involved. The computing system refers to an entity that consists of multiple components (e.g., hardware, software, humans, and other entities) that interact with each other to deliver services. The system components should ensure a high degree of dependability by providing the intended service as expected. Dependability is defined by Avizienis et al. [45] as "the trustworthiness of system such that reliance can justifiably be placed on the service it provides". As depicted in Figure 1.1, the concept of dependability can be divided into three parts: the dependability attributes, the dependability threats, and the means to achieve dependability [44].

1.3.1 The Attributes of Dependability

Dependability is an umbrella concept which encompasses different attributes such as reliability, availability, safety, integrity, confidentiality, and maintainability. In this thesis, we focus mainly on reliability and availability. Reliability refers to the system's ability to perform its correct service consistently and accurately without failures while availability refers to the system's readiness to

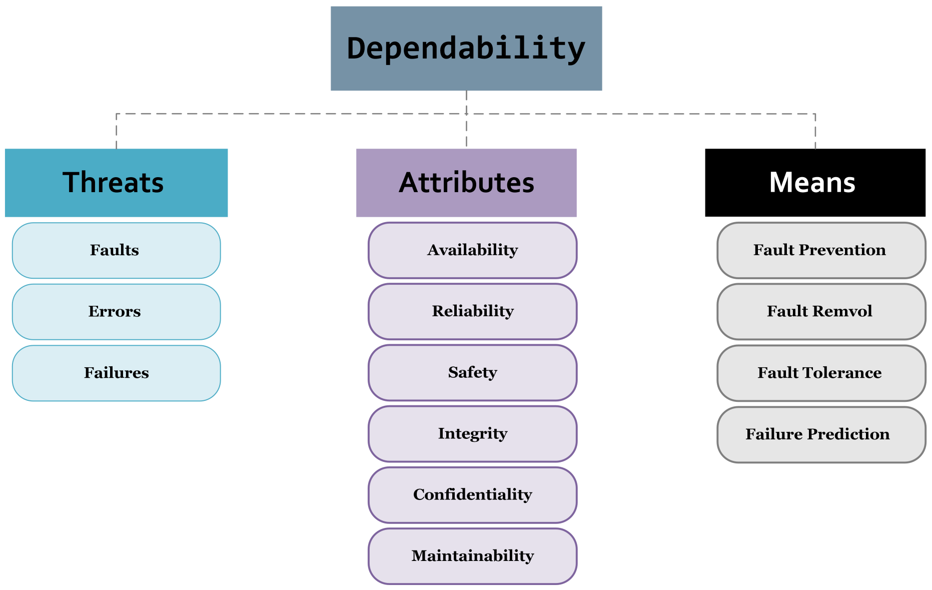


Figure 1.1: Dependability Tree [45]

perform its correct service when needed [45].

1.3.2 The Threats to Dependability: Faults, Errors, and Failures

The dependable system ensures delivery of the intended service without failure or interruption, whereas, the system components are vulnerable to errors that can arise because of system faults, leading to system failures. A system's dependability can be compromised by different incidents or events, which are collectively known as dependability threats. These threats are categorised into faults, errors, and failures, as shown in Figure 1.2. **Correct service** is achieved when the service performs the system function; on the other hand, the **service failure** (i.e., **failure**) is defined by Avizienis et al. [45] as "an event that occurs when the delivered service deviates from correct service". In other words, a failure occurs when a system shifts from achieving the correct service to the incorrect service by failing to perform the system function correctly. A service fails due to either non-compliance with the functional specification or the specification itself failing to define the system function sufficiently. A **service outage** is defined as the period during which incorrect service is being provided. The shift from this incorrect service to the correct service is known as **service restoration**. Service failure can be classified into different severity levels based on how much the faulty service is deviated from the correct one. A **fault** can be described as a hardware or software level anomalous behaviour that can lead to errors (i.e. illegal system states) [45]. This means that faults cause errors (i.e., an error can be considered a manifestation of a system's fault [150]). Programming mistakes or hardware defects are also examples of faults. The fault remains dormant or undetected unless it is evolved active to cause an **error**. In other words, an **error** appears in the log file when its corresponding faults are activated [175] and may lead to failures when the system deviates from the correct service state. That is, faults are the main reasons (i.e., root causes) for service failures. Moreover, if a sub-component fails, it may become a fault for the relatively another or large component since each large component generally consists of sub-components in a system. Failures can occur without occurrence of any previous errors, thus a failure is not always the result of an error. The failure lead time is defined as the time interval between the timestamp of the error and the timestamp of the failure [93].

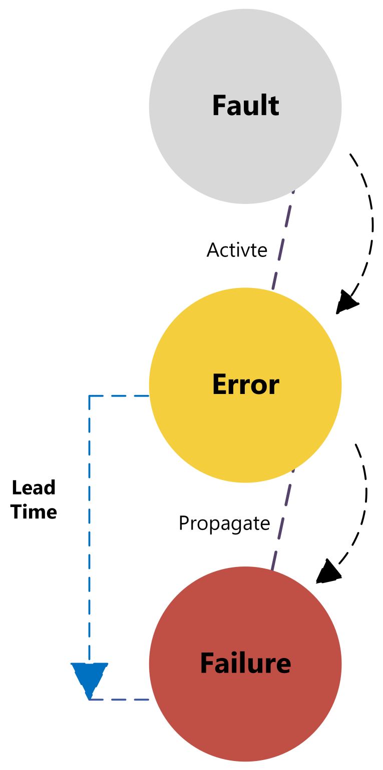


Figure 1.2: The relationship between faults, errors and failures

1.3.3 The Means of Dependability

Developing dependable systems is a complex and challenging task because it requires managing multiple aspects such as hardware and software design, users, security, maintenance, etc. Four classes of means have been proposed by Avizienis et al. [45] for managing faults - preventing, removing, tolerating, and forecasting, wihch are closely related to building dependable systems:

- **Fault Prevention:** refers to the means taken to prevent the introduction or occurrence of faults.
- **Fault Removal:** refers to locating faults, diagnosing and removing them.
- **Fault Tolerance:** refers to the means taken to allow the system to avoid service failures and continue functioning despite the occurrence of faults. Error detection and system recovery are examples of achieving this task.
- **Fault Prediction:** predicts the occurrences of current and future failures and their potential impacts.

1.3.4 Fault Tolerance and Failure Prediction

Fault tolerance is closely tied to a system's dependability. A system is known to be fault tolerant if it has the capability to continue to perform even if certain components fail. Distributed systems (e.g., supercomputer clusters) are often designed to be fault tolerant because they generally involves masses of components and compute resources connected by a network. More specifically, one important capability in a distributed system design is being able to bounce back from failures automatically without affecting the system's overall performance, such that the system can always work in a satisfactory, expected status. In general, the essential fault-tolerance attributes/characteristics in a distributed dependable system generally include availability, reliability, safety, and maintainability.

Key fault tolerance and fault prediction techniques are very diverse, including failure avoidance/mitigation, error detection, and failure prediction; Recovery management solutions are the cornerstones of fault tolerance in large-scale distributed systems. *Error detection or anomaly detection* aims at identifying anomalous patterns that deviate from a system's normal behaviour patterns. As mentioned previously, faults manifest themselves as errors, and systems record these errors as error log events. These errors could be the root cause or indication of upcoming failures; therefore, they serve as patterns or signatures of impending failures. Methods of error detection in HPC systems have focused on various aspects such as identification (i) of erroneous log entries [241], (ii) of failure-inducing erroneous execution sequence [148], [346], [147] and (iii) of detecting quantitative relationship among logs [195],[217]. Chapter (5) addresses the first problem and focuses on the automated classification of failure log entries, thereby obviating the need for the time-consuming manual labelling of such entries besides detecting the anomalous components (e.g., nodes). *Failure prediction* methods aim to predict the potential failure of software or hardware components to facilitate recovery actions to avoid or mitigate failures before they occur. To effectively handle failures, it is important to focus on two critical vital factors of failure prediction: (i) the spatial aspect to pinpoint the location of the potential failure, i.e., which component is expected to fail, and (ii) the temporal aspect to estimate the lead time to the failure occurrence moment, i.e., how much time is left before the component(e.g., node) fails. On the one hand, developing an accurate failure prediction approach to tell where the failure may occur is significant because it determines whether the proactive failure recovery method could be triggered correctly/accurately in space. On the other hand, the lead times to failures in large-scale systems also need to be predicted accurately, such that the recovery solutions can be scheduled at an appropriate timing (not sooner or later), minimising downtime and reducing costs. Specifically, if the lead-time predicted is over-long, unnecessary recovery actions may be executed, resulting in unnecessary costs. If the lead-time predicted turns out to be too short, the proactive operations cannot be done in time, so that the system component (e.g., node) may still fail, inevitably causing unexpected job failures, unplanned downtime and lost productivity.

1.4 Problem Statement

HPC systems, such as supercomputer clusters, execute resource-intensive applications to solve large-scale problems such as weather prediction and aerodynamics simulation. These systems are made up of sophisticated hardware (HW) and software (SW), which may often experience failures because of their scale and design complexity. The SW components, such as OS and parallel file systems, typically generate log messages that capture the health of various components in the system. Log data is a critical source of information for system administrators and fault-tolerance researchers to diagnose and improve the HPC system's dependability. As a result, system administrators can take advantage of the system log data to detect anomalies or predict system failures because it contains rich information about normal behaviour (i.e., informational messages) or abnormal behaviour (i.e., error messages) of various system components. Consequently, failure log analysis of HPC systems is attracting more and more researchers from academia and industry to improve the dependability of such systems. The AI and NLP have experienced a breakthrough revolution by the ever-advancing machine and deep learning techniques which motivates us to leverage them to develop fault-tolerant approaches with significant improvements in efficiency, accuracy, and automation.

The objective of this thesis is to address the following research questions:

I. Given a set of log messages generated by a large-scale system, can we develop an efficient AI-based approach to detect the errors of these systems? What level of accuracy can be achieved for this task?

II. Given a set of log messages generated by a large-scale system, can we develop an efficient AI-based approach to predict the failure of large-scale systems components (e.g., nodes) before they actually fail? And if so, can we develop an efficient AI-based approach to predict lead times for these failure events (i.e., how long is left before the predicted failure happens)? What level of accuracy can be achieved for both prediction tasks(i.e., failure and lead time prediction)?

1.5 The Approach

In this thesis, we introduce multiple novel approaches to error detection and failure prediction for large-scale systems based on the analysis of log data generated by the components of these systems.

I. Owing to assumptions that log messages often encapsulate the sentiment of system developers, which pertains to the perceived health of the system, we exploit these sentiments to generate a machine-learning based sentiment lexicon automatically, which can be used to detect errors for HPC systems and identify the faulty components. This model allows us to exploit source system logs labelled with severity levels and extract their sentiment features to automatically label log entries of other unlabelled (target) systems; thus, error detection is performed without prior knowledge of the target system logs. This integrated technique is designed to take advantage of the strengths of both approaches, sentiment lexicon and machine learning, to enhance error detection accuracy.

II. Owing to the properties that the health status of HPC system components can be deduced from consecutive log messages generated over time, we propose deep learning failure prediction approaches which are fully self-supervised without the need for labeling to predict large system failures and their lead times. In fact, failure events are often characterised by certain log sequence patterns, including the causes (i.e., error events) of these failures [77]. It is challenging to apply supervised learning because these logs are unlabeled, huge, and complicated. To overcome this challenge, our approach works by leveraging the preceding log events sequence for each component to forecast its entire health state through generating a sequence of forthcoming log events and then identifying if a failure event is part of the predicted sequence; if yes, we predict how long is left before the predicted failure happens (i.e., lead times prediction), so that appropriate proactive methods with low cost could be triggered in time. Unlike the supervised learning approach, our approach is self-supervised learning which falls into the category of unsupervised learning, where the training is without manual data labelling and expert knowledge. The unsupervised approach copes with any new types of log sequences and emerging failure patterns because of various cases, such as upgrades of the HPC system components (i.e., software, hardware, and services). The new jobs (e.g., applications) running on HPC systems can also induce new log patterns that have not been met before.

1.6 Thesis Contributions

This thesis aims to accomplish fault tolerant and dependable HPC systems. It introduces a significant contribution by exploring modern machine learning and deep learning approaches for error detection, failure prediction, and failure's lead-time prediction in large-scale systems (i.e., supercomputers clusters). We perform the evaluation using six log datasets generated by five different supercomputer clusters developed by different vendors. The results show that our machine and deep learning-based approaches significantly outperform other related state-of-the-art methods of error detection and failure prediction proposed for HPC systems. Our research contributions are summarized as follows:

- We propose a novel sentiment analysis-based utilising stochastic gradient descent logistic regression to automatically construct a generic and re-usable sentiment lexicon over the large-scale system logs. Based on that sentiment lexicon model, we propose two novel algorithms: the first for error detection based on the sentiment intensity score of log messages and the second to discover erroneous components (e.g., nodes) based on logs' sentiment polarity scores. Our method efficiently captures the developers' sentiment features from the log data of (source) systems that are labelled to automatically label the massive number of unlabeled logs (i.e., millions) of different (target) systems.
- We propose a novel failure prediction approach called **Clairvoyant** which is a self-supervised (no need for labels) transformer-decoder based model to predict node failures in HPC systems by first predicting the future sequence of events (future health state) and then identifying if a failure is part of the sequence.

- We improve our Clairvoyant model to enable the failures' lead-times by proposing a novel scalable log-based, self-supervised model, called **Time Machine**. It predicts (i) forthcoming log events, (ii) the failure location and (iii) the expected lead time to failure. Time Machine is designed by combining two stacks of transformer-decoders with the self-attention mechanism. The first stack predicts log events sequence to identify if a failure event is part of that sequence, while the second stack predicts lead time to failure.

- In the Time Machine model, we devise a novel synthetic minority oversampling technique for online time-based tasks to construct the training instances in real-time from failure sequences. This method has never been discovered before in AI or other domains and can be generalised to other domains for time-based tasks (e.g., business, healthcare, booking business).

- In the Time Machine model, we introduce a novel technique to reduce/convert the time prediction problem (a regression problem) into a self-annotated multi-class classification problem, by predicting the class for the failure lead time. This method can also be generalised to other domains for similar time-based tasks (e.g., business, healthcare, booking business).

- This thesis presents a taxonomy and a comprehensive survey of log analysis techniques from the perspective of four primary categories: log parsing, error detection, failure diagnosis, and failure prediction.

1.7 Outline of this Thesis

The remaining chapters of the thesis are structured as follows:

Chapter 2 provides the relevant background on the machine and deep learning which we utilised to develop our error detection models (to be presented in Chapter 5) and failure and lead time prediction (to be presented in chapters 6 and 7). This chapter also highlights the state-of-art models we compare our models with.

Chapter 3 provides a taxonomy and comprehensive literature review on dependability in HPC systems. To do so, this chapter outlines log-based fault tolerant approaches and categorises them into four primary domains: log parsing, error detection, failure diagnosis, and failure prediction.

Chapter 4 describes the generic system model of supercomputer cluster systems for which the approaches can be applied. This chapter presents an overview of five production clusters along with their logs used in our experiments. In addition, the assumed fault model in this thesis and HPC system node failure are illustrated, along with some essential terminologies related to the log messages and the techniques utilised are defined.

Chapter 5 illustrates our novel sentiment analysis-based approach for error detection in large-scale systems by automatically capturing the developers' sentiments in the log messages to build a sentiment lexicon. This chapter details our two algorithms to detect system errors and identify the components with erroneous behaviours based on sentiment intensity and polarity scores, respectively. The content of this chapter is mainly from my published work in the DSN'2021 conference [34].

Chapter 6 presents our developed *Clairvoyant*, a novel self-supervised approach to predict node failures in HPC systems based on a deep learning approach called transformer-decoder and the self-attention mechanism. The chapter details how Clairvoyant predicts node failures by (i) predicting a sequence of log events and then (ii) identifying if a failure is a part of that sequence. This chapter is fundamentally based on my paper published in the ICS'2022 conference [35].

Chapter 7 presents our novel real-time online approach namely *Time Machine*, which is fully self-supervised without the need for labeling to improve the Clairvoyant framework by enabling the lead-time prediction to failure. This chapter explains the framework of *Time Machine*, which consists of two transformer-decoder neural network stacks to predict possible failures and the failure lead-time accurately. Our strategy is converting the time prediction problem (a regression problem) to a self-annotated multi-class classification problem, by predicting the class for the failure lead time. Furthermore, this chapter introduces our novel synthetic minority oversampling technique for online time-based tasks to construct the training instances from failure sequences. This chapter is fundamentally based on my accepted paper in the DSN'2023 conference [36](To Appear).

Chapter 8 concludes and summarises our essential findings of approaches presented along with their limitations. It further discusses future research directions.

Chapter 2

Background

Preface

In this chapter, we provide a comprehensive overview of the background prerequisites for sentiment analysis, in light of our approach and its related baselines, which are presented in Chapter 5. In addition, this chapter will discuss the essential background concepts and notations for the transformer neural networks, with regard to our two approaches and their related baselines (i.e., Recurrent Neural Networks (RNN) variants) presented in Chapter 6 and Chapter 7.

2.1 Sentiment Analysis

The rapid expansion of applications, particularly social network apps and shopping platforms, has allowed people to express their opinions, provide reviews, communicate their feelings (positive, neutral, or negative), and share their experiences online toward products, services, or any entity. This has led to a tremendous and unprecedented growth in text data on websites, social networks, and marketing-related content that cannot be managed and analyzed using traditional methods. Thus, researchers, organizations, businesses, and governments have deployed high-efficiency NLP techniques such as Sentiment Analysis (SA) to manipulate the massive amount of data. Sentiment analysis is one of the NLP techniques, also known as opinion mining or attitude analysis, and it is the process of collecting, extracting, and classifying opinions, sentiments, and attitudes on various subjects, matters, products, and services[43, 314]. The sentiment polarity indicates whether an opinion is positive, neutral, or negative, while the sentiment intensity decides the degree of attitudes expressed, ranging from weak to strong. Sentiment text consists of three **O**s: (i) **Opinion Holder:** the individual or organization that contributes opinions to toward an entity; (ii) **Object:** the entity that the individual writes their opinion about; and (iii) **Opinion:** the text written by the user about that object [142, 306].

2.2 Sentiment Analysis Approaches

There are four main sentiment classification approaches: (i) lexicon-based, (ii) machine and deep learning, (iii) hybrid approaches, and (iv) other approaches (not our focus). An overview of the different methods used under each approach employed in sentiment analysis is depicted in Figure 2.1.

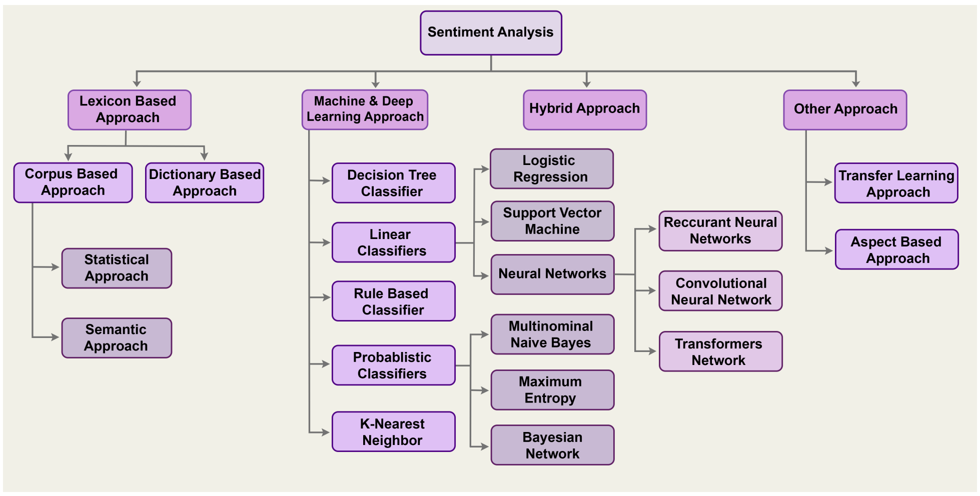


Figure 2.1: Sentiment Analysis Approaches[255]

2.2.1 Lexicon-based Approach

A sentiment dictionary (lexicon) is a collection of positive and negative features (i.e., tokens or words) where each feature in the lexicon is pre-assigned with a sentimental score based on its intensity of polarity, such as +1 for positive and −1 for negative. In the lexicon-based method, a given text (microblog, message, review, etc.) is classified based on the overall polarity score of the text. This is calculated through the summation of the sentimental score of each token within that text. Multiple researchers have utilized sentiment lexicons for text classification in their studies (e.g., [156, 169, 194, 269]). A sentiment lexicon can be generated using one of the following methods [200, 300]:

Manual-based Approach: The dictionary manual-based approach consists of a list of pre-defined sentiment features with their associated polarity scores assigned manually by domain experts [70, 190]. The lexicon elements (features) can be extended by exploring for antonyms and synonyms in available language resources. To ensure the lexicon generation process quality, the final stage includes a manual review by domain experts or integration with the other automated approaches to avoid mistakes resulting from incorrect sentiment score assignment [164, 236, 290].

Corpus-based Approach: The corpus-based approach also comprises a set of seed sentiment terms with pre-known scores. It then searches for syntactic and co-occurrence patterns in a huge corpus using language constraints, grammatical patterns, and connectives (e.g., AND, OR, BUT) to find other new sentiment tokens and add them to the lexicon. The process of discovering new sentiment features is conducted via a semantic approach or a statistical approach, as illustrated in the following:

i. Semantic Approach: In this approach, a computation process is conducted to calculate the similarity score between the pre-defined sentiment lexicon terms and the large corpus' tokens. In general, this technique explores large dictionaries (e.g., WordNet [244], SentiWordNet [283]) for similar words based on certain rules to calculate the semantic scores between terms and assigns the exact sentiment score to the semantically close terms to expand the sentiment lexicon [52], as presented in [51, 344].

ii. Statistical Approach: This approach adds the new sentiment terms based on a statistical method, such as co-occurrence frequency and computing mutual information, as proposed in [154, 302]. The basic concept of this method is that if sentiment tokens with similar sentiment co-occurred in the same context frequently, they tend to have the same sentiment polarity [314].

The main advantage of using the lexicon dictionary approach is that it does not require any training data; thus, it is considered an unsupervised technique by some experts. The primary drawback of this approach is that it is domain-specific; hence, features and their sentiment scores specific to one domain may not be utilised in other domains. In addition, it is a time-consuming approach [248].

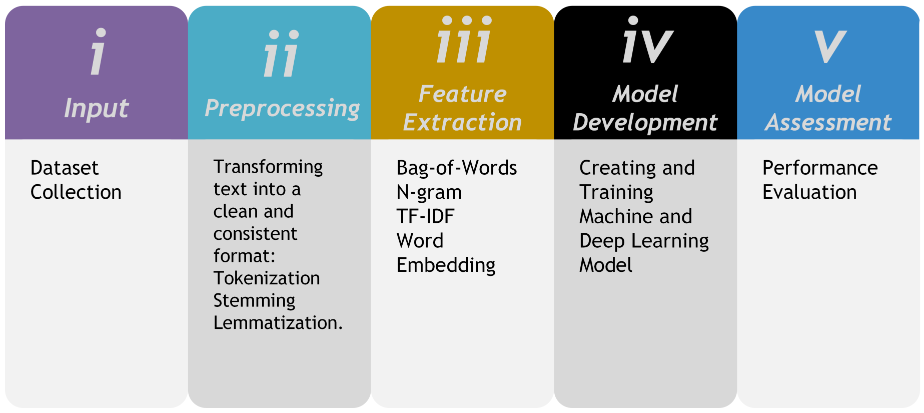

i
Input
Dataset Collection
ii
Preprocessing
Transforming text into a clean and consistent format: Tokenization Stemming Lemmatization.
iii
Feature Extraction
Bag-of-Words N-gram TF-IDF Word Embedding
iv
Model Development
Creating and Training Machine and Deep Learning Model
v
Model Assessment
Performance Evaluation

Figure 2.2: Sentiment Analysis Detection Phases Using Machine and Deep Learning

2.2.2 Machine and Deep Learning Approach

The machine learning (ML) and deep learning (DL) approaches utilise the ML and DL algorithms to classify sentiment polarity (e.g., negative, neutral, and positive) by training these algorithms on adequate textual data. This approach generally consists of five phases [255]: input data collection, data pre-processing, feature extraction, model training, and model assessment, as shown in Figure 2.2, and these phases are illustrated in Chapter 5.

The machine and deep learning techniques for sentiment classification can be categorized based on training types into two primary classes: supervised learning techniques and unsupervised learning techniques. Most machine & deep learning techniques applied for sentiment analysis are supervised and have demonstrated satisfactory accuracy; however, these methods require labelled data. On the other hand, unsupervised machine & deep learning models do not require labels during training, but they may cause inadequate accuracy. Both categories contain a variety of different models. We focus only on illustrating the Stochastic Gradient Descent - Logistic Regression (SGD-LR), which we employed to develop an ML-based method to automatically construct a sentiment lexicon for error detection in large-scale systems. Details are presented in Chapter 5. Moreover, we will explain the baselines – ML and DL models which we used to compare our sentiment lexicon's performance, including Random Forest (RT), Extreme Gradient Boosting (XGBoost), Multinomial Naive Bayes (Multinomial NB), K-Nearest Neighbor (KNN), and Long Short-Term Memory (LSTM).

Stochastic Gradient Descent - Logistic Regression (SGD-LR)

Stochastic Gradient Descent is an effective ML method proposed in [343] for solving linear prediction problems and improving the performance of support vector machines (SVM) and logistic regression (LR) models. The SGD classifier has demonstrated its performance in classifying large data with highly sparse features that are prevalent in textual data into either two classes (such as positive and negative) or one of the multiple classes [188] in the domain of NLP and sentiment analysis, such as [31, 133, 191, 193]. SGD is fundamentally an optimization technique that aims to determine the suitable parameters (coefficients) of a function that minimizes a cost (loss) function. SGD in the sentiment analysis domain is utilised as a linear classifier with a combination of one of the discriminative machine learning methods, such as SVM and LR. It is trained based on the textual data in the following way: predicting each text example's class output (label), comparing it to the actual class value, and calculating the difference between predicted and actual values. Based on a single random selection from training instances for each iteration, the gradient descent adjusts the feature weights until the predicted value is closer to the real value. The loss function determines how the SGD is optimised for the SVM or LR models. The SGD behaves similarly to a LR model when selecting the logit (sigmoid) loss function. On the other hand, when opting for the modified Huber loss function, the SGD works quite similar to the SVM classier [37]. Our model in chapter 5 is implemented based on the SGD classifier with the logit loss function, which works as an LR model to extract the developer sentiments and add them to a lexicon for error detection in HPC systems. Our model presented in Chapter 5 is designed to use binary classification, as this allows predicting the output class Y (negative or positive) based on a set of textual input features X (i.e., log messages). The model uses a sigmoid function that restricts the output between the range 0 and 1. The SGD-LR and its related functions and concepts are detailed in Chapter 5.

Random Forest (RT)

Random Forest [57] is a tree-based ML approach that has demonstrated its efficacy for regression and classification tasks in different domains, which include text classification. In sentiment analysis (e.g., [116, 192]), multiple decision trees are generated, and each tree is trained based on particular data features. Finally, the trees' output results are integrated to obtain a final prediction of the sentiment class label (negative or positive), which usually outperforms the decision tree approach. In sentiment analysis, the algorithm employs various properties (e.g., word frequency and grammatical structure) to detect the sentiment features. As shown in Figure 2.3 [199], each tree votes for a final decision on the sentiment class label for specific text data. The class label with the highest votes will serve as the class label for that text data [96].

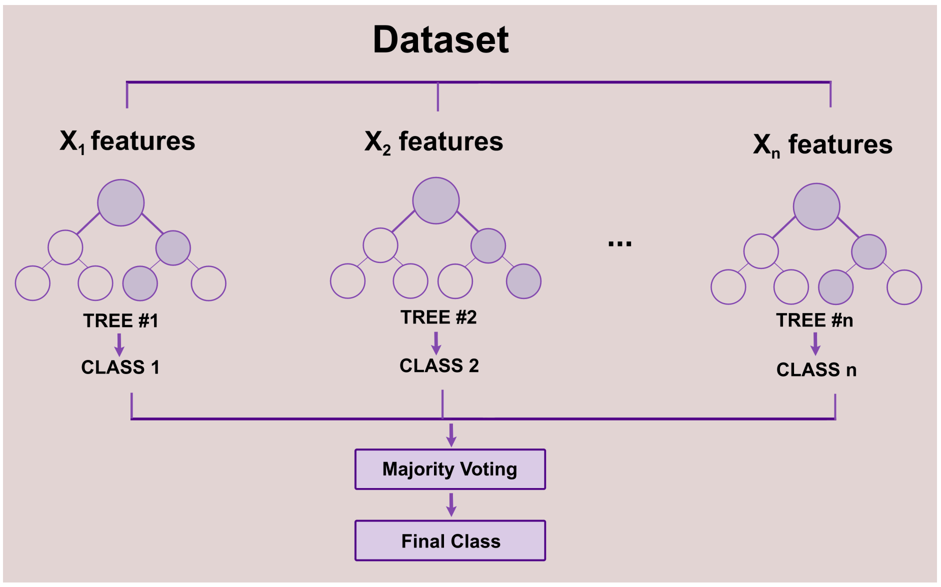


Figure 2.3: Random Forest (Example of ensemble of decision trees)

Extreme Gradient Boosting (XGBoost)

Extreme Gradient Boosting [67] is a sequential ensemble decision tree-based ML approach that employs a gradient boosting technique with parallel processing and early stopping abilities. XGBoost has been used for sentiment classification (e.g., [262, 280, 345]) and achieved high accuracy because of its ensemble learning feature facilitated by the gradient boosting process, where the predictive results of multiple classifiers are incorporated. During the boosting process, a sequence of tree models is created one after another with the goal of decreasing errors from the previous tree. Each tree model learns from the preceding tree models and rectifies residual errors. As a result, all the subsequent modes learn from these adjusted residuals. In boosting, the first classifier models usually tend to provide less accurate results. The boosting technique combines these results to produce a powerful model using the information contributed by each one to predict the final sentiment class label [29]. XGBoost predicts the class label $\widehat{y}_l^t$ after a sequence of training on the dataset X_i along with its corresponding labels Y_i, as defined in Eq 2.1:

$$\widehat{y}_l^t = \sum_{k=1}^{t} f_k\left(x_i\right) = \widehat{y}_l^{t-1} + f_t\left(x_i\right) \tag{2.1}$$

where $\widehat{y}_l^{t-1}$ is the prediction of the preceding model and $f_t\left(x_i\right)$ is the new model prediction. XGBoost aims to minimize the objective function, which contains loss function l and regularization parameters $\Omega\left(f_t\right)$, as depicted in Eq 2.2:

$$\mathcal{L}^t = \sum_{i=1}^{n} l\left(y_i, \widehat{y}_l^{t-1} + f_t\left(x_i\right)\right) + \Omega\left(f_t\right) \tag{2.2}$$

Multinomial Naive Bayes (Multinomial NB)

Multinomial Naive Bayes is a variant of the Naive Bayes technique, which is a probabilistic, supervised machine learning algorithm employed in various classification tasks. Based on Bayes' theorem and the use of a multinomial distribution for every text feature (i.e., word), Multinomial NB is applied for sentiment analysis detection (e.g., [26, 114, 293]) to classify a text into one of the sentiment classes. This model calculates a class' posterior probability based on the text's word distribution without considering the word's position within the text corpus (i.e., all features are independent of each other). The assignment of sentiment class is decided by the frequency of occurrence of features in the textual data, such as a review, sentence, or document, as shown below:

$$P(\text{ label } \mid \text{ features }) = \frac{P(\text{ label }) \cdot P(\text{ features } \mid \text{ label })}{P(\text{ features })} \tag{2.3}$$

The prior probability of a class label, denoted as $P(label)$, refers to the probability of a label being assigned to a random set of features (words or phrases), whereas $P(features/label)$ represents the prior probability of classifying a given set of features as a label. $P(features)$ denotes the prior probability of a set of features having occurred. Since the Multinomial NB assumes that all features are independent, we can write the above Eq 2.3 as follows [236] :

$$\begin{gathered} P(\text{ label} \mid \text{ features }) = \\ \frac{P(\text{ label }) \cdot P(f_1 \mid \text{label}) \cdot \ldots\ldots \cdot P(f_n \mid \text{ label })}{P(\text{ features })} \end{gathered} \tag{2.4}$$

K-Nearest Neighbor (KNN)

K-Nearest Neighbor is one of the traditional machine learning techniques that has been applied frequently for classification and regression tasks. It works based on the data instances' proximity to classify them into suitable classes derived from the insight that similar data are located close to one another. Accordingly, for the sentiment analysis (e.g., [83, 86, 166], the KNN technique utilises an analogy-based classification approach that involves comparing the new text data with the K-nearest neighbours of the most similar instances from the training dataset to measure similarity scores and assign that new text to the most comparable nearest class after a weighted average of the K neighbours' class labels. The similarity score is calculated using one of the distance metrics (e.g., Euclidean distance (Eq 2.5) and helps to form decision boundaries between the classes[109].

$$\text{distance } = \sqrt{\sum_{i=1}^{d} |Xe_i - Xt_i|^2} \tag{2.5}$$

where d represents the number of features, Xe_i refers to feature i in the new data, and Xt_i denotes feature i in the training data.

Long Short-Term Memory (LSTM)

Long Short-Term Memory (LSTM) [165] is a type of RNN deep learning technique employed to learn performing tasks such as regression, classification, and prediction in various domains. LSTM has demonstrated excellent performance in sequential data and NLP tasks, particularly for text generation and sentiment analysis. Regarding sentiment analysis, LSTM is widely exploited for sentiment analysis applications because it is similar to other deep learning methods that can automatically learn to extract sentiment features from textual data without manual feature engineering. Thus, researchers have employed its architecture to classify the sentiment or emotion expressed in textual data, such as negative, neutral, or positive (e.g., [208, 231, 251]. Generally, the architecture of the LSTM consists of an input layer, an output layer, and one or more hidden layers between them. Its architecture design is illustrated in detail in (Section 2.3.1).

2.2.3 Hybrid Approach

Many researchers have proposed hybrid sentiment analysis frameworks by combining lexicon-based and machine or deep learning-based techniques [326]. This can be accomplished by utilising a machine learning technique for generating the lexicon elements along with their scores or by integrating the machine learning model with a pre-defined lexicon dictionary. These hybrid techniques are designed to take advantage of the strengths of both approaches, pre-defined sentiment lexicon and machine deep learning, to enhance sentiment classification accuracy as demonstrated in many different research studies [94, 137, 288, 337].

2.3 Natural Language Generation (NLG)

Natural Language Generation (NLG) is a subset of NLP that focuses on generating a natural language text based on specific objectives. The generated texts can range in length and complexity from short responses (e.g., one word, sentence) to long responses (e.g., novel, long dialogue). NLG comprises multiple sub-tasks, such as text prediction, translation, chatbots, and summarisation. NLG aims to find an optimal text sequence $y_{<N+1} = (y_1, y_2, \ldots, y_N)$ that satisfies [271]:

$$y_{<N+1} = \underset{y<N+1}{\arg\max} \log P_\theta \left(y_{<N+1} \mid x\right) = \underset{y<N+1}{\arg\max} \sum_{t=1}^{N} \log P_\theta \left(y_n \mid y_{<n}, x\right) \quad (2.6)$$

where N denotes the number of words in the produced sequence, y is a set including all possible sequences, and variable $P_\theta \left(y_n \mid y_{<n}, x\right)$ denotes the conditional probability of the subsequent word y_n given the preceding sequence $y_{<n} = (y_1, y_2, \ldots, y_{n-1})$ and the source text sequence x, based on the model parameters θ.

The following subsections highlight the two primary approaches, Recurrent Neural Networks (RNN) and transformer neural networks, which have been widely employed for addressing text generation work. RNN includes the traditional RNN, LSTM, Bi-LSTM, and GRU models, which are our baselines in Chapter 6 and Chapter 7, whereas transformer is the base technique used for the approaches that we propose in Chapter 6 and Chapter 7.

2.3.1 Recurrent Neural Networks (RNN)

Recurrent Neural Networks are deep learning approaches that have manifested high performance in many NLP tasks, especially for modelling sequential data such as text prediction. Their effectiveness in sequence modelling tasks lies in their internal memories, which allow the storing of information from previous inputs alongside current ones [286]. In other words, the dependency among inputs where the output at each time step is computed by not just the current input, but also by the previous steps' output, which makes the RNN models suitable for manipulating complex NLP tasks such as text inference, language translation, etc.[170]. The traditional RNN consists of an input layer, a hidden layer (or more), and an output layer. The core element of the RNN is the hidden state, also referred to as "memory" as depicted in Figure 2.4 [292]. Given the input sequence x_t at time step t, we can calculate the hidden state,

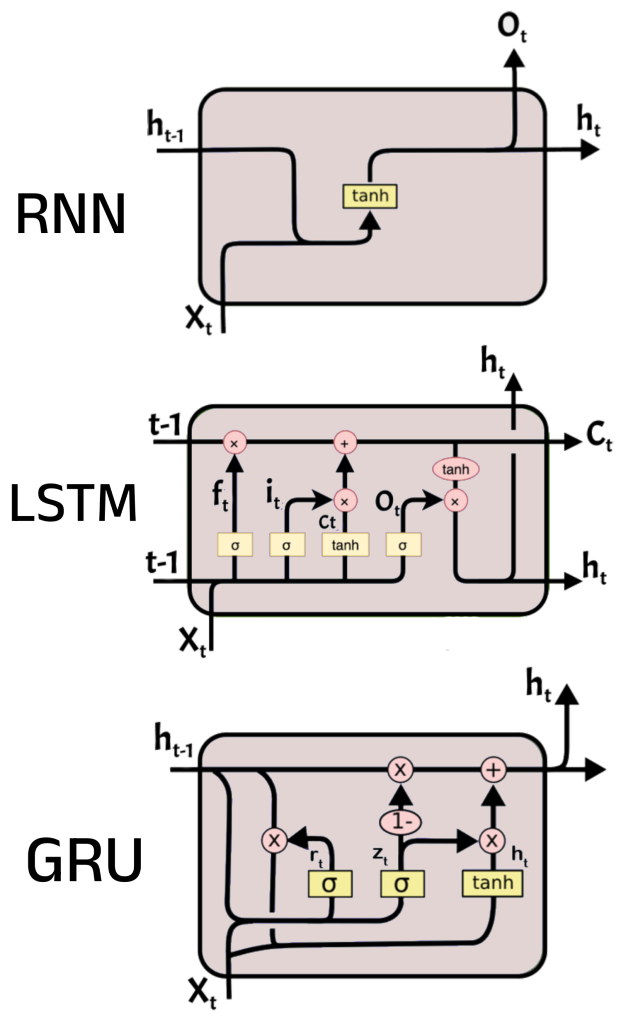


Figure 2.4: RNN Variants Cell Diagrams [292]

h_t, and the single hidden layer output, o_t, as follows:

$$\begin{aligned} h_t &= \sigma_h \left(U_h x_t + V_h h_{t-1} + b_h\right) \\ o_t &= \sigma_y \left(W_y h_t + b_h\right) \end{aligned} \tag{2.7}$$

where x_t denotes the input vector, h_t represents the hidden layer vector, o_t indicates the output vector, b is the bias vectors, V represents parameter matrix, U & W are parameter matrices, and both σ_h & σ_y are activation functions.

The RNN networks feed the input x_t at every iteration (time step) t to calculate the h_t, also taking into account the previous output h_{t-1}. Despite RNN's usefulness for addressing problems with sequential data, which requires it to share the weights among the inputs across the time steps, these models suffer from certain drawbacks, including long training times due to recurrent learning, and the vanishing gradient problems, which leads to limited accuracy due to losing earlier memory of the previous output. Therefore, it is challenging to use the vanilla RNN to solve tasks that require learning long-term dependencies because of the vanishing gradient problem and resultant loss of information over time. Accordingly, different variants (e.g., LSTM, Bi-LSTM, GRU) have been proposed to address some of the limitations of traditional RNN and enhance its performance.

Long Short-term Memory (LSTM)

Long Short-Term Memory [165] is an extension of the traditional RNN to circumvent the main limitation of its architecture. This is achieved by adding components such as 'memory cell states units $C_t \& \tilde{C_t}$', which include memory states for the previous state and the current memory to determine what to keep in memory and what to remove or ignore. This improvement allows the LSTM to preserve more information in memory for long sequences and learn longer-term dependencies. In addition, to reduce the vanishing gradient problem, three gates (input gate i_t, forget gate f_t, and output gate o_t) are added to control the flow of data to and from a memory cell in LSTM networks (See Figure 2.4). Below are the equations for computing the state of each gate and cell.

$$\begin{aligned}
f_t &= \sigma\left(W_f \cdot [h_{t-1}, x_t] + b_f\right) \\
i_t &= \sigma\left(W_i \cdot [h_{t-1}, x_t] + b_i\right) \\
o_t &= \sigma\left(W_o \cdot [h_{t-1}, x_t] + b_o\right) \\
\tilde{C_t} &= \tanh\left(W_c \cdot [h_{t-1}, x_t] + b_c\right) \\
C_t &= f_t \odot C_{t-1} + i_t \odot \tilde{C_t} \\
h_t &= o_t \odot \tanh\left(C_t\right)
\end{aligned} \tag{2.8}$$

where i_t, f_t, o_t denote the three gates (input, forget, and output), x_t is the input vector, h_t indicates hidden layer vector, b represents the bias vectors, W_f, W_i, W_c, W_o are weight matrices, $\{\sigma$ and $\tanh\}$ denotes two different activation functions, C_t and $\tilde{C_t}$ indicate the memory cell and the internal hidden state, respectively, and $\odot$ refers to the element-wise vector product operation.

These gates can selectively add or delete data in a cell's state, which helps the network to preserve relevant data and forget specific information (such as anything irrelevant) as needed. These improvements allow LSTM to replace the traditional RNN in a broad range of sequential data tasks, including text inference, speech recognition, and time series prediction, and multiple models have been presented based on its framework, such as [58, 222].

Bidirectional Long Short-term Memory (Bi-LSTM)

The second variant of RNN is Bidirectional Long Short-term Memory (Bi-LSTM) [282], which is designed to process sequence data in two directions (forward and backward) by duplicating the LSTM layer. In comparison, the previous LSTM framework can only process sequences in a forward direction. In other words, to capture information from both ends of the sequence, Bi-LSTM represents the input data twice (forward and backward), then concatenates the last output hidden states. Thus, Bi-LSTM requires twice as many parameters and computation steps as LSTM does, incurring higher computational costs than LSTM. Different models employed Bi-LSTM framework for text generation tasks, such as [28, 163].

Gated Recurrent Unit (GRU)

Gated Recurrent Unit [72] was developed by Cho et al. in order to reduce the RNN's vanishing gradient problem. Its architecture differs from that of LSTM regarding the number and type of gates. Specifically, as shown in Figure 2.4, GRU contains two gates— reset (r_t) and update (u_t); where the update gate determines how much previous information is passed along to the next state to reduce the vanishing gradients, and the reset gate which is designed by merging the LSTM's input and forget gates to decide how much past information needs to be disregarded to determine if the amount of the previous cell state is relevant. The following are the equations to compute the cell states and the output of each layer:

$$\begin{aligned}
u_t &= \sigma\left(W_u \cdot [h_{t-1}, x_t] + b_u\right) \\
r_t &= \sigma\left(W_r \cdot [h_{t-1}, x_t] + b_r\right) \\
\tilde{h}_t &= \tanh\left(W_h \cdot [r_t \odot h_{t-1}, x_t] + b_h\right) \\
h_t &= (1 - u_t) \odot h_{t-1} + u_t \odot \tilde{h}_t
\end{aligned} \tag{2.9}$$

where x_t is the input vector, h_t and $\tilde{h}_t$ indicate the hidden layer vectors, W_u, W_r, W_h represent parameter matrices, b denotes the bias vectors, and $\sigma\&\tanh$ indicate the used two activation functions.

GRU has fewer parameters, making it a less complex architecture and thus faster than LSTM. Its performance has been shown to be as accurate as LSTM. Many studies based on GRU have been proposed for text prediction (e.g., [40, 196]).

We refer the readers to see [80, 170] for more details about the RNN models and their technical framework regarding text generation.

2.3.2 Transformers

Although solutions in the form of RNN' variants have been proposed to address the serious vanishing gradient problem, none of these techniques fully resolve it, especially when dealing with long textual sequences. In addition, all RNN-based models still handle sequences sequentially in recursion mode token by token, thus hindering the parallel processing. Furthermore, if input sequences become excessively lengthy, the model is susceptible to either forgetting the information from distant positions within the sequence, or twisting it with the information of their subsequent positions. Consequently, a breakthrough, the transformer neural networks approach by Vaswani et al. [305] in AI and NLP, was introduced to tackle 's limitations through the *self-attention mechanism* and *parallelization.*

Transformers are similar to RNN in that they are developed to handle sequential data, however, unlike RNN, they, through the self-attention mechanism, do not necessarily need to process data in sequence one after another. Instead, transformers utilise self-attention to capture relationships and connections between consecutive sequence elements (i.e., words) while processing

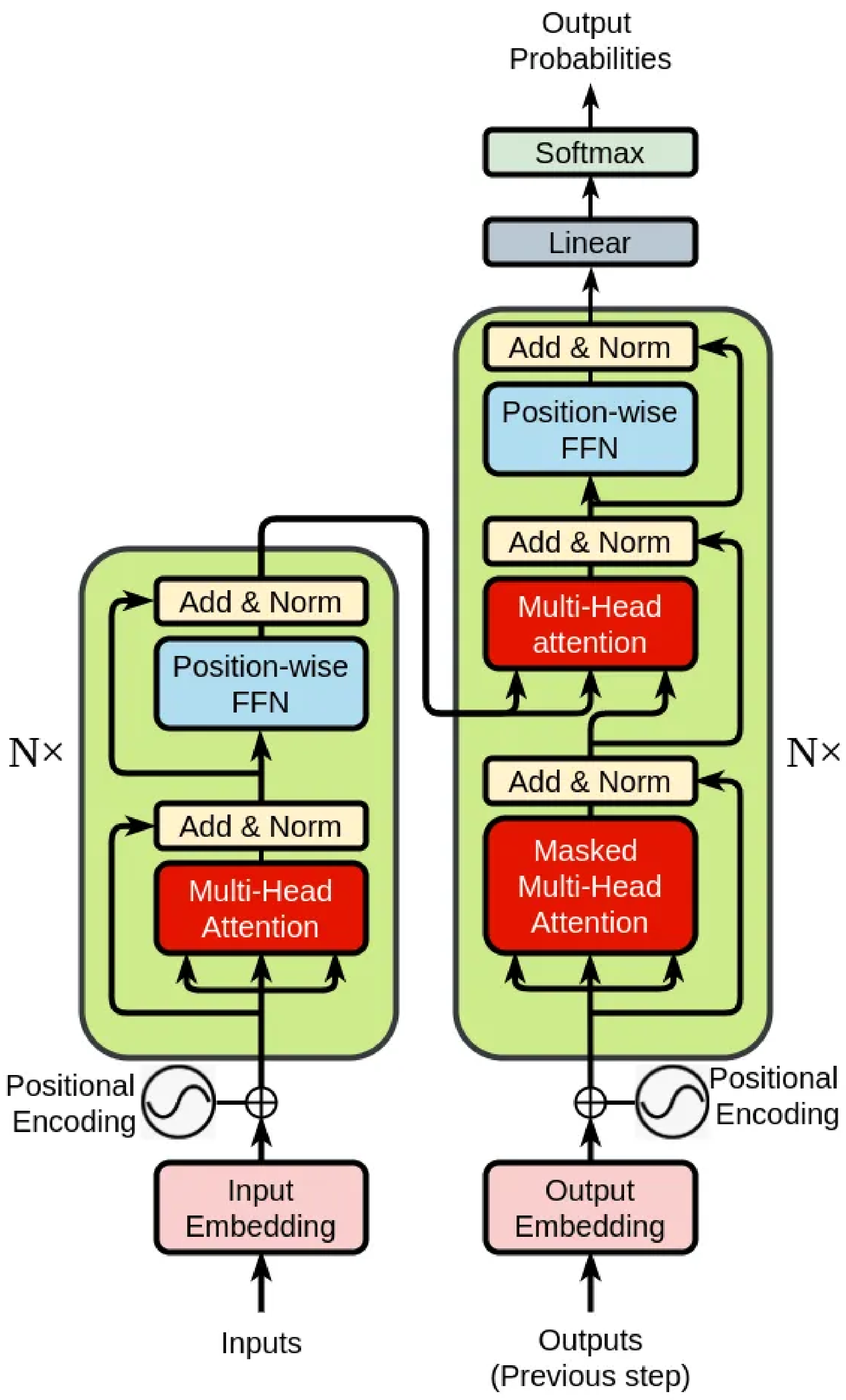


Figure 2.5: Transformer Neural Networks Architecture [19, 305]

at once in parallel. The new positional embedding layer facilitates parallel processing to replace the recursion mechanism, the main weakness of RNNs. This section highlights the vanilla architecture proposed by Vaswani et al. [305] that other transformer variants architectures are based on with different design changes. Figure 2.5 depicts the vanilla architecture of the transformer model, which comprises two stacks: six encoders and the same decoders. In general, each encoder/decoder is made up of three neural network components: multi-head self-attention, feed-forward network (FNN), and add & normalisation. The decoder stack contains another sub-layer that calculates multi-head attention over the encoder's output. Utilising self-attention and feed-forward neural networks, the encoder takes the input sequence and generates a sequence of hidden representations, each containing information about a particular data point in the input sequence. These hidden representations are employed by the decoder, which also computes hidden representations for each data point in the output sequence using self-attention and feed-forward neural networks. The following are the high-level explanations for the main components of each encoder/decoder stack and their purposes:

- **Multi-Head Attention:** The transformer uses multi-head attention, which enables the model to simultaneously take into account several pieces of the input or output sequence (i.e., tokens) while computing hidden representations to understand the relations among the sequence elements. It does so by emphasising the different elements in the sequence with different degrees of significance (i.e., weight).

- **Feed-Forward Neural Networks**: It receives the output of the hidden layers that capture the hidden correlations among the sequence's elements by applying the activation function to add non-linearity to the model. In this way, it learns more complex and abstract representations of the input data, and then passes forward what has been captured to the next layer.

- **Embedding layers:** The transformer framework represents the input sequence by combining two different types of encoding: input embedding and positional embedding. Input embedding represents each token in the sequence with fixed-size dense vectors, learnt throughout the training phase, to capture the token's meaning. Positional encoding, on the other hand, is utilised to compute information about the position of each token, because self-attention does not contain information about the tokens' order in the sequence. As a result, input embedding depicts the tokens' meaning, while positional encoding indicates their location, which are then combined to capture both features of the input sequence.

- **Residual connection and normalisation layers:** Each transformer contains residual connection and normalisation layers. Residual connections allow information to travel over the transformer layers without modification and prevent the vanishing gradient problem when its values become small, which can cause slow learning. The normalisation layers stabilise the distribution of hidden states across layers, thus making the mode training more effective.

For more details about the vanilla transformer, see [32, 305]; for details about other generative models, see [103]. In our thesis, we employ Generative Pre-trained Transformer-2 (GPT-2) [272], which is a variant of [305], to model our solutions. This is explained in detail in Chapter 6 and Chapter 7. GPT-2 is a decoder-based transformer developed especially for generating text tasks (e.g., predictions for the next word, text summary, etc.)[33].

Summary

This chapter provided insight into an important aspect of sentiment analysis, and its different approaches, which is the fundamental background to our proposed sentiment analysis-based technique and its baselines; these are presented in Chapter 5. Particularly, we analyzed the Logistic Regression (LR) model with the Stochastic Gradient Descent (SGD) optimiser, RF, XGBoost, Multinomial NB, KNN, and LSTM. Moreover, this chapter provided a prerequisite overview of the transformer neural networks utilised to develop our two frameworks (Clairvoyant and Time Machine), which are presented in Chapter 6 and Chapter 7. Furthermore, we outlined the Recurrent Neural Networks and its three famous variants, LSTM, Bi-LSTM, and GRU, which we used as state-of-the-art approaches for a comparison with our own proposed approaches, which is detailed in Chapter 6 and Chapter 7.

Chapter 3

Literature Review

Preface

Logs are data that record information about the health states of software and hardware components for HPC systems, and in the last decades, they have become increasingly important in the reliability assurance process. Numerous automated log-based analysis studies have been conducted to utilise logs effectively and efficiently in large-scale system reliability. This chapter presents a survey of crucial automated log analysis research efforts from the four primary categories perspectives, as illustrated in Figure 3.1: log (parsing) preprocessing, anomaly detection (i.e., error detection), failure diagnosis, and failure prediction, which serves as a significant related work study to support the failure prediction and error detection focused by this thesis.

The first domain is log parsing, as detailed in Section 3.1, which includes solutions that aim to transform the raw, unstructured message logs produced by large-scale systems into more structured data. The second domain is about anomaly detection and related profiling methods of machine learning (ML) and deep learning (DL), as detailed in section 3.2, which are designed to identify abnormal system and unexpected behaviour patterns. The methods used in the two key domains can be further divided into sub-categories for each. The third domain is developing failure diagnosis approaches to uncover the root causes (i.e., errors and faults) of failures in the components of large-scale systems, which is described in section 3.3. The last domain includes the use of failure prediction to generate proactive early alerts to avert failures in HPC systems, which is detailed in section 3.4.

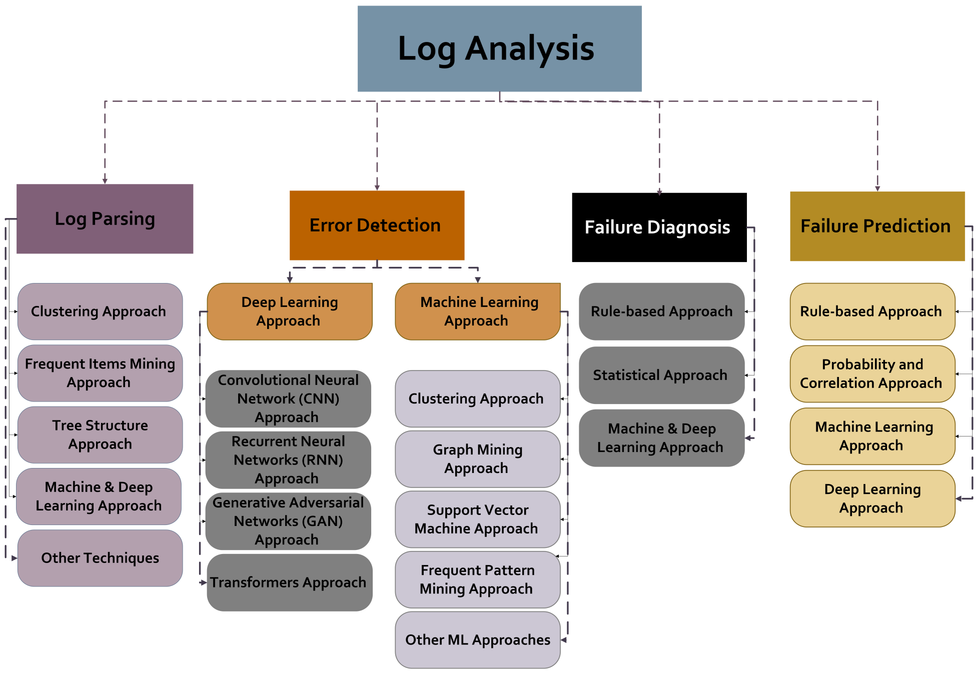

Log Analysis
Log Parsing
Error Detection
Failure Diagnosis
Failure Prediction
Clustering Approach
Frequent Items Mining Approach
Tree Structure Approach
Machine & Deep Learning Approach
Other Techniques
Deep Learning Approach
Machine Learning Approach
Convolutional Neural Network (CNN) Approach
Recurrent Neural Networks (RNN) Approach
Generative Adversarial Networks (GAN) Approach
Transformers Approach
Clustering Approach
Graph Mining Approach
Support Vector Machine Approach
Frequent Pattern Mining Approach
Other ML Approaches
Rule-based Approach
Statistical Approach
Machine & Deep Learning Approach
Rule-based Approach
Probability and Correlation Approach
Machine Learning Approach
Deep Learning Approach

Figure 3.1: A Taxonomy of Log Analysis Approaches

3.1 Log Parsing

The health of large-scale systems is usually determined upon a challenging analysis of a large amount of log data which is often unstructured, redundant and incomplete. Administrators use the log data to observe and monitor their systems' health because it contains runtime details. However, log data analysis is still challenging because of the growing extent and complexity of large-scale systems, which cause exponentially increasing log size, as well as its unstructured nature. Consequently, it is impractical to manually inspect the massive and unstructured log files generated by these sophisticated systems. For these reasons, log preprocessing (parsing) serves as the first crucial phase preceding the subsequent log analysis. It filters and processes high quantities of large-scale systems' raw and unstructured message logs into structured data. To this end, considerable studies (e.g., LogAider [99], MoLFI [242]) have been conducted on automated log parsing, in addition to the creation of industrial solutions tools (e.g., Logentries [9], Splunk [14]), resulting in the development of various log preprocessors (parsers) based on various methodologies.

Log files contain millions of log messages, each incorporating multiple pieces of information to report a specific state of the system component.

Each log message contains two parts: the header and the content. The header includes several attributes that can be easily extracted and manipulated through preprocessors. Event time (timestamp), severity level (e.g., INFO, WARN, FATAL), a component that triggers the log message (location), and the associated job are some examples of header attributes, which may vary from system to system. The message content describes a system event, generally written by the developers in the form of free text, so extracting valuable information from the message context is non-trivial. Message content involves constant strings and variable values. The constant strings are the English words or characters used by the system developers to describe a specific event, so they are reused frequently to represent the same event in the log file. The variable values are parameters that record dynamic runtime information, often combined with the constant strings. The objective of log parsing is to transform each log message into a particular event template. For instance, the log message (2013, Feb, 10, 02:02:03 1330599150 i147-409 kernel soft lockup CPU stuck for <3>BUG: 5359) is transformed to its associated log template (kernel soft lockup CPU stuck for *BUG:*), symbolising the key parameters (e.g., ['3', '5359']), as shown in Figure 3.2.

Log parsing is the first task followed by multiple log analysis steps, so it serves as the backbone of the entire analysis. Various data-driven parsers have been proposed based on distinctive parsing approaches, and some existing parsers are built by incorporating two or more parsing methods. These parsers differ in terms of parsing strategy, accuracy, efficiency, robustness and mode (i.e., online or offline). In this chapter, we classify the existing parsers into several categories based on their parsing approaches: clustering, frequent pattern mining, tree structure, machine/deep learning techniques and other approaches (e.g., heuristic techniques). More details are provided on each category method with different parser examples in the following sections.

3.1.1 Clustering Approach

Log parsing can be considered a log message clustering task because the log messages that share the same log templates are classified into the same groups (clusters). LenMa [287], LogMine [153], CLF [339], LogOHC [331] and

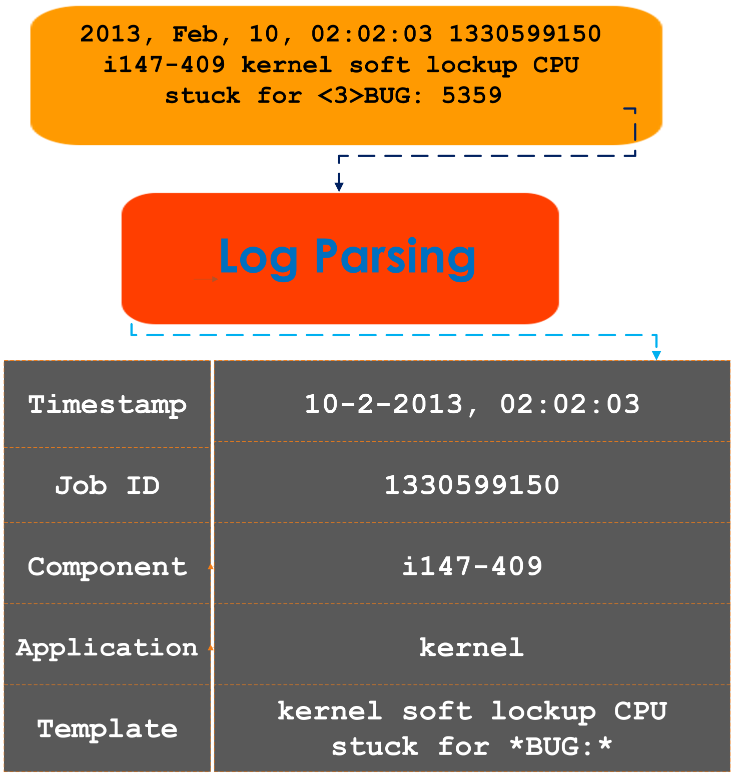

2013, Feb, 10, 02:02:03 1330599150 i147-409 kernel soft lockup CPU stuck for <3>BUG: 5359
Log Parsing
Timestamp
10-2-2013, 02:02:03
Job ID
1330599150
Component
i147-409
Application
kernel
Template
kernel soft lockup CPU stuck for *BUG:*

Figure 3.2: Raw Log to Structured Log

amulog [203] are instances of clustering algorithms used for parsing the log files. LenMa and LogOHC utilise the length of each log message token (i.e., word) and word distributed representation (word2vec), combined with a hierarchical clustering method, to deploy online template clustering parsers. They calculate the similarity of each newly arriving log message with existing log template clusters. If it is successfully matched, the log message will be attached to the existing cluster with the highest similarity score; otherwise, a new log template cluster will be generated. LogMine is an offline parser to generate log event templates by a hierarchical clustering method that groups log messages into clusters from bottom to top. Zhang et al. proposed CLF [339] which is clustering method based on extracting the log message templates from raw log based on the length and first word integrated with some heuristic rules. Amulog [203] is a general framework to parse log messages by dividing them into headers and segmented messages, employing a template-matching strategy to cluster them and storing the log templates in a database.

3.1.2 Frequent Items Mining Approach

Frequent pattern mining is a kind of log parsing because the log message templates can be viewed as a dataset of transactions, and each event template can be treated as a transaction that comprises a collection of items that occur frequently. LFA [252], LogCluster [303], Logram [84], and Sequence-RTG [157] are examples of this parsing strategy. These four log parsers follow similar steps: first, multiple passes over log messages are conducted; then, frequent itemsets (tokens) are created from each traversal; after that, log messages are collected into different groups; and finally, event templates from each group are extracted. LFA employs the word frequency distribution in each log event to parse log messages rather than all of the log data. LogCluster parses the log messages based on the frequent occurrence of tokens without considering their position in the log messages. Logram is based on frequent n-gram dictionaries, and its key insight comes from states that frequent n-grams are generally constants. Sequence-RTG (Sequence-Ready-To-Go), an extended version of the pattern-mining framework called SEQUENCE [12], was proposed to identify patterns in system log messages. LogAider [99] is a toolkit deployed for parsing and mining the Reliability, Availability and Serviceability (RAS) and job logs of IBM BlueGene HPC series supercomputers [98] based on statistical correlation methods across log message fields using an optimised K-means algorithm for spatial correlation.

3.1.3 Tree Structure Approach

A tree-form structure is utilised as a base for multiple log parsers and may sometimes be combined with other parsing strategies. Specifically, He et al. proposed Drain [160], an online log parser that preprocesses the log messages using some heuristic rules (regular expressions) and then utilises a fixed depth parse tree to speed up the process of grouping log messages. Those belonging to similar log templates are grouped into the same leaf node through these heuristic rules encoded within the internal nodes of the tree. Another example is FT-tree [341], which combines two different parsing strategies – tree structure and frequent pattern mining – to develop a frequent template tree (FT-tree)-based parser. This design is inspired by the FP-tree technique [155], a structure that is incrementally re-trainable to identify ‘correct’ templates of syslogs of data centres’ switches. The FT-tree design is motivated by the fact that a syslog message sub-type is usually the longest sequence of frequently occurring tokens, thus obtaining a template entails identifying the longest combination of frequent tokens from syslog messages. Researchers compared FT-tree performance to signature tree [270], statistical template extraction (STE) [197], and LogSimilarity [198] using real-world switch syslogs of 10 data centres over two years. The evaluation results showed that the FT-tree achieved the highest accuracy but it suffers from a high computational cost, as it is incrementally re-trainable. Vervaet et al. [307] proposed USTEP, a log parser based on an evolving tree descent structure that encodes parsing rules based on the parsed logs in an online mode. This paper presents USTEP-UP, a method of handling multiple instances of USTEP in a parallel manner. Also, Plaisted et al. [267] proposed the disagreement index parser, a decision-tree-based log parser. The core idea of this method is using some conditions to determine if tokens at which similar log events disagree are parameters. It can match similar events to the same log template class if the tokens of disagreement are ‘look like’ parameters.

3.1.4 Machine & Deep Learning Approach

Over the last decade, machine & deep learning-based log event parsing has been extensively studied, and numerous artificial intelligence method-based approaches have emerged. For instance, Meng et al. [240] developed LogParse by turning the parsing of the log template problem into a token classification problem and utilising Support Vector Machine (SVM) to learn constant and variable features of the log template. Different log parsers are built based on deep learning neural network models. For example, lexical information word embedding [298] and semantic word embedding [220] was employed by Liu et al. [239] to develop the Log2Vec-based, LSWE, which is a semantic-aware representation preprocessor technique for online parsing the switches’ log data in the datacentres. The embedding vector representation of logs’ tokens effectively extracts the semantic information and handles unseen out-of-vocabulary tokens of new log messages. Other log parsers have been developed based on different word embedding variants, such as [238, 311]. Tao et al. [299] and Zhao et al. [348] developed log parsers based on Bidirectional Encoder Representations from Transformers (BERT), and Nedelkoski et al. [256] developed NuLog parser based on the encoder transformer, which is a self-supervised learning technique to preprocess log messages by addressing the parsing task as masked language modelling. Setianto et al. [285] fine-tuned the Generative Pre-trained Transformer-2 (GPT-2) to parse Cowrie Secure Shell honeypot logs. Cowrie is a honeypot interaction system that engages malicious user attackers and monitors their actions by generating tracing logs to investigate the systems' vulnerabilities.

3.1.5 Other techniques

Different log parsers are built based on different approaches, such as heuristic rules, iterative partitioning, and the longest common subsequence algorithm. AEL [185] is a log parser based on a list of heuristic rules. For each pair, such as 'variable=value', AEL masks numerical 'value' with a 'v' sign. The heuristic rules-based parsers, in general, are hard to be extended to preprocess other logs. An iterative partitioning technique, combined with some heuristic rules, is used by IPLoM [232] to divide log messages into their log templates based on message length, token location and mapping relation. The longest common subsequence-based method was employed by [106] to develop Spell, an online streaming parser used to dynamically identify log pattern templates to filter out large system real-time log files. Messaoudi et al.[242] modelled MoLFI log parsing as a multi-objective optimisation problem and solved it by adopting an evolutionary algorithm, the non-dominated sorting genetic algorithm II NSGA-II algorithm [95] to search for log event templates in the solution space.

Most log parsers require a preprocessing phase to remove common variable values, such as IP addresses and numbers, though some parsers do not. Additionally, some parsers are open-source or deployed in production for industrial use (e.g., elastic [10]), while others are not.

3.2 Error Detection

This section elaborates on error detection's main existing approaches and related work. Error detection is the process of identifying anomalous patterns that deviate from a system's normal patterns of behaviour on log data (e.g., HPC system logs). These anomalies include potential faults, errors, or failures logged by system components; so error detection is also known as anomaly detection. Anomaly detection approaches divide log data into two groups: normal logs and abnormal logs. Often, abnormal behaviour detection is achieved using rule-based technique, which aims to (i) explore how to detect errors in terms of the potential features or correlations in log messages or (ii) identify anomalous log patterns that do not coordinate with common behaviours on log messages. Such rule-based methods (e.g., regular expressions [122]), however, are often not effective as expected because the rules have to be updated to adapt to new types of log patterns. Note that the amount of log messages and patterns may continue to evolve due to the ever-increasing scale and complexity of today's large-scale systems and the jobs assigned to these systems.

As illustrated in Figure 3.1, anomaly detection methods are classified into two broad categories: machine learning and deep learning. Machine learning refers to the methods for which the detection of anomalous patterns depends on the traditional machine learning techniques such as Logistic Regression, Support Vector Machine, etc. Deep learning methods, on the other hand, are the methods which detect anomalous patterns using Artificial Neural Network (ANN)s. Overall, in this section, we survey the approach/algorithm, the features to be utilised by the model, the learning method (supervised, unsupervised, or self-supervised), and offline or online mode. Unlike unsupervised techniques, supervised approaches necessitate labels for model training. The self-supervised methods are self-contained labels; more detailed explanations about the types of learning modes can be found in section 4.4. The online mode means its input is processed in a streaming fashion, one input data at a time. In contrast, the offline mode requires all the input data to be available before detecting the anomalies.

3.2.1 Machine Learning Approach

Different anomaly detection approaches (e.g., classification, clustering, regression, etc.) adopt various machine learning models. Traditional ML techniques are utilised to map the event count vectors onto a vector space effectively. A log sequence vector is considered to be anomalous if it deviates from the majority (i.e., it violates certain invariant relations among the log events counts).

Feature engineering is one of the most crucial steps in building ML models. It aims to extract, identify, transform, and organise essential features from raw data (log data) in a suitable way before applying an ML model. Examples of feature engineering methods include imputation methods (e.g., mean substitution, mode imputation) for handling missing data, variable transformations (e.g., normalisation, one-hot encoding) for transforming variables' values from one scale to another using such mathematical functions, or creating new features from existing ones.

Various anomaly detection and classification techniques have been built based on different machine learning algorithms, which are classified as follows :

Clustering Approach

The clustering based log anomaly detection is an unsupervised learning method that aims to divide log data features into a number of clusters based on some similarity or distance metric such that features in the same group are more similar and dissimilar to other features in other clusters. A high percentage of anomalies can be attributed to clusters with very few log data instances. The new log data is classified based on its features to one of the available clusters; if not, a new cluster is created. Ning et al. [260] designed the Heterogeneous Log Analyzer (HLAer) tool based on pair-wise similarity utilising the hierarchical clustering algorithm OPTICS [42] for heterogeneous logs categorization, format recognition, indexing, information retrieval, and outlier detection. Hamooni et al. overcame some of HLAer's shortcomings by developing LogMine [153], which is similar to HLAer in that it also employs a hierarchical clustering technique to abstract heterogeneous log messages. However, LogMine [153] circumvents some of HLAer's shortcomings. For example, HLAer was resistant to heterogeneity and was insufficient for abstracting big log files due to the memory demand and communication cost. On the other hand, LogMine employs MapReduce, with no assumptions about the log messages' features and no requirements for user intervention. Lin et al. [217] proposed LogCluster, a problem identification technique based on an agglomerative hierarchical clustering technique that takes into account all of the features of the logs generated by online service systems. It does so by assigning different weights to log events and grouping them into clusters based on their log sequence pattern similarities. Moreover, to identify the critical issues in service systems, He et al. [162] introduced the Log3C model based on system key performance indicator (KPIs) logs. First, a cascade clustering approach is used to categorize logs efficiently. Then, a linear regression model is employed to pinpoint the underlying causes of degrading key performance indicators. In general, the proposed clustering techniques are easy to use because they do not require labels for detecting anomalies; however, this method is very sensitive to distance metrics and clustering parameters, which can result in limited accuracy. Besides, the clustering results can be subjective and need expert knowledge to distinguish between identifying anomalous from normal patterns.

Graph Mining Approach

Graph-structured models have been broadly used to detect the anomalies and health states of large-scale systems using graphical features. Representing the log messages through graphs facilitates the capturing of entities of HPC components (i.e., graph nodes) and their relations (i.e., edges); this can help with gaining insights from the log data. The graphical features are typically used to generate a graph model characterizing the system health state, such as the flow of a process's execution, to identify hierarchical or sequential relations between the components of the system and log events. These types of relationships include dependency, co-occurrence, among other things. For instance, Nandi et al. [254] used the Control Flow Graph (CFG) method to detect abnormal runtime behaviours of distributed system applications from execution logs. This approach can identify two types of anomalies, namely sequence anomalies and distribution anomalies, by calculating the closest neighbour groups to more precisely discover the temporal co-occurrence of log events. An unexpected child's absence from a parent node inside the allotted time window indicates a sequence anomaly; on the other hand, a violation of an edge probability indicates a distribution anomaly. Moreover, Fu et al. [118] proposed a graph-based model to detect two anomaly classes: faults occurring during the execution pathways and low execution performance. The study characterized each system module's workflow as a state transition graph by learning Finite State Automatons (FSAs) via the log sequences constructed from log data. Each FSA transition is considered a log key sequence. The time consumed and circulation number are both collected for each state transition to detect short transition times and low transition loops. Execution low-performance changes can be captured using a Gaussian distribution technique and selecting a proper threshold. A Probabilistic Penalty Graph (PPG) is employed in [247] and [81] to detect "unexplained" log message patterns. These two approaches were designed to simulate the temporal behaviour of log events. Then, anomaly detection is achieved by evaluating log sequences that cannot be "explained" by any constructed path. Various graph-based anomaly detection approaches have been introduced by utilising neural network architectures. For instance, Xie et al. [324] presented LogGD, a technique for log-based anomaly detection utilising the power of the Graph Transformer Neural Network, which integrates graph structure and node semantics. Graph creation, graph representation learning, and graph classification are the three components of the LogGD framework. Nettie et al. [257] presented a model that is based on RF to categorise the errors that can occur in HPC systems nodes. Using the graph techniques for anomaly detection in large-scale systems has several advantages, including high ability to detect complicated erroneous patterns and high flexibility to scale massive volumes of data. However, the used algorithms and the logs quality can impact the accuracy of the results, and it also requires powerful processing resources.

Support Vector Machine Approach

Support Vector Machine (SVM) algorithm is used by many researchers to classify erroneous log entries or log sequences (i.e., anomalous) from non-erroneous ones (i.e., normal). Different statistical features are used to detect abnormal instances from normal ones. SVM is a supervised ML algorithm method that aims to find a hyperplane in an N-dimensional space (i.e., N features) that distinctly separates the classes of logs' data points. Determining the hyperplane is an optimisation task aiming to maximise the distance between the hyperplane and the most immediate log data point for the classes. A new log message or sequence is positioned above the hyperplane; it is classified as an anomaly while tagged otherwise as normal. For example, based on six log message features (e.g., the number of events in a time window, accumulated events number). [214] utilised SVM and other machine learning (ML) methods to detect anonymous IBM BlueGene/L systems logs. Kimura et al. [198] applied the SVM method with the Gaussian kernel on other statistical features, including periodicity, burstiness, frequency, and correlation with failures and maintenance for proactive failure detection on large-scale network logs. Furthermore, He et al. [161] compared between three supervised methods (logistic regression, SVM, decision tree) and three unsupervised methods for anomaly detection on different large-scale systems' logs with different settings. It was found that SVM achieved better accuracy than the other two supervised methods. Additionally, Meng et al. in [237] and its extension [241] proposed LogClass to identify individual errors for real-world switch logs and HPC log datasets using partial labels based on Positive and Unlabeled Learning (PU Learning) [261] and support vector classifier techniques. The LogClass framework can be outlined as follows:

i. Manual partial positive labels and PU Learning are employed to address unlabeled log problems.

ii. The SVM method is utilised to calculate the similarity of the word tokens' combination between a given log message and labelled anomalous logs (i.e., partial positive labels).

iii. LogClass also proposed a novel method named TF-ILF (Term Frequency-Inverse Location Frequency) inspired by TF-IDF (Term Frequency-Inverse Document Frequency) [297] to weight the log tokens in the construction of the feature vector phase.

LogClass is the most related method to our model [34] detailed in (chapter

5) where both proposed to classify log entries to faulty and non-faulty based on discriminative linear classifiers. However, we used SGD-LR to construct a sentiment lexicon for error detection & erroneous component identification in large-scale systems. Our method efficiently captures the developer's sentiment features from the (source) system log data that are labelled to automatically label the massive number of unlabeled logs of different (target) systems.

Frequent Pattern Mining Approach

Frequent pattern mining (e.g., association rule mining) approach is unsupervised technique that identifies frequent patterns, relationships, and correlations within datasets. Various anomaly detection techniques have been built based on different frequent pattern mining approaches to identify the log patterns within a log message that appears the most frequently and depict the normal behaviour of a system. Anomalies are the occurrences of log patterns that do not follow frequent patterns. The existence of certain log messages, as well as their order, can form patterns. For instance, the framework developed by Lim et al. [215] employed frequent pattern and statistical analysis techniques on logs generated from large-scale telephony systems to demonstrate the efficacy of these methods in identifying system failures and abnormalities. Furthermore, Lu et al. [224] developed CloudRaid to mine log sequential order patterns to detect erroneous behaviour or bugs in distributed systems. Moreover, Lou et al. [223] introduced an invariant searching method to automatically mine invariants from console events groups. These invariants can measure the underlying linearity of the program workflow to detect anomalies in large-scale systems such as Hadoop and CloudDB. Similarly, Farshchi et al. [115] developed a statistical regression approach to mine the linear correlation and causal associations between log messages and metric changes that show the status of cloud systems. The model's outcome is employed to construct a set of assertions utilised for anomaly detection during the runtime execution of cloud application tasks. Therefore, anomalies are detected if there is any non-compliance between the new log message with these metrics. Frequent pattern techniques can prune infrequent logs and classify them as anomalies without requiring labels (unsupervised approach). However, it can be computationally intensive (e.g., requiring huge memory) and produce many irrelevant patterns, resulting in limited accuracy.

Other ML Approaches

There are various anomaly detection approaches that do not fall into any of the aforementioned machine learning technique categories. For instance, for network failure monitoring, Yamanishi et al. [329] applied HMMs (Hidden Markov Models) on syslogs to monitor network failure symptom detection. The logistic regression model was also one of three supervised log-based anomaly detection methods in [161], where event count vectors were constructed from log messages and trained on labelled log data. Molan et al. [246] developed a bayesian-based classifier named TrueExplain to identify anomalies in supercomputers nodes; it outperformed other ML models such as Decision Trees (DT), Random Forest (RF), Nearest Neighbors (KNN), and Radial Basis Function SVM (RBF-SVM). Moreover, some features of the log data are not essential to detect the erroneous behaviours of large-scale components. These features are captured and removed by converting the log data with high dimensions into a representation with low dimensions in the way that some of the significant characteristics of the original log messages can be preserved in the low-dimension space. The most well-known unsupervised method of this dimensionality reduction approach is Principal Component Analysis (PCA). It works by checking if the gap between the data points and the first K main components is bigger than a threshold. If so, it indicates an anomaly. [327] projected log data points to K principal components; the faulty behaviour can be detected if the projected distance for new logs exceeds a certain threshold. Furthermore, in [148], the authors combined PCA with entropy and mutual information to perform error detection in supercomputer clusters based on two types of logs: resource usage data and event logs. In addition, the method in [204] uses PCA and independent component analysis (ICA) to identify outlier nodes in HPC systems. ICA is another effective dimensionality reduction approach used to reduce the log messages' features, seeking more accurate detection.

There are also some existing works leveraging sentiment related techniques for log analysis. Allen et al. [38] is the first research group that utilized a sentiment lexicon through a pre-built library called IBM Watson API to analyze software logs and assign sentiment scores for log data. Yadwad et al. [328] applied machine learning and time series models (e.g., PCA, Naïve Bayes, and logistic regression) on combined data of the social tweets, emails and logs for service outage detection. By comparison, our work [34] focused on the domain of a large-scale system. Furthermore, our domain-specific sentiment lexicon items are extracted automatically with the use of a machine learning-based technique, since a feature's sentiment is affected by the domain in which it is used. Allen et al.[39] also proposed a method based on using at least keyword and synonym matching percentage analysis criteria to classify log messages' levels in applications code. In contrast, our model [34] was designed to detect faulty components and errors of large-scale systems based on AI technique.

3.2.2 Deep Learning Approach

Although traditional machine learning approaches are faster, less complex, and can learn from small data in comparison with the deep learning methods, the deep learning algorithms have been more effective on unstructured data with high accuracy. Furthermore, they require less human intervention in feature engineering before the data is fed to the training phase. The deep learning architectures are utilised in many fields and research areas, such as computer vision, NLP, business intelligence, healthcare, and security, to name a few. Due to the extraordinary capability of deep learning to capture complex relationships, multiple anomaly techniques have been developed based on various neural network architectures. These deep learning-based models are more suitable to represent log data (e.g., [273]) that is typically unstructured, complex, huge, redundant, and incomplete. As stated before, deep learning-based log parsers are developed as explained in section 3.1.4, failure diagnosis and failure prediction models, as will be illustrated in section 3.3.3 and section 3.4.4, respectively; moreover, this current section will go on to clarify anomaly detection models based on the approaches they take. There are four main deep neural network design architectures with many variants for each category: Convolutional Neural Network (CNN), Recurrent Neural Networks (RNN), Generative Adversarial Networks (GAN), and transformer neural networks. Thus, in this thesis, we divide the DL-based anomaly models into four categories: CNN-based approach, RNN-based approach, GAN-based approach, and transformer-based approach.

Convolutional Neural Network (CNN) Approach

Convolutional Neural Networks (CNNs) are known for their dominant performance in computer vision tasks due to their convolution layers. Various log-based anomaly detection techniques employ this architecture to discover abnormalities in HPC systems' health state. The general idea of using CNN for anomaly detection techniques depends on considering the log data as sequential data with one dimension (i.e., a one-dimensional matrix); thus, an embedding layer and a one-dimensional convolutional network are required to perform this task. [227] leveraged CNN to investigate the hidden complicated relationship in logs and find abnormalities in big systems' logs. This model consists of three different layers: 1-D convolutional layer, a dropout layer, and a max-pooling layer; these are all preceded by a logkey2vec embedding layer. In this method, logkey2vec is employed to create the embeddings by mapping the log keys to embedding vectors. Embeddings are then fed into convolutional layers with various filters, each of which has a width equal to the length of a set of log lines. The maximum value for each feature is then selected by applying a max pooling layer. In order to generate the result for the probability distribution, a softmax layer is added at the top of this network architecture. Hashemi and Mantyla [158] presented a hierarchical classifier, OneLog, to detect large-scale system errors using the character-based convolutional neural network. Similarly, [64] Cheansunan and Phunchongharn proposed a CNN-based anomaly detection framework for a Hadoop Distributed File System (HDFS). Furthermore, Wang et al. [313] presented a new lightweight log anomaly detection algorithm based on the Temporal Convolutional Network (TCN) for anomaly behaviour detection supported by the use of word2vec and post-processing algorithms (PPA) to create a low-dimensional semantic vector space. As a result of these two changes, the detection performance of a standard TCN is enhanced, while the number of parameters and computations is drastically reduced. CNN achieves a high accuracy in detecting the HPC system anomalies because it can learn hierarchical logs representations, capturing local and global features. However, similar to other deep learning methods, CNNs significantly depend on extensive training data.

Recurrent Neural Networks (RNN) Approach

As explained in (section 2.3.1), RNNs are a category of neural networks that have attained superior results on sequential data such as textual and time series data. This superiority comes from their flexible designs allowing previous outputs to be used as inputs supported by different hidden states. Long Short-Term Memory (LSTM) (either single or dual) and Gated Recurrent Unit (GRU) are the most famous variants of this class and have been widely employed to learn sequential patterns in log messages for detecting anomalies in large-scale systems. Anomalies are captured when any deviations occur from the normal path. For anomaly detection in HPC systems, Du et al. proposed a log-based solution called DeepLog. In DeepLog, a log message is shown as a sequence of tokens with specific patterns that can be easily modelled by the LSTM neural network to learn normal log patterns from normal execution and consider any deviations from that as abnormal behaviour. Furthermore, Meng et al. [238] proposed a semantic-based log anomaly detection approach, namely LogAnomaly, to model log data as a sequence in a natural language. This method proposed template2vec to represent the log messages inspired by the embedding mechanism word2vec in NLP to capture the semantic meaning hidden in log data; it is also complemented by the LSTM neural network and is able to identify sequential and quantitive log anomalies of real-world large-scale systems at the same time. Given that the log data lacks stability, Zhang et al. presented [346] LogRobust, a technique to extract capturing logs' semantics by utilising TF-IDF [277] weights and off-the-shelf word embedding, FastText [187]. Then, the Bi-LSTM model is employed to detect anomalies in Hadoop and other large-scale online service systems. Anomalies in software execution path logs are identified using a Siamese network structure on top of LSTM layers in SiaLog [159]. It showed more efficient results with low-cost training time than other neural network baselines. Zhang et al. [338] utilised the Bi-LSTM neural network to develop a sentiment analysis model, SentiLog, for analysing parallel file systems (e.g., Lustre and BeeGFS) logs and detecting their anomalies. This work showed promising results; however, its model is trained using the source code logging statements, which is not easily accessible, rather than the more available runtime logs. Furthermore, Yang et al. [330] introduced PLELog, a GRU-based detection model, to handle the issue of inadequate labels of log data. PLELog applied on open-source log data of large-distributed systems classifies log sequences into two categories: normal or anomaly. Studiawan et al. [295] also utilised the GRU model to propose, Pylogsentiment, which is an anomaly detection technique for OS logs of different systems (including HPC systems) based on utilising positive and negative sentiment features. Furthermore, based on the context and content attention model, Studiawan et al. [296] employed the GRU technique to identify aspect terms and the corresponding sentiments to extract events of interest from log files in the forensic timeline. In comparison, as presented in our paper [34], we built sentiment lexicon based on stochastic gradient descent logistic regression algorithm to detect errors of large-scale systems and faulty components (e.g., nodes).

Generative Adversarial Networks (GAN) Approach

Effort has also been devoted to exploring the feasibility of Generative Adversarial Networks (GAN) for anomaly detection. The GAN-based approach [136] falls under the category of unsupervised learning; in this deep learning method, new or synthetic data is generated depending on patterns encountered in the original dataset (input). GAN networks consist of two main parts: a generator and a discriminator. The generator is an encoder that aims to generate synthetic data samples via the features during its learning phase after performing certain transformations. The discriminator serves as a critic and comprehensively understands the problem domain and generated data. The discriminator receives both joint actual and generated samples to classify these data as either fake or real. As an example, Zhao et al. [348] proposed Trine, a GAN-based model including three transformer encoders to identify anomalies in system log data. On the one hand, the first encoder aims to extract feature representations from the log message entries. On the other hand, the second and third encoders are used as a generator and discriminator for generative adversarial networks, respectively. This research addresses the anomaly detection task as a binary classification (normal or abnormal). Similarly, Xia et al. [321] presented LogGAN, an LSTM-based generative adversarial network utilising permutation log event modelling to detect systems' anomalies. The two main parts of LogGAN consist of (i) a generator and (ii) a discriminator. The generator aims to capture the actual log training data distribution and synthesises plausible instances, while the discriminator seeks to differentiate fake cases that are constructed from real and synthetic datasets. The trained LSTM-based generator identifies whether the upcoming log message is normal or anomalous based on the latest log entries. By constructing realistic log samples to increase training data, GAN-based solutions can potentially increase detection accuracy. Nevertheless, training GANs may be computationally costly and challenging to stabilise, resulting in a more extended training period and limited performance.

Transformers Approach

Various studies have utilised self-attention and parallel learning [305] with different transformer variants to detect HPC errors and anomalies. Transformer-based architectures have been created to fix the limitations of RNN-based models using the self-attention mechanism to train data in the parallel and self-supervised learning modes. The inputs (log events in our case) interact with each other (self) in parallel to determine which ones require more focus (i.e., attention). The final outputs are vectors that aggregate these attention scores. For instance, [144] proposed LAMA, self-attention-based model to detect anomalies of sequential log events. LAMA's pipeline is passed through three main phases: the embedding phase, the learning and attention phase, and the anomaly detection phase. [318] also employed the transformer-encoder architecture to develop an unsupervised anomaly detection technique called A2Log. Each log message is assigned an anomaly score through the self-attention neural network in this approach. Then, the anomaly decision is determined using an unsupervised method that applies data augmentation on the normal training data to determine the anomaly decision boundary value. Furthermore, to prevent the loss of information due to log parsing errors such as OOV (out-of-vocabulary) words and semantic misunderstandings, [206] proposed NeuralLog, which is another anomaly detection approach. Preprocessing, neural representation, and transformer-based classification are the three phases that comprise NeuralLog. Preprocessing the logs is the initial stage during which log messages are divided into a set of word tokens, and all non-character tokens are removed. Following this, BERT is employed to encode the semantic meaning of log messages into a semantic vector. As a result, NeuralLog can avoid the loss of essential information from log messages. NeuralLog adopted the WordPiece tokenisation [281, 319] which is capable of handling OOV terms issues. Finally, the transformer vanilla model [305] is utilised to identify log sequence anomalies. Combining both the learning of the log events' dependency relationships and log sequences' proximity at the same time, Yongzheng Xie et al. [323] proposed LogDP, a log-based anomaly detection approach. In this study, normal log events patterns are divided into dependency patterns and proximity patterns. The former refers to the log events that have occurrence relationships among them, whereas the latter is associated with events independent of other events. Normal patterns of dependent events are learned using dependency, while independent events are identified using proximity. Thus, any log sequence that deviates from the normal pattern is considered anomalous. Other research studies utilised self-attention with different transformer variants for error and anomaly detection, such as LAnoBERT[207], LogAttention [108], and [206]. However, in our two frameworks Clairvoyant [35] and Time Machine [36], we utilised self-attention and transformer neural network architecture to predict HPC system components' failures and lead time to failures.

Many other studies have been proposed for error propagation and fault tolerance (e.g., [146, 176–184]).

3.3 Failure Diagnosis

Failure diagnosis is a process that includes all approaches and tools to identify error propagation paths that lead to failures [77]. This means that failure diagnosis is a root cause analysis that focuses on tracing the pathway to the root-cause (i.e., faults events) that is likely to be symptomatic of the failure in question. Failure diagnosis in HPC systems is a very challenging and complex process aiming to resolve the problems that reduce the system reliability. According to the definitions of error, fault, and failure provided in [205], the goal of failure diagnostic techniques is to pinpoint the errors that result in the malfunction observed by end users. With the growth in sophistication of contemporary large-scale systems, it is very challenging to effectively separate the relationships between the error, the failure, and the manifestation experienced by the end users. In the last decades, log data has become the primary resource for developing failure diagnosis tools. Furthermore, log messages and resource usage data are the two types of log data employed in many studies in which the system administrators examine the cause of failures. These logs are interleaved in time, and only a tiny percentage of the events are pertinent to the diagnosis of a particular failure. Many approaches have been deployed on automated log-based failure diagnosis for the HPC and distributed systems; these can be categorised into three main categories: rule-based approach, statistical correction approach, and machine learning & deep learning approach, as is explained below.

3.3.1 Traditional Rule-based Approach

Many traditional rule-based techniques have been developed to diagnose failures in large-scale system logs, however, such rule-based approaches are often inflexible. This is because the rules must be adjusted when new types of log patterns are generated. Specifically, such approaches depend on a set of predefined rules derived from experts in the domain of failure diagnosis. Two typical examples are LogMap [65] and SherLog [336], where a set of rules is predefined based on two inputs: (i) log messages collected during the failed task execution, and (ii) the source code. This is to reconstruct the execution paths of failures from log messages with respect to their corresponding logging lines in the source code to assist the developers in analysing the paths of failures with their constraints and variable values. These studies face difficulties with redefining the predefined regular expression rules when new failure paths arrive or are re-employed in different systems with different source codes and log data.

3.3.2 Statistical Approach

Statistics-based failure diagnosis methods have also been proposed for years. Lu et al. [226] quantitatively measured the degree probability of underlying root causes of failures in Spark log data to define a weighted combination threshold constructed from seven factors. The authors employed a statistical approach, including the mean and median of job running time and the standard deviation of the entire data, for the diagnosis and detection phases, which are preceded by log preprocessing and feature extraction phases. Also, a fine-grained root-cause analysis was made feasible by Zheng et al. [351], who proposed an automatic root-cause diagnostics tool for HPC systems to identify the failure layer combined with the failure time and location of an error event that causes its corresponding failure based on RAS logs of the IBM Blue Gene/P system. This mechanism involves four phases: (i) the parsing phase to preprocess the system logs; (ii) the information fusion phase for synthesising data from RAS, job, and environment logs; (iii) the layer identification phase to specify the searching space by analysing numerous logs across the system software, application, and hardware components; and (iv) the time and location identification phase to identify the event that caused the failure and trace its location and occurring time. The authors identified the events of a node power error, insufficient memory error, invalid memory address error, and machine check error as the main reasons for BPC clock failure, the application out-of-memory failure, network torus sender failure, and kernel panic, respectively. Mdini et al. [235] utilised a large number of logs stored in a Hadoop-distributed file system to propose an automated model, ARCD (Automatic Root Cause Diagnosis), for root cause diagnosis in the different devices of large mobile networks based on statistical graph correlations. Furthermore, Das et al. [93] applied statistical correlations of internal and external logs to diagnose the breakdown of compute nodes across five HPC systems. The main finding from this study is that though hardware and software errors drive failures, the fundamental cause is the application malfunctioning which is inducing the system nodes to fail.

There are some other diagnostic frameworks which combined log messages with resource usage data to avoid the problem caused by the incompleteness issue of log messages. Resource usage data indicates the amount of resources consumed or produced by all the jobs in the system, including the system hardware performance, file system operation counts, and network device usage. For instance, the resource usage data file may contain different counters for the amount of memory, network, Lustre filesystem, or processors a job uses at a given time. CORRMEXT (CORrelating Resource use and MEssage logs and eXtracting Times) [77] is a diagnostics framework that analyses patterns of system errors and generates reports on the success and failure of error recovery protocols. FDiag [74], ANCOR [75], and CORRMEXT assume that when two message types are strongly positively correlated, the messages can be used to determine the likely cause of a given system failure. While FDiag and ANCOR focus on failure diagnosis, CORRMEXT focuses on identifying error propagation patterns. CORRMEXT processes rationalised message logs [152], Syslogs [75] and resource usage data [113]. It consists of three modules: (i) a data types extractor module, (ii) a correlation module, and (iii) a time-bins extraction module. The data types extractor module is composed of two data type extractors. A resource use extractor extracts resource use counters from resource usage data and organises these counters by time-bins into a resource use data matrix. The authors defined a time-bin as a time window of one fixed time interval. A message types extractor extracts message types from rationalised message logs and Syslogs and organises counts of message types by time-bins into a message types data matrix. The correlation module uses Pearson correlation and Spearman-Rank correlation algorithms to obtain correlation coefficients for the resource use counters and correlation coefficients for the message types. It uses a correlation threshold value of 0.8. If the correlation strength of two message types or two resource use counters is greater than or equal to 0.8, then the message types or resource use counters are stored in a list of strongly positively correlated message types or strongly positively correlated resource use counters. It uses Fisher's z-transform to test the significance of all correlation coefficients and Bonferroni Correction to identify false positives. A fixed threshold value is used to identify strongly positively correlated resource use counters and strongly positively correlated message types. The time-bins extraction module obtains differences between the variance of two adjacent time-bins for the strongly correlated resource use counters, and separately for the strongly positively correlated message types. EXERMEST (EXtracting fEatures and coRrelating resource use counters and MESsage Types) [78] enhanced CORRMEXT with several feature extraction methods and linked significant resource use counters and message types with node failures and error recovery protocols.

3.3.3 Machine & Deep Learning Approach

Various machine learning and deep learning algorithms have been explored to uncover the root-causes of their corresponding failures by transforming the diagnosis problem into a detection procedure based on the fact that failure diagnosis is the process of detecting the error events after the failure has occurred. For instance, Chen et al. [66] utilised the decision tree to diagnose request failures in the eBay internet service system by classifying the log patterns of requests that succeeded and failed during the faulty time. Then, the output graph of decision trees is used to interpret the root causes for failed requests. Furthermore, based on terabytes of log data, Bansal et al. [47] introduced DeCaf, a method for diagnosing and triaging performance issues in large-scale cloud services applications. The DeCaf framework is developed based on the Random Forest algorithm in conjunction with custom scoring functions. In fact, Microsoft adopted DeCaf for two large-scale commercial cloud services, where it successfully diagnosed the root causes of 31 unknown issues alongside 10 known cases. The authors of [234] showed that the one-class SVM model outperformed the decision tree and Bayesian network in diagnosing and troubleshooting the failures of large service distribution networks. In LogFlow[268], the authors demonstrated that the deep learning, LSTM architecture with the support of the Temporal Attention Gated Model [264] is a practical solution for inferring correlations between log events in supercomputers and cloud systems. LSTM is used to predict the upcoming flow of events, and the outputs of the attention module are used in LogFlow as the basis for determining correlations by assigning weights between log events. The weight scores increased as the correlation among log events increased.

3.4 Failure Prediction

Failure prediction is the process of forecasting failures before they occur to alleviate their effects entirely or partially by triggering one of the suitable proactive management techniques at the right time. It is more challenging than anomaly detection because it requires predicting the impending failures significantly ahead of occurrence time, such that various precautions can be executed in time. In HPC systems, popular proactive failure management techniques are used, such as task migration and checkpointing/restart. However, these techniques are expensive procedures and need to be used only when required, e.g., the computational overhead associated with these techniques may be exacerbated if they are wrongly triggered due to incorrect failure prediction. Thus, it is important to develop efficient failure prediction techniques so that the overhead can be kept minimized and tractable. Moreover, there are times when certain components may show abnormal behaviours; however, these abnormal actions can be transient (e.g., low memory) and recovered before they lead to failures [210]. In these cases, if failure is predicted too soon, it may lead to unnecessary precaution solutions; on the other hand, if a failure is predicted too late, it can result in costly impacts. Consequently, the prediction methods must determine failure at the right time, without being either too early or too late.

The log data generated by the components of large-scale systems are a primary resource for developing proactive failure prediction solutions that keep track of the system health state, such as the current state of each component, configuration, and other operational data. Hence, a considerable number of log-based research studies in academia and industry have been proposed to generate early warning alarms by predicting failures in large-scale systems so that these costly events can be prevented from occurring while the systems remain fully operating with their assigned jobs. Most of the proposed methods lack the ability to predict the failure lead-time, whereas others are attached with a sub-model to estimate the failure lead-time based on the log patterns. In this section, the log-based failure prediction methods for HPC systems are highlighted and categorised into four classes[1]: rule-based approach, probability

[1]The finer taxonomy (9 classes) for the HPC failure prediction methods can be found in

and correlation-based approach, machine learning-based approach, and deep learning-based approach.

3.4.1 Rule-based Approach

Similar to the rule-based methods proposed for anomaly detection, other studies have been presented to predict failures based on constructing a list of rules that alert the system administrators in advance before the occurrence of failures if conditions have been met. For instance, Watanabe et al. [316] presented an online method to forecast data centres' failures by extracting log event patterns as failure symptoms. The learned patterns are accumulated in a dictionary using Bayes' theorem to calculate the co-occurrence probability of a log pattern and a failure. Then, each incoming pattern is checked against all the patterns stored in the pattern dictionary, and the system triggers an alert if the probability of a match is high enough to exceed a predetermined threshold. Zheng et al. [350] proposed a method to predict failures (FATAL log events) and their lead times from logs collected from an IBM Blue Gene/P supercomputer cluster. Their methodology is based on the Genetic Algorithm (GA) and the Michigan encoding technique. The genetic algorithm generates and selects some rules based on the log events that precede the fatal events; then, the Michigan method is used to convert these candidate rules into genetic individuals. The GA method re-evaluates the selected rules using the Fitness function for optimal prediction accuracy. Other rule-based failure prediction methods are presented, such as [90, 143, 229, 315], in which the authors generally try to establish some predicate rules such as if/then statements extracted from offline log datasets. The rule-based failure prediction approaches necessitate periodic revision since system patterns and behaviours might change over time, owing to various variables such as updates on software and hardware; thus, new log patterns often emerge. Besides, each system requires creating its own set of rules.

3.4.2 Probability and Correlation Approach

Many studies predicted the large-scale systems based on calculating correlation association scores between the log events and failures, even if predicting failures

Jauk et al.'s survey paper [174].

was not the primary objective for some of these studies. For instance, over a hundred days of RAS event logs were compiled for the Blue Gene/L supercomputer system's health state behaviour in [213]. The authors explored the temporal and spatial characteristics of fatal failure events and the relationship between fatal and non-fatal failure events. Then, they developed straightforward failure prediction techniques based on spatial and temporal skewness in the failure distribution. As much as 80% of memory and network failures and 47% of application I/O failures may be anticipated. Fu et al. [119] introduced LogMaster, an approach to predict HPC system failures by leveraging the Apriori-LIS data mining technique to extract association rules from the unique characteristics and the correlation among log messages. The failure prediction is developed based on representing these associated rules using Events Correlation Graphs (ECGs). The correlation scores at ECGs' vertices are used to estimate the failure occurrence probability in large-scale cluster systems. The LogMaster framework limitations (e.g., low recall score) are rectified in [121]. The authors provided a three-step methodology for enhancing failure prediction accuracy in large systems by identifying events' dependencies correlations in Failure Generating Processes (FGPs). The event sequences are first divided into event groups based on a frequent occurrences clustering technique. Then, the causal relationships between events are identified within each cluster based on a Causal Dependency Graph (CDG). Finally, failure prediction rules are extracted based on the assumption that similar events occurring on similar nodes or from the same job applications usually exhibit similar operational behaviour. Ma et al. [230] analysed Redundant Array of Independent Disks (RAID) failure data from the world's largest backup enterprise systems. The analysis showed that the accumulation of reallocated sector (RS) errors is the main reason for degrading disc reliability; thus, these errors can be employed to predict disc reliability deterioration. Using these results, the authors implemented the RAISHIELD framework, which consists of PLATE and ARMOR components to monitor and prevent disc failures before they happen. In order to prevent RAID failures in production environments, PLATE provides an assessment of disc health by capturing the number of RS and proactively recognising unstable discs. ARMOR quantifies the RAID reliability deterioration and predicts vulnerable

RAID DGs, even in cases where individual discs have not yet suffered enough damage to warrant the activation of warnings. Ren et al. [275] proposed a mining frequent patterns-based mechanism and applied it to predict the failures of three large-scale systems, including a cloud computing system based on Hadoop that involves calculating the mining correlations among different events and failures. This mining process is conducted by leveraging the FP-Growth algorithm [155], which facilitates discovering the correlations and association relationship between failure signs (i.e., error events) and failures. Utilising the Self-Updating Cause-and-Effect Graph (SUCEG), [335] presented a method for failure prediction in large HPC systems that can automatically mine the causality among log messages from systems and update the failure prediction Cause-and-Effect graph (CEG) throughout the systems' life cycle. Then, the obtained CEG under different parameters is used to predict the failure events. This prediction process passes over five phases: log parser, event identification, cluster system event base (CSEB), SUCEG, and failure prediction. Other analytic-based approaches [55, 82, 120, 126, 128, 212, 301, 349] perform failure prediction by probability analysis, correlation analysis, or curve fitting. Multiple statics and correlation-based methods have also been proposed, such as the apriori algorithm in [121, 168] and a signal-based approach for failure prediction in [125, 127].

Time-series forecasting statistical methods have also been employed to predict failures' occurrence time in HPC systems through previous data to predict what will occur in the future. The prominent techniques for this approach are Moving Average (MA), Auto-Regression (AR), Autoregressive Moving Average (ARMA), and Autoregressive Integrated Moving Average (ARIMA). For instance, a combined technique including the Fault Tree Analysis and ARMA was built by Chalermarrewong et al. in [63] to forcest hardware failures within data centres. In this study, a set of system parameters that may induce cluster faults are monitored. The role of the ARMA model is to detect anomalies in the parameters' values and then transform them into binary signals that are then fed as inputs for the fault tree model. A warning is triggered to the cluster's administration in the case of a potential failure. The limitations of this study are that the evaluation was conducted only on simulated clusters and software failures were not considered. On the other hand, Hora [266] replaced Fault Tree Analysis and ARMA with the Bayesian networks and ARIMA to predict failures of the RDF Site Summary (RSS) reader application of Netflix's server. Furthermore, this study attempted to predict memory leaks, sudden node crashes, and system overloads. Lastly, Rawat et al. [274] proposed a prediction methodology that integrates ARIMA and stochastic models to predict the failures of virtual machines in cloud centres. Although the correlation-based approaches link some of the system failures with their preceded errors, they may not be able to capture complex and unprecedented failure patterns, which is a significant drawback.

3.4.3 Machine Learning Approach

In the last decades, the use of machine learning approaches for failure prediction in large-scale systems has attracted academic and industrial attention. A significant amount of failure prediction frameworks have been proposed to predict the failures in HPC systems based on traditional machine learning algorithms such as SVM, tree structures (e.g., Decision Tree, Random Forest), and Latent Dirichlit Allocation (LDA).

Liang et al. [214] extracted multiple features from the raw log dataset to capture the characteristics of failures and applied four ML classification techniques to predict fatal events from non-fatal events in IBM Blue Gene/L RAS logs collected over 142 days. They divide the entire time into fixed windows and predict whether a fatal event would occur in the next time window based on the event patterns in the previous time windows. The RIPPER (a rule-based classifier), SVM, the traditional KNN, and a customised KNN classifier were applied to RAS logs. The results showed that the customised KNN method outperformed the other three methods. Furthermore, the window size affects the accuracy of the failure prediction, where the prediction accuracy is much degraded when the window size is decreased. Moreover, this solution requires a fair amount of effort and data engineering to extract the features from logs before applying the ML methods, and this does not accommodate the need for fast online real-time failure prediction.

Mohammed et al. [245] inspected machine learning's predictive capabilities to enhance the accuracy of future components' failures in virtualised HPC cloud systems and applications based on five years of log data. The authors defined the prediction task as a multi-classification problem. They then executed a comparison among five failure prediction models, including SVM, classification and regression trees (CART), RF, KNN, and LDA. The SVM-based model achieved the highest prediction accuracy among the other models.

Fulp et al. [123] employed labelled Self-Monitoring Analysis and Reporting Technology (SMART) log messages and modelled the hard drive failure prediction problem as a classification task to apply a spectrum-kernel SVM algorithm. This method focuses on maximising the hyperplane to classify faulty from non-faulty next windows based on occurrence frequency features extracted from the previous log messages to predict probable failures in the sub-sequences.

An empirical evaluation performance investigation of seven commonly used machine learning classifiers is conducted in [320] to predict large-scale, six-month data centres' Dynamic Random Access Memory (DRAM) failures. These classifiers include SVM, LR, RF, decision tree, gradient boosting decision tree (GBDT), Extreme Gradient Boosting (XGBoost), and light gradient boosting machine (LGBM). The results revealed that LGBM and XGBoost obtained better prediction accuracy than other ML techniques. The same failure prediction task is then reformulated as an unsupervised anomaly detection task, and three anomaly detection techniques, including HBOS [135], iForest [219], and COPOD [211] are evaluated on the same dataset, with HBOS turning out to be the best solution among them.

Nie et al. [259] proposed a series of different ML models, which includes GBDT, LR, Artificial Neural Network (ANN), and SVM, in order to predict GPUs errors in HPC systems. The GBDT-based method outperformed other ML techniques in both F-1 score and recall.

Ana et al. [129] introduced a novel hybrid approach that combines signal analysis and data mining to predict failures in large systems integrated with failure avoidance techniques. Additionally, a failure prediction framework is proposed in [89] for extracting failure messages from Cray cluster systems that are symptomatic of compute node failures through three steps: first, both the

Cray system logs and the job logs are analysed. Second, the impact of compute and service node failures on the system and user applications is highlighted by providing frequency estimates of these failure occurrences. Finally, a node failure prediction technique is proposed, TBP (time-based phrase), using a ML technique called Latent Dirichlit Allocation (LDA) [53] and the Topics over Time (TOT) [312] model to enable the extraction of the crucial log entries that indicate node failures. In order to extract the intended entries, this method uses phrase likelihood estimation while considering continuous time-series data. This framework warns of impending failures with a lead time of 20 seconds to 2 minutes.

Yangguang et al. [210] assisted DevOps engineers by providing an AIOps (Artificial Intelligence for IT Operations) solution to leverage data analytics, including: alert data (message logs), spatial data (e.g., the location of nodes), and build data (i.e., the hardware and software configuration information of nodes) for node failure prediction in an ultra-large-scale cloud computing centre, SystemX at Alibaba, using three machine-learning (ML) techniques. The ML algorithms employed were LSTM [165], MING [218], and Random Forest [57]; the results show that RF outperforms the other two techniques in terms of node failure prediction accuracy and computation time speed.

There are a few more research studies [63, 243, 284, 309, 310] which have been proposed to predict the hardware failures of data centres (e.g., disk failure) that lead to the outages of cloud services and subsequent failures of the running jobs (i.e., applications). SVM, trees, HSMM, and other supervised machine learning-based methods [46, 117, 141, 342, 347] have been developed to predict task or job failures for cloud services in data centres based on numerical inputs, which include CPU usage, memory usage, disk usage, unmapped page cache, etc.

Other machine learning-based approaches have been proposed to predict HPC systems failures such as [30, 60, 62, 111, 123, 130, 139, 140, 201, 202, 204, 214, 228, 253, 265, 266, 279, 291, 294, 301, 315, 333, 353], which include failure prediction methods using the machine-learning techniques mentioned above or other ML algorithms such as PCA, Bayesian networks for hierarchical online failure prediction, and Hidden Semi-Markov models (HSMMs). Despite these contributions, these solutions have limited prediction accuracy or suffer from high computational overhead. Furthermore, most of these works focus on the failure prediction of one specific type of component, e.g., switches [342], compute nodes [210], memory [334], and GPUs in [258, 259].

The machine learning approaches assume that the data they process is unchanging and unrelated, but this is not the case in log data in supercomputers or cloud data centres. Accordingly, they are not suitable for dealing with log data in large-scale systems that change over time (i.e., sequence data) or have multiple variables, where the data at different points in time or with different features may be connected to each other. Moreover, the training models of traditional machine learning algorithms usually depend on annotated data that require manual labelling by a human expert, which is a time-consuming and labour-intensive task. Consequently, machine learning approaches are not the best solutions to accurately predict failures in cloud data centres [131]. To handle the challenge of machine learning-based methods, different studies have used deep learning techniques that have been demonstrated to be effective solutions. Deep learning models can facilitate unsupervised learning solutions and effectively capture long-term dependencies of log data, which are essential for accurately predicting failures in large-scale systems, as detailed in the following section.

3.4.4 Deep Learning Approach

Several studies have contributed in predicting HPC system failures using deep neural networks architectures that perform better with sequential data (e.g., Recurrent Neural Network(RNN) [41, 134]). The deep learning-based approaches (e.g., [91, 172, 225]) leverage deep neural networks that generally are composed of a significantly higher number of layers than the plain neural networks; thus, they often need relatively long training on top of a large amount of samples (i.e., log messages). Among all the approaches of failure prediction methods, the DL-based approaches have gained significant favour over other types mainly because of their outstanding prediction accuracy. These studies have shown the superiority of deep learning-based methods compared to traditional machine-learning methods such as decision trees, SVM, etc.

As stated before, the RNN is a type of deep neural network that has been widely utilised for many NLP tasks, especially for tasks that train on sequential data to learn and make predictions (e.g., text generation). Many studies also demonstrate the superiority of RNNs in predicting failures in large-scale systems. RNNs can handle input sequences with varying lengths using their internal state memory, which suits the sequence log events that depict the system health state, where errors and other events precede the failures. The RNN-based failure prediction methods are state of the art and mainly based on one of the four types of neural networks architectures: the simple RNN, LSTM, Bi-LSTM, and GRU.

We discuss several failure prediction approaches that leveraged RNN as follows. Cheng et al. [68] applied the standard RNN to predict job failures for the Google cluster workload traces with around 12,000 nodes based on job resource usage data (e.g., CPU and memory usage). Similarly, [325] employed the standard RNN to evaluate the health state and predict the potential failures of hard drives based on monitoring data called Self-Monitoring, Analysis, and Reporting Technology (SMART) to select the urgency level of the hard drive recovery and schedule repairs. However, training the standard RNN model to handle failure prediction requires understanding long-term dependency and can be challenging. This is due to the vanishing gradient problem, in which the gradient of the loss function diminishes exponentially with sequential data length. As error signals back-propagate through time, their size begins to decrease, making it difficult for the network to remember long-term dependencies. This issue is solved partially by improving the standard version of RNN to LSTM, BiLSTM, and GRU versions, which use gates to control information flow, allowing the network to keep a longer-term memory [134]. Therefore, different research studies proposed different failure prediction techniques for large systems by utilising one of these variants. For instance, Islam et al.'s work [173] aims to predict the jobs or tasks that eventually fail or are terminated in the Google cluster workload trace. To accomplish this, an LSTM model was trained on the consumption of resource data, including disk usage, memory usage, cache memory usage, CPU usage, etc. This LSTM-based model outperformed Feed-forward Neural Networks (FNN), SVM, and standard

RNN-based methods in terms of prediction accuracy. Both studies [68, 173] models were trained on numerical data (i.e., usage resources data) and were unable to predict when the jobs or applications are going to fail (i.e., lead-time prediction). Das et al. proposed Desh [91], which is a deep learning-based approach designed to predict compute node failures and their lead times based on the LSTM network. It involves three critical stages: (i) training to recognise chains of log events leading to failure; (ii) retraining the chains of log events to predict the lead-time to the failures; and (iii) online prediction/inference, obtaining high accuracy in HPC node failure prediction. In addition, Das et al. presented a method called Aarohi [87] to expedite the identification of the failure chain from log events. Aarohi is an extension of Desh that provides an online HPC systems node failure framework and expedites the prediction time to facilitate sufficient lead time and make proactive recovery solutions more feasible. Aarohi consists of two main phases: the first phase is offline learning by leveraging the Desh framework (LSTM-based) with log parsing to learn node failure chains. The second phase is the online prediction by scanning and comparing the stream of log events with failure chains, accomplished by regular expressions (RE) and the use of grammar-based rules. The incoming stream events are compared to these rules to predict future failures. The second phase speeds up the prediction time 27 times more than prior methods. However, this approach requires retraining and updating failure chains in the case that a new service or hardware/software upgrades are added. Furthermore, Lu et al. [225] leveraged a hybrid technique of the convolutional neural network and long short-term memory (CNN-LSTM) for Hard disk drives (HDDs) fault prediction based on SMART data. The SMART data used in this study were collected from a well-known data centre operator's 64 locations, covering 380,000 hard drives over two months. The study reached high accuracy for 10 days prediction horizon. Other solutions [27, 69, 105, 167, 218] also employed LSTM to predict the failures of HPC systems components, such as compute nodes, HDDs, GPUs, switches, etc.

The LSTM-based techniques can increase the failure prediction accuracy; however, they still face some weaknesses mentioned above. The LSTM-based method is designed to assign higher weights to predict the log events with shorter distances and lower weights to data that are more distant from each other. However, such a design inevitably suffers from significant shortcoming in that the accuracy of failure prediction may be greatly affected, as distant data can have a significant impact on failure in many cases. To address this issue, some other studies utilized the Bidirectional LSTM neural networks to improve prediction accuracy. Gao et al. [131] proposed Bi-LSTM-based framework for task and job failure prediction in large cloud centres. The learning in Bi-LSTM is conducted in a forward and backward LSTM network that can manage more input features to increase the failure prediction accuracy. This study showed that the failure prediction accuracy (in terms of F1 score, precision, and recall) can be improved compared to other state-of-art prediction methods, including HSMM, SVM, RNN, and LSTM. Likewise, Dai et al. [85] presented a failure prediction model to forecast the faulty nodes due to the CPU and memory faults for the migration of service in the cloud centres based on 12 fluctuated resource usage data metrics (e.g., CPU usage, memory usage). The Bi-LSTM model gains a better accuracy than the LSTM model, with over 95% and 97% accuracy for CPU and memory failures, respectively, whereas the LSTM model achieves an accuracy of approximately 90% for both types of failures (CPU and memory). In general, the Bi-LSTM-based approaches require a long training time with slower prediction speed because they combine two LSTM neural networks.

A few other researchers have utilised the third improved version of the standard RNN, Gated Recurrent Unit (GRU), to build their failure prediction solutions. Islam and Miranskyy [171], for example, enhanced the reliability through predicting failures of cloud centres based on multi-dimensional resource usage metrics data. They employed the GRU neural network, and the results demonstrated that GRU could be utilised as an effective model for predicting HPC systems failures. However, the GRU-based failure prediction methods have similar drawbacks to those of LSTM-based approaches.

The RNN-based approaches(e.g., [91] (Desh), [131], and [171]) are state-of-the-arts when it comes to predicting the HPC systems' failures. However, they suffer from non-trivial weaknesses: (i) long training time because of the absence of parallelisation in recurrence learning, and (ii) the vanishing gradient problem with loss of earlier "memory", which may cause limited accuracy. Clairvoyant [35] and Time Machine [36] approaches improve on the failure prediction approaches through (i) a self-attention mechanism and (ii) parallelisation, which are the crux of transformer neural networks and are explained in detail in Chapter 6 and Chapter 7, respectively. Specifically, Clairvoyant used one stack transformer-decoder to predict failures only. To enable the prediction of failure lead time, there are several key innovative designs proposed in the Time Machine model. Time Machine framework adopts a two-stack transformer-decoder architecture to predict not only failures but their lead times. The adaptation of the transformer-decoder to predict the failure lead times is based on a novel approach to self-attention: specifically, the Time Machine framework demonstrates how the self-attention mechanism developed for text prediction is used to predict failure lead times by encoding/decoding log events to map each log event onto its timestamp step during the training and prediction phases. In many domains (e.g., [61, 352]) except for fault tolerance, other transformer variants have been utilised for predicting time series as a regression task (i.e., supervised learning), which requires label data and results in limited accuracy. However, Time Machine is the first model to overcome these limitations by formulating the time prediction as a *self-annotated* multi-class classification problem by predicting the class for the failure lead time. Moreover, Time Machine can construct training instances in real-time because of our novel synthetic minority oversampling design. The Time Machine method can be generalised to other domains for similar time-based tasks (e.g., business, healthcare, booking business).

Additional failure prediction models have been developed for HPC systems. However, these solutions are mainly based on supervised learning methods, requiring extensive data labelling such as [202, 349]. However, most failure prediction solutions do not address the problem of predicting the lead-time to failure, such as [85, 131, 171].

Our solutions, Clairvoyant (Chapter 5) and Time Machine (Chapter 6), present self-supervised learning that does not need labels, unlike the deep learning-based methods, which are utilised as supervised learning-based methods. For instance, the failure prediction techniques that were applied to predict

Table 3.1: Comparative Analysis Of Failure Prediction Solutions

Solutions	**Method**	**Unsupervised**	**Lead-Time**	**Online**
Zheng et al. [350]	Genetic Algorithm (GA)	χ	χ	✓
Hora [266]	ARIMA (Autoregression)	χ	✓	✓
Fu et al. [121]	Episode Mining	χ	χ	χ
Berrocal et al. [50]	Void Search, PCA	χ	✓	χ
Klinkenberg et al. [202]	Supervised Classifiers	χ	χ	χ
Desh [91]	LSTM-based	✓	✓	✓
Goa et al. [131]	Bi-LSTM-based	✓	χ	χ
Islam and Miranskyy [171]	GRU-based	✓	χ	χ
Aarohi [92]	Compiler-based	✓	✓	✓
Clairvoyant [35]	Transformer-Decoder	✓	χ	✓
AtFP [209]	Transformer-Endoder-Decoder	χ	χ	✓
Time Machine [36]	Transformer-Decoder	✓	✓	✓

failures in cloud centre systems based on collected workload metrics (CPU usage, network usage, memory usage, disk usage, etc.) are supervised solutions requiring labels.

In October 2022, Li and Znati presented the AtFP framework [209], the second study published after our study Clairvoyant [35], which is an attention-based failure prediction model for large-scale systems. The evaluation showed that AtFP outperforms the LSTM technique, as we had demonstrated before. The AtFP model architecture relies on the encoder-decoder [305], whereas our models, Clairvoyant and Time Machine, rely on the GPT-2 decoder variant [272]. Moreover, the AtFP method labelled the input sequence as failure or no-failure; on the other hand, Clairvoyant and Time Machine are self-supervised approaches (i.e., no need for labels). Furthermore, our models have been applied on many data logs, while AtFP was applied only on one dataset. Finally, the Time Machine framework is capable of predicting not only the failures but also the lead times to the failures, whereas AtFP predicts the failures only. Table 3.1 shows a comparative analysis of existing failure prediction solutions.

Summary

This chapter reviews the previous research on automated log analysis for large-scale systems' reliability. The review divides prior studies into four domains: log (parsing) preprocessing, anomaly detection (i.e., error detection), failure diagnosis, and failure prediction. We also discussed the advantages and disadvantages of each approach carefully. This chapter serves as fundamental literature to other chapters. Most of the methods presented in this survey rely on HPC systems' log message data. The next chapter illustrates the large-scale systems (i.e., supercomputer clusters) model, log datasets, and fault model that are used to validate our methodologies.

Chapter 4

System Description, Log Data, And Fault Models

In this chapter, we present the system model (Section 4.1) and the targeted supercomputer systems as well as their system logs used in our experiments in (Section 4.2). Also, we describe the fault model and HPC system compute node failure in (Section 4.3). The definition of fundamental terminologies related to fault tolerance, log events, and machine learning are detailed in Section (4.4).

4.1 System Model

In most supercomputer clusters such as IBM Blue Gene/Q, Cray, Lonestar4, and Ranger, the typical system model consists of interconnected compute nodes that execute various jobs (i.e., application), and a scheduler that assigns these jobs to nodes. A node consists of a set of production resources (e.g., core-hours) to execute the scheduled jobs. In addition to enabling job executions, the supercomputer HPC system includes a range of components such as a parallel file system (PFS). The cluster components generate log messages generally stored in the form of files and collected on a central repository to capture the system's health.

Specifically, we describe the general supercomputer HPC system model targeted in this thesis as follows: In the HPC cluster system, there is a set of compute nodes $C = \{C_1, \ldots, C_m\}$ provided to execute user-submitted jobs $J = \{J_1, \ldots, J_n\}$ (e.g., weather forecasting, scientific simulations). We assume the compute nodes to be co-located geographically such that message delays between nodes and a central log repository is bounded. Also, we assume nodes to be time-synchronised, through the use of clock synchronisation, which is important for event timestamping. A job scheduler is used to assign the jobs to different production time-slots $T = \{T_1, \ldots, T_p\}$ on specific nodes. As the system operates, a bunch of log messages are generated to capture the health of the system and collected on a central log server or file [35, 150].

4.2 Production Systems and Log Data

We now describe the production clusters along with their logs used in our thesis, which is the fundamental information of our evaluation in the following three chapters. Table 4.1 shows the configuration of the five supercomputers generated six log datasets used in our evaluation. Specifically, these system are of various scales (from 200 nodes to 49,152 nodes), various interconnects (Infiniband and Aries Dragonfly), different file systems (Luster, MarFS, etc.), different processors, and different logging mechanism (IBM Blue Gene/Q, Ranger Syslogs, Ranger and Lonestar4 Rationalized Logs, Cray Log Messages, and Cray Console Messages). More details are described in the following text.

Table 4.1: The Configuration of the Supercomputers Used in our Evaluation

Supercomputer	**System Scale**	**Total Cores**	**Interconnect**	**File System**	**Processors**
Mira	49,152 nodes	786,432	5D torus	GPFS	PowerPC A2
Ranger	4,084 nodes	62,976	Infiniband	Lustre	AMD Opteron
Lonestar4	1,888 nodes	22,656	Infiniband	Lustre	AMD Opteron
Cray XC 30	5,600 nodes	133,824	Aries Dragonfly	Lustre/SuSE	IvyBridge
Cray XC 40	200 nodes	10,000	Aries Dragonfly	MarFS	Haswell/KNL

4.2.1 Blue Gene/Q Mira Cluster

The Mira supercomputer [11] used to be one of the most powerful supercomputers in the world, and its comprehensive system logs (including RAS log, job scheduling log, I/O behavior log, etc.) have been released to the public to promote the understanding of extreme-scale systems. Mira consists of 48 racks, each containing two midplanes. Every midplane has 32 compute nodes, each being facilitated with 16 active cores on a PowerPC A2 1600 MHz processor and a total of 16G DDR3 memory. As such, the entire Mira system consists of 49,152 nodes, and has a total of 786,432 cores and 786,432 GB of memory. The

Mira system's storage capacity reaches up to 35PB with 384 I/O nodes, and adopts a 2GB/s chip-to-chip-link 5D torus network, with each node containing 10 links with 2 Gb/s bandwidth. Mira's job-scheduling log involves 32.44 billion core-hours, about 1,300 users, and about 630 projects throughout its whole 5-year service [98], which creates an outstanding use-case for a resilience study.

RAS Logs

In the Mira cluster with diverse system logs, the Reliability, Availability, and Serviceability (RAS) log is our focus. The core monitoring and control system (CMCS) of the Blue Gene / Q systems is responsible for monitoring the hardware components, including compute nodes, I/O nodes, and different networks. CMCS reports monitored information as RAS event records that are used to provide facilities for detection, diagnosis, and prediction of failure. The event in the RAS log is identified by one of three severity levels (INFO, WARN, or FATAL). Although one message in the RAS log consists of 14 fields, we focus only on a few of them related to the system reliability, such as MESSAGE, MSG_ID, LOCATION, SEVERITY, and EVENT_TIME, as suggested by [97–99].

Table 4.2: Description of the Key Fields in RAS logs

Field	**Description**
Msg Id	Unique identifier of RAS event
Severity	Severity level of the message (FATAL, WARN, or INFO)
Event Time	Time stamp of the event (e.g, 2015-04-03-17.30.26.691211)
Location	Particular occurrence location (e.g., midplane, node board)
Message	Detailed description of the event

4.2.2 TACC Ranger Cluster

Ranger [18] is a well-known supercomputer, which used to be one of most powerful machines in the world (ranked 45th in 2008 Top500). The Ranger system consisted of 4,084, and has 15,744 quad-core AMD Opteron microprocessors in total (involving a total of 62,976 cores), which were connected via an Infiniband network. During its lifetime, this supercomputer served 4K+ scientists from 2,244 research projects, with over 3 million simulation experiments completed.

In our experiments, we adopt two real-world supercomputer system logs both generated by Ranger system, which have been widely used for failure analysis [34, 104, 149, 289]. The Ranger system adopted a well-designed monitoring and logging system, and its jobs were managed by the Sun Grid Engine [112]. Two types of system logs were generated by two *different* logging frameworks, called SysLogs and Rationalized Logs. In the following, we describe the two logs, SysLogs and RatLogs, in details.

Syslogs

The Syslogs dataset is produced by a specific logging system called syslog [25], which uses POSIX standard (very similar to linux cluster systems) to output logging messages. The syslog handles the logs via Linux's /var/log directory and keeps kernel system messages such as core dumps in /var/log/messages. Table 4.3 describes the five fields for each log event in Syslogs.

Table 4.3: Description of Syslogs Message Fields

Field	**Example**	**Description**
Timestamp	Feb 2 02:02:03	Occurrence time
Host	i152-304	The node on which the job ran
System Id	kernel (linux)	System's ID/type
Application	Lustre	Name of application
Text message	The ost_write operation filed with -122&key	Detailed message of one event

Rationalized logs

Compared with Syslogs, Rationalized logs (abbreviated as RatLogs) was a new logging framework for TACC supercomputers. Unlike Syslogs, RatLogs has a few additional fields to record more information, for example, job-ID (an identification number to identify each running job). Such a logging framework was adopted for analyzing the log-based failures effectively in the system. In particular, RatLogs can be used to parse unstructured log messages more efficiently. It may also commit error mappings and job failures directly. The detailed logging metrics and their descriptions are shown in Table 4.4.

Table 4.4: Description of Rationalized Logs Message Fields

Field	**Example**	**Description**
Job Id	2211223	Job's identification number
Timestamp	Oct 21 22:539:18	Time stamp
Host	i112-102	The node onw hich the job ran
Prog	kernel	Name of the protocol
Text message	X Northbridge Error	Detailed message of the event

4.2.3 TACC Lonestar4 Cluster

The Lonestar4 [8] HPC cluster system is also operated by the Texas Advanced Computing Center (TACC) [76]. It consists of 1,888 compute nodes, each with two 6-core processors, for a total of 22,656 cores. It has a total memory capacity of 44 TB and 276 TB of local disc space, with a potential peak computation performance of 302 TFLOPS [15]. It was operated from 2009 to 2015. As was described in the description of Ranger, a high-speed Infiniband network is deployed to provide the communication among all the nodes on Lonestar4; a job scheduler is employed to deal with job scheduling and resource management, and the Lustre file system is deployed to provide high-speed data I/O service.

Lonestar4 Rationalized logs

Lonestar4 write their events using unstructured rationalized logs (Ratlogs) similar to Ranger Rationalized logs in (Section 4.2.2) with the same logging framework to allow freedom for the formatting of logs. The logging metrics and their descriptions are similar to the Ratlogs format displayed in Table 4.4.

4.2.4 Cray XC30 Cluster

The fourth supercomputer system studied in this thesis is called Cray XC30 [93], which has been widely used in the resilience studies. The log messages were generated on the System Management Workstation (SMW) in the form of syslog format. The Cray system is composed of Login, boot, Service Database Node (SDB), as well as compute nodes, syslog, I/O and networking nodes. It has a total of 5600 nodes facilitated with IvyBridge processors, which are connected via Aries Dragonfly network. All information about service nodes are kept in the SDB. The shared file system (Lustre) managed by a boot node is connected with service nodes and compute nodes. The user job submissions are managed by Slurm under the Application Level Placement Scheduler (ALPS), such as aprun, apbridge, apshed, apinit etc. The system log used in this thesis covers 1 month of operation.

4.2.5 Cray XC40 (Mutrino) Cluster

Trinity Cray XC40 [16] was one of the most powerful supercomputers (ranked at #6 in the top 500) in 2018, which was managed under a joint effort between Los Alamos National Laboratory (LANL) and Sandia National Laboraties (SNL). Mutrino, sited at SNL, is a readiness test platform for the Trinity Cray supercomputer [56]. According to [21], Mutrino is identical the full Trinity system except for the size. Specifically, Mutrino/Trinity utilizes the Cray Aries Interconnect network to connect hundreds of Cray nodes, including 100 Haswell nodes and 100 KNL nodes. It has approximately two PetaBytes of DDR4 that can feed data to the processors at a rate of over 1PB/s. It is also facilitated with Burst Buffer which is composed of 4 PetaBytes of SSD drives running Cray DataWarp software that can move data in and out of memory at a rate of 6TB/s. Below the Burst Buffer, the system adopts the Lustre filesystem to keep the data for several weeks, and uses a so-called MarFS filesystem to stage large amounts of data for several months. The system uses an HPSS tape archive system for essentially permanent storage.

The Mutrino log data (Cray XC40) used in this thesis can be found online [21]. It spans 16 months, and comprehensively covers different states of the system, including standup, a variety of naturally occurring and induced network, electrical, thermal, and functional failures (such as Aries errors, DIMM failures). The log resides also on the System Management Workstation (SMW), which provides a single point of access to log data. Most of the log messages are stored in the form of multi-line text format, and some log files are stored in binary format. Most of the log messages depend on the syslog format (e.g., netwatch).

The detailed logging message format of both Cray systems will be described in detail in the next subsection because Cray XC30 and Cray XC40 are sharing the same format in logging messages.

Cray XC40 and Cray XC30 Logs

For both Cray XC30 and Cray XC40, there are two types of logs (console and message) used in our evaluation. The logging files (console and message) are stored in the p0-directories, and their formats are presented in Table 4.5 and Table 4.6, respectively. Specifically, they both contain timestamps and node ids (cX-cYcCsSnN) per line. The node id is represented as a sequential decimal number, which can be transformed to physical id, i.e., cX-cYcCsSnN format for identifing a node's physical location, e.g., which node (N), on which cabinet (XY), in which blade (S), and in which chassis (C). Additional references to boot messages and job logs provide the status of nodes and jobs over time.

Table 4.5: Description of Cray Console Log Fields

Field	**Example**	**Description**
TimeStamp	2016-01-11 16:11:36.347690+00:00	Time Stamp of The Event
Node	c5-3c2s4n2	Execution node
Message	PCIe Bus Error	The detailed message of the event

Table 4.6: Description of Cray Message Log Fields

Field	**Example**	**Description**
TimeStamp	2015-07-26T15:02:06-06:00	Time stamp of the event
Node	c0-0c0s0n1	Execution node
Application	ccrdhelper	The application related to the event
Message	Unable to determine BIOS type for node 1	The detailed message of the event

4.3 Fault Model

In this section, we detail the assumed fault model, viz. node failures, of this work. We subsequently explain the potential causes of these failures.

4.3.1 Fault Model: HPC Node Failures

HPC node failure is a state in which the operating system kernel hang-up, becomes unresponsive, goes stuck, or loops without ends, blocking other processes from executing and ultimately causing the nodes to shutdown [35]. In HPC systems, there could be many factors or different types of preceding errors (from hardware errors to software/application faults) that can result in node failures. The preceding errors that cause node failures are very diverse, including hardware issues (memory, GPU, network), OS process errors, file system errors, application errors, etc.

The consequence of these preceding errors may differ a lot. Some errors may induce failures very quickly because of their fast propagation: i.e., the sequence of log events between the first error message and the ending failure event could be very short. On the other hand, some other errors may take a long time before their corresponding failure occurs, corresponding to a lengthy sequence of log events with a relatively high delay between the first error message and the ending failure event. This could be due to, e.g., the system attempting a recovery that eliminates the error (e.g., message loss + re-transmission). However, the error may come back later and if recovery is unsuccessful despite many attempts, failure may ensue, with a long lead time. It is also worth noting that there could be many interleaved & irrelevant events recorded between node failures and their preceding error events, for both short and long sequences, making failure prediction more challenging.

In this thesis, we focus on the prediction of node failure events as well as their failure lead-times through our methods proposed in Chapter 6 and Chapter 7, which can also be applied to failures of other components (such as switches, GPUs).

4.3.2 Causes of Node Failures

A fault model specifies the way a system is expected to be affected by faults. Our designed failure prediction framework can be applied on various types of discrete faults at different levels, such as hardware, system, application level, file system, and at an aggregate supercomputer level. As a fault occurs, the resulting errors will result in error messages in the system log file. A failure to address the causes of these error messages will likely result in a system/application failure, which will also be logged.

The system faults have been classified into various fault models based on experts, and the following are some categorisation approaches [150]: (i) **Design-Runtime Fault Model**, which divides faults depending on their origin, with design faults that occur as a result of system design flaws and runtime faults that emerge during the production time. At the design phase, fault tolerance approaches (e.g., system testing) are utilised to reduce or remove these flaws, while error detection/prediction can capture erroneous log patterns induced by runtime faults that may result in failures; (ii) **The Hardware-Software-Human Fault Model**, which categorises the system faults based on how they are designed and the executed operations, with hardware faults owing to hardware defects, software faults induced by application or software flaws, and human faults caused by human activity. When these faults are activated, their corresponding errors are generated, which can be identified and predicted by error detection and failure prediction methods. It is important to emphasise that there are cases where the error events are not logged in the log file; thus, their corresponding faults may go unrecognised until the associated failures occur, as such failures have no signatures or patterns; (iii) **The Permanent-Transient-Intermittent Fault Model**, which categorises faults depending on how long they remain in the system(i.e., their duration). Permanent faults, which are generally caused by hardware issues, persist until they are fixed, whereas transient faults, such as temporary communication faults, disappear once they occur. Intermittent faults, such as a loose connection, arise and disperse temporarily but may return. Additional fault models have been suggested, such as how software faults are observed [138] or the faults' behaviour/impact [48].

Failures in one component of the system (such as network) can lead to a fault in another component of the system (e.g., memory). This is due to the fact that, through fault transformation, the interactions between the two components are impaired, causing the assumptions of the second components to be violated. This assumption violation is transformed into a faulty situation for the second component [77, 304].

4.4 Basic Definitions

This section provides some basic terms and definitions related to fault tolerance, log messages, and machine learning.

- **Log event (message):** Log event is represented as a line of unstructured text in the log file, which incorporates multiple pieces of information in the regard of different components or states of the system (e.g., NodeID, timestamp, JobID, etc.). Each log event/message is reported with a timestamp indicating the moment that the event occurs. In this thesis, we use the terms log message, log event, and event interchangeably

- **Log sequence:** A log sequence (event sequence) consists of one or more consecutive log events that are reported within a specified time window [148].
- **Fault**: Fault is a hardware or software defect within the system, which can lead to system failures (e.g., node crashes) or fatal issues (i.e. the illegal system states). Programming mistakes, hardware defects, and data transmission mistakes are examples of faults.
- **Error:** Error is an activated fault (or a situation through which the fault turns out to be manifest), which potentially leads to failures when the system deviates from the correct service state.
- **Failure:** An event that occurs when the delivered service deviates from the correct and intended service [45].
- **The failure lead time:** The failure lead time is defined as the time interval between the timestamp of the precursor log message (e.g., the error event) and the timestamp of the failure [93].
- **Job (Application):** Job or application is an execution instance of an application submitted to a supercomputer for completing some simulation work (generally in parallel). As a job is submitted, it will be first put in a queue waiting for the schedule, and then executed once available compute nodes are allocated to it. Various pieces of specification information are attached to HPC jobs (e.g., number of tasks, workflow, and other resource requirements). The number of tasks indicates the total number of individual tasks or units that comprise the entire job, whereas the workflow defines the steps required to complete the job. Other resource information to execute jobs also is needed, such as the number of CPUs, cores, memory, GPUs, etc. On a supercomputer, the jobs are generally involved in many different scientific application domains, such as climate

simulation, earthquake and cosmology simulation, quantum chemistry simulation, and fusion reactor simulation.

- **Chain (Sequence) length:** Chain (sequence) length indicates the number of the log events in a log sequence.
- **Failure chain (Sequence):** Failure chain (sequence) is referred to as the log sequence that ends with a failure event.
- **Online model:** Online model enables incremental learning as new data arrives; thus, the model's parameters are continually updated based on a continuous stream of incoming data accompanied by improvement in model performance.
- **Offline model:** Offline model is trained based on the entire accumulated data (i.e., batch) at periodic intervals (e.g., weekly, monthly, yearly); thus, it is called batch learning, and the model performance would decrease as new data have emerged, hence retraining with the entire dataset is required which includes the previous batch data and the new data.
- **Supervised learning:** Supervised learning is an approach in which training the models requires labelled data (i.e., requiring manual labels); thus, the model function focuses on mapping the inputs to the correct labels. This approach includes algorithms such as support vector machine, decision tree, random forest, etc.
- **Unsupervised learning:** Unsupervised learning is an approach in which the models are trained on unlabeled data. The unsupervised models seek to discover the hidden patterns among the input data. This approach includes algorithms such as K-means and apriori algorithm.
- **Self-supervised learning:** Self-supervised learning is an approach in which the models do not require manual labels; however, the models train themselves by leveraging some portions of data to act as labels. Thus, this approach is considered a subset of the unsupervised learning technique because it does not need human labels. This approach includes language modelling algorithms for NLP tasks such as BERT, GPT-2, etc.

Summary

This chapter presents an overview of the supercomputer cluster's system and fault model. Also, five production systems and their log datasets which will be used in our experiments in the following three chapters have been detailed with examples. Moreover, we defined some basic terminologies related to fault tolerance and log messages.

Chapter 5

Sentiment Analysis Model For Errors Detection In Large Scale Systems

Preface

Log messages generated by the components of HPC systems are the primary resource for developing error detection solutions and keeping track of the system's health state. However, as the size of these large systems expands, the number of log messages increases exponentially, which makes it challenging to identify faulty messages efficiently. We conjecture that these log entries capture the sentiments of system developers regarding the health of the generating components (e.g., a network timeout expresses a negative sentiment about the network health). Leveraging these sentiments, we propose novel sentiment analysis-based algorithms for error detection in large-scale systems. One approach for sentiment analysis is through the use of a sentiment lexicon, which, however, is often generated manually. In this chapter, we propose a novel approach for the automated generation of a reusable sentiment lexicon to support log analysis of different large scale systems, since these large systems often share similar components. Our contributions are four-fold. (i) We develop a machine learning-based approach to automatically build a sentiment lexicon, based on the system logs. (ii) Using the sentiment lexicon, we develop an algorithm to detect system errors. (iii) We develop an algorithm to identify the nodes and components with erroneous behaviours, based on sentiment polarity scores. (iv) We evaluate and compare our sentiment lexicon's performance with those of machine/deep learning algorithms, including Random Forest, XGBoost, Multinomial NB, KNN, and LSTM to detect three different supercomputers' errors: Blue Gene/Q Mira, Ranger, and Lonestar4. The results show the viability of our approach and outperform other related state-of-the-art approaches.

5.1 Introduction

There are two approaches to handling failures of large-scale systems: (i) failure prediction and (ii) error detection. Potential failures are forecasted, and proactive management actions are triggered to prevent or lessen them in the first approach. In the second approach, errors are detected as they occur, and measures are conducted to recover from them. This chapter focuses on the second task (i.e. error detection) based on the analysis of log messages.

Error detection, using system logs has been extensively studied for years. In [148], the authors performed error detection in supercomputers by combining entropy, mutual information, and PCA approaches. Several different works were generally based on detecting system anomalies (e.g., [54, 59, 161]). Further techniques based on NLP and AI have also been applied towards failure log analysis [107, 110, 322].

Works on error detection in HPC systems have focused on various aspects such as identification (i) of erroneous log entries [241], (ii) of failure-inducing erroneous execution sequence [148],[346],[147] and (iii) of detecting quantitative relationship among logs [195],[217]. This chapter addresses the first problem and focuses on the automated classification of failure log entries, thereby obviating the need for the time-consuming manual labelling of such entries. Often, automated labeling is achieved using rule-based techniques. However, such rule-based approaches are often inflexible as the rules need to be revised when new types of log entries are generated.

In this chapter, we explore a novel perspective on the problem of failure log analysis. We conjecture that log messages often encapsulate the sentiment of system developers, which pertains to the perceived health of the system. For example, if a typical log entry states a timeout issue on a particular node, this message implies that there is something wrong at the network level. That is, such a message indicates that the system developer has a negative sentiment about the current network status. The question is: can we leverage these sentiments to develop a sentiment analysis-based technique for failure log analysis? Our answer is yes, as we propose two novel algorithms for detecting error messages and faulty nodes and components in these systems. This is particularly helpful for error detection in systems with non-labelled logs (e.g., Ranger (4.2.2) and Lonestar4 (4.2.3), detailed in Chapter 4).

Sentiment analysis, which is a text classification technique that combines NLP and AI, is based on assigning weighted sentiment scores to the text entities within a word, phrase, sentence, or document. One class of approaches for sentiment analysis makes use of a sentiment lexicon where the focus is on developing specific list of words that carry cues of affection or sentiment, instead of using every word as a feature [189]. However, the development process of the sentiment lexicon has some weaknesses: (i) it is often generated manually, which is tedious and inaccurate to users; (ii) it tends to be domain-specific for efficiency reasons. We address these two respective problems as follows: (i) we develop a machine-learning approach that exploits the log sentiments to develop a sentiment lexicon to support the detection of errors in large systems[1] and identify the erroneous components or nodes and (ii) based on our observation that such HPC systems often share similar components such as OS, a lexicon for one system (i.e., source system) can be reused for another target system.

The fundamental principle of our design is that the sentiment intensity scores can accurately represent the system state, among which the system developers generally use very similar concepts or terms to record the events/messages across different systems. In fact, the system developers often use negative sentiments to log serious problems such as errors and failures, neutral sentiments to highlight informational messages indicating the system works as expected, and positive sentiments to mark the system faults/problems that have been fixed, which inherently captures the three main classes (sentiment polarities) of a log message.

However, to detect errors, sentiment polarity is not adequate. A sentiment intensity score keeps track of the strength of a sentiment, e.g., the features

[1]A system fault refers to a potential event that may adversely affect the system execution.

'failed' and 'unexpected' may be associated with higher negative scores than the features 'slow' and 'monitor', while the features 'recovered' and 'successfully' are assigned higher positive scores than 'normal' and 'valid' states. This potentially allows us to exploit system logs of a source system that are labelled with severity levels (e.g., Blue Gene systems 4.2.1) and extract their sentiment features to automatically label log entries of other unlabelled (target) systems as an unsupervised approach.

We draw four key research contributions of this chapter as follows:

i. We develop a ML-based method using stochastic gradient descent logistic regression, to automatically construct a reusable sentiment lexicon for such systems.

ii. We develop an algorithm for error detection based on the sentiment intensity score of log messages.

iii. We develop an algorithm to discover system components (e.g., nodes) which show erroneous behaviors based on sentiment polarity scores of messages logged by those components during a specific time period.

iv. We perform the evaluation using the system logs of three large systems: (i) Blue Gene/Q Mira (4.2.1), (ii) Ranger (4.2.2) and (iii) Lonestar4 (4.2.3) which were built by three different vendors - IBM, Sun, and Dell respectively. We also compare our sentiment lexicon's performance with ML/deep learning classification algorithms, including Long Short Term Memory (LSTM), Random Forest (RT), Extreme Gradient Boosting (XGBoost), Multinomial Naive Bayes (Multinomial NB), and K-Nearest Neighbor (KNN). Experiments show that our sentiment-based solution can efficiently detect error messages based on their associated sentiment scores, with an average MCC-score of 91% and an average f-score of 96%, whereas state-of-art ML/deep learning model (LSTM) obtains only 67% and 84% respectively. Our technique identifies error messages with an f-score of about 99% for Blue Gene/Q Mira system. Using the sentiment lexicon items extracted based solely on the Blue Gene systems logs, a majority of errors in Ranger and Lonestar4 logs can be successfully detected, with f-scores of about 94%, and 95%, respectively. This effectively shows that our technique can generate reusable lexicon for such systems, enabling the automated labelling of any system's logs.

The remainder of this chapter is organized as follows: Section 5.2 formulates the research problem and the objectives of this chapter. Section 5.3 presents the main steps of our approach. Section 5.4 shows the evaluation metrics and the experimental results performed on the logs of three large systems. This chapter concludes with a summary.

5.2 Problem Formulation and Research Objective

We formulate our research problem based on the three aforementioned supercomputers, including Blue Gene/Q Mira (involving 49K nodes with a total of 786K cores), TACC Ranger (involving 4K nodes) and Lonestar4 (about 2K nodes with up to 63K cores), which were built by three different vendors - IBM, Sun, and Dell respectively.

Without loss of generality, the general system model for a large-scale system (e.g. IBM Blue Gene) contains a set N of nodes, a queue of J jobs, a set T of production times, a job scheduler JS, and various software components such as a file system. The scheduler JS allocates the J jobs to the N nodes to execute during time period T. Further, the components write message logs in to a central writing container [76].

The problem that our approach addresses can be formulated as follows: Assume (i) a set of log messages is generated by a large-scale system, (ii) these log messages have different severity levels, and (iii) the message templates comprise system developers' sentiments (either negative, neutral, or positive), our objective is to develop an efficient approach that is able to:

i. Automatically construct a reusable sentiment lexicon intended for large-scale systems.

ii. Using the lexicon, identify system individual faults.

iii. Using the lexicon, develop an unsupervised approach to detect faults of other (target) systems that are missing the severity attribute information.

iv. Using the generated lexicon, identify the erroneous nodes and components to assist precaution (e.g., avoid node crash), thereby preventing job failures.

5.3 Methodology

This section details our approach which features sentiment analysis model for detecting errors in large-scale systems. We first explain our technique to extract developers' sentiment features with their weights from the source system to automatically construct lexicon items. Second, we explain our algorithm for error detection based on the sentiment intensity score of a log message. Third, we outline our algorithm to discover system erroneous components based on sentiment polarity scores of log messages.

5.3.1 Lexicon Construction Using Stochastic Gradient Descent Logistic Regression

In this section, we describe a machine learning based model that allows for the automatic construction of sentiment lexicons to detect errors in large-scale systems via the stochastic gradient descent logistic regression technique. A sentiment lexicon is a dictionary that consists of features (N-gram) associated with their sentiment polarity values, and these sentiment scores are estimated based on a model trained on a sample of log message templates. Stochastic gradient descent logistic regression was employed since it is a discriminative model which assigns high weights (sentiment scores) to the significant log message features that can distinguish error messages from non-error messages. To generate a sentiment lexicon for a large-scale system, our model requires four components:

- M input/label pairs of log message templates (x^i,y^i) where each input log message template x^i is represented by a vector of f^j features $[f^1, ..., f^j]$.
- the sigmoid (logistic) function to compute the estimated class $\bar{y} = \sigma(w.x + b)$ for each log message template.
- the cross-entropy loss function for features weights (i.e.,coefficients) learning through minimizing error on training log messages.

- the stochastic gradient descent algorithm for optimizing the cross-entropy loss function and updating weights.

This method performs two main steps to learn sentiment lexicon items (features and their weights).

Phase I. Log Message Template Preprocessing

In the first step, we use M representative training log message templates (x^i,y^i) that are labelled as either $[-1, 1]$ faulty or non-faulty messages where the numbers of erroneous and non-erroneous messages are balanced. We can use severity levels as log message template labels for those large-scale systems with this feature in their log data. The following preprocessing steps are conducted on the dataset with log message templates:

- Divide log message templates into tokens such as strings, variables, and punctuation.
- Remove all alphanumeric words, punctuation, stop words, variables that are not strings from log messages and use other NLP methods to clean text, such as lowercasing all texts, etc.

Phase II. Sentiment Lexicon Learning

We present the sentiment lexicon learning pseudo-code in Algorithm 5.1. Basically, the stochastic gradient descent logistic regression (SGDLR) technique obtains sentiment scores (a vector of weights) of lexicon elements by learning from the log message templates training set. Each weight w^j(used later as a sentiment score) is a real number ($\in \mathbb{R}$) and is linked with one of the log message features f^j. The weight w^j signifies how important the log message feature is to classifying a faulty log message from a non-faulty log message. Without loss of generality, we assume that a high positive weight indicates a message with a normal state or correctable error, and a very negative weight implies that the message is a failure or non-correctable error. This machine learning technique automatically computes the scores of lexicon items as follows:

Step 1. We employ the Term Frequency-Inverse Document Frequency (TF–IDF) representation technique [277] to extract n-gram features f^j from log

Algorithm 5.1: SGDLR to construct a sentiment lexicon for large-scale systems

```
    Input: $M$ log message templates $(x, y)$,Logistic regression $h()$,Loss function $L()$,
           learning rate $\eta$ , regularization parameter $\lambda$
    Output: Log features $f$ with their weights $W$
1    initial $W$, b, $\eta \leftarrow 0.01$, $\lambda$;
2   for each $(x) \in M$ do
        1. Tokenization
        2. Removal of alphanumeric words, variables, etc.
        3. Convert into TF-IDF representation format via Formula (5.1).
3   repeat
4       for $(x, y) \in M$ randomly do
            1. Compute $\bar{y} \leftarrow \frac{1}{1+e^{-\theta^T x}}$
            2. Compute the loss by Formula (5.5).
            3. Compute the gradient $g \leftarrow \eta \nabla L(h(x; \theta), y)$
            4. Update weights and bias $\theta_{t+1} \leftarrow \theta_t - \eta \nabla L(h(x; \theta), y)$
5   until SGD Converged;
6   return $(f, W)$
```

message templates x^i and convert these features to numerical vectors $\in R^{|L|}$, where lexicon L is a set of n-gram features. The TF–IDF value of each feature is calculated by multiplying two metrics: Term Frequency $tf(f^j, x^i)$ and Inverse Document Frequency $idf(m_{f^j}, M)$ as follows:

$$tfidf(f^j, x^i, M) = tf(f^j, x^i) \times idf(m_{f^j}, M) \tag{5.1}$$

In TF-IDF, the TF part measures how frequently a feature f^j occurs in a log message template x^i and is defined as follows

$$tf_{(f^j, x^i)} = \frac{n_{f^j, x^i}}{\sum_k n_{f^k, x^i}} \tag{5.2}$$

where n_{f^j, x^i} is the total number of feature f^j occurrences in a log message template x^i divided by the total number of features $\sum_k n_{f^k, x^i}$ in that message template. The IDF part, $idf(f^j, M)$, measures how important a feature f^j is in all message templates M by taking the logarithm of the ratio of the total number of log message templates M to the total number of log message templates $m_{f^j} \leq M$ containing the feature f^j plus 1, to prevent dividing by zero, as follows:

$$idf(m_{fj}, M) = \log\left(\frac{M}{m_{fj}+1}\right) \tag{5.3}$$

We refer the readers to [277] for more details.

Step 2. We use SGDLR to train our model on log message features to extract the dense weights vector W as follows:

i. Compute the estimated class $\bar{y} = \sigma(w.x + b)$ for each log message template via the following logistic regression function:

$$h(x) = \frac{1}{1 + e^{-}\theta^{T}x} \tag{5.4}$$

where θ includes two types of parameters: features' **weights** W and **bias** b (we neglect this parameter).

ii. Then, we use the cross-entropy function and L1 regularization (Formula (5.5)) to compute the loss $L(\bar{y}, y)$ in order to measure how close $\bar{y}$ is to the actual label y.

$$L(\bar{y}, y) = -[ylog\bar{y} + (1-y)log(1-\bar{y})] + \lambda|w| \tag{5.5}$$

The weights W of log message features and bias b are learned from labelled log messages training set through a loss function that must be minimized to make $\bar{y}$ for each log message as close as possible to the actual output y. L1 regularization (i.e., Lasso Regression) $\lambda|w|$, where parameter $\lambda > 0$, is added to the cost function to prevent the overfitting problem and improving model generalization by penalizing weights.

iii. Minimize the loss function in our model via the stochastic gradient descent technique (Formula (5.6)) to obtain the optimal weights W of log message features.

$$\theta_{t+1} = \theta_t - \eta\nabla L(f(x;\theta), y) \tag{5.6}$$

Stochastic gradient descent is a technique which is used to minimize the loss function by calculating its gradient after each mini-batch of log messages and updating the vector's parameters values θ (weights W and bias b). Since the loss function for logistic regression is convex, the SGD will reach the minimum of a lost function. The learning rate η is a hyper-parameter to adjust the model based on the calculated loss each time the model weights W are updated.

Step 3. The dense weight vector $\boldsymbol{W}$ is used as the sentiment scores with their associated log message features $\boldsymbol{f}$ as our lexicon items. For systematic observation, we normalize sentiment scores by dividing them by a uniform coefficient.

5.3.2 Sentiment Polarity-based Error Detection

In this section, we present a novel error message detection algorithm, i.e., log message labelling algorithm, exploiting the sentiment lexicon, which is constructed using stochastic gradient descent logistic regression explained in the previous Section (5.3.1). We present the approach in Algorithm 2.

Algorithm 5.2: Error detection algorithm based on sentiment lexicon

```
Input: Unlabelled messages logs $(x_1, \cdots, x_n)$, Sentiment Lexicon
       Items$(f_i, w_i)$,Absolute lexicon threshold $\mu = \mu$, detection threshold $\varphi = \varphi$
Output: $(x_{1\mathbf{posScore}}, \cdots, x_{n\mathbf{posScore}})$, $(x_{1\mathbf{negScore}}, \cdots, x_{n\mathbf{negScore}})$,
        $(x_{1\mathbf{SentiScore}}, \cdots, x_{n\mathbf{SentiScore}})$, Labels$(y_i, \cdots, y_n)$
1 Parsing messages logs $(x_1, \cdots, x_n)$
2 for $i = 1$ to $n$ do
3     $x_{i\mathbf{posScore}} \leftarrow \frac{\sum_i w_i}{\sum_i f_i}$ ;  $w_i > \mathbf{0}$
4     $x_{i\mathbf{negScore}} \leftarrow \frac{\sum_i w_i}{\sum_i f_i}$ ;  $w_i < \mathbf{0}$
5     $x_{i\mathbf{SentiScore}} = x_{i\mathbf{posScore}} + x_{i\mathbf{negScore}}$
6     if $x_{i\,SentiScore} < \varphi$ then
7         $y_i \leftarrow$ faulty;
8     else
9         $y_i \leftarrow$ non-faulty;
```

The algorithm includes three phases, which are described as follows.

Phase I: Log Parsing

This phase is similar to the log message preprocessing phase presented in Section 5.3.1, where the opensource toolkit LogAider [99] and NLP techniques are employed to preprocess and clean the system logs. For example, all duplicated events with spatial or temporal correlations are filtered out by LogAider.

Phase II: Log Message Sentiment Scores Assignment

In this phase, each log message x_i is associated with a sentimental polarity score using an assignment method similar to NLP's lexicon techniques (e.g., Vader [169]), in which the lexicon learned from our model is used to assign each log message a sentiment score made up of three different scores: positive, negative, and the overall log sentiment score, as follows:

$$x_{i\mathbf{posScore}} = \frac{\sum_i w_i}{\sum_i f_i} \ ; \quad w_i > \mathbf{0} \tag{5.7}$$

$$x_{i\mathbf{negScore}} = \frac{\sum_i w_i}{\sum_i f_i} \ ; \quad w_i < \mathbf{0} \tag{5.8}$$

$$x_{i\mathbf{SentiScore}} = x_{i\mathbf{posScore}} + x_{i\mathbf{negScore}} \tag{5.9}$$

In general, the faulty messages contain the system developers' negative sentiments expressing concern about unexpected system operations, unusual situations, serious problems, failed services, and corruption. Consequently, the $x_{i\mathbf{negScore}}$ is calculated by summing the negative valence scores $w_i < 0$ for each feature f_i of a log message that matches the sentiment lexicon features divided by their total number; the intensity of this score lies between 0 (neutral) and -1 (extremely negative). Moreover, log messages are generally embedded with more negative sentiments than positive or neutral ones, as the system developers tend to be more interested in abnormal systems' events. On the other hand, non-faulty messages include neutral sentiments indicating normal system behaviors and progress of system software (e.g., service started or stopped); positive sentiments indicate the system issues that are resolved (e.g., corrected errors, recovered failures). Therefore, the $x_{i\mathbf{posScore}}$ is calculated by summing the positive valence scores $w_i > 0$ for each feature f_i of a log message that matches the sentiment lexicon features divided by their total number; the intensity of this score lies between 0 (neutral) and +1 (extremely positive). The $x_{i\mathbf{SentiScore}}$ is the overall log polarity score that combines the $x_{i\mathbf{negScore}}$ and $x_{i\mathbf{posScore}}$. In other words, by summing the valence scores (i.e., whether it is positive or negative) for each feature f_i of a log message that matches the sentiment lexicon features divided by their total number. This score lies between -1 (extremely negative) and +1 (extremely positive). Similar to Vader, we used the Hutto normalization method (Formula 5.10) to ensure the overall sentiment score of the log message falls within the range of -1 to 1.

$$\frac{x_{i\textbf{SentiScore}}}{\sqrt{{x_{i\textbf{SentiScore}}}^2 + \alpha}} \tag{5.10}$$

α is a normalization parameter that is set to be 15.

Phase III: Detection Phase

Once system log messages are associated with sentiment polarity scores, the $x_{i\textbf{SentiScore}}$, detection threshold φ, and absolute lexicon threshold μ are used to detect whether these messages are faulty or non-faulty. A log message is classified as faulty when $x_{i\textbf{SentiScore}} < \varphi$ and as non-faulty otherwise. We can refine the φ threshold until we achieve an optimal value which results in a satisfied classification, however, in our analysis, we note that $\varphi = 0$ is a near-optimal setting. Furthermore, the absolute lexicon threshold μ can be adjusted until satisfactory classification accuracy with fewer lexicon features is achieved.

5.3.3 Erroneous Component Identification Based on Sentiment Polarity Scores

We can use our learned sentiment lexicon to calculate the sentiment polarity scores of the log messages for a certain time window t over some period (e.g., one hour time window over one day) for identifying the problematic components (e.g., nodes). The idea is that based on the sentiment scores, the system administrators can forecast which components may have erroneous behaviors, such that the jobs involved can be reassigned to other backup resources, and the problematic components would be temporally isolated until their problems are fixed. Specifically, the components' sentiment scores are anticipated to be neutral when they work as expected by logging informational messages or logging nothing. The negative scores are anticipated to associate with some components experiencing errors, especially when these abnormal states last for multiple consecutive time windows. However, when the components' issues have been resolved and they start to log the recovery and correction messages, their sentiment scores are expected to be increased or set to positive, indicating that they have been recovered well.

Our approach is composed of our sentiment lexicon that was learned in section 5.3.1, system's components $(C_i, ..., C_k)$, log messages $(x_1, ..., x_n)$, time window t, where the system developers define the start time t_S and end time t_E , and a detection sentiment score φ. The phase of erroneous component identification is similar to that of detecting error message logs; however, the components' associated sentiment scores are calculated within a certain time window specified by the systems' administrators. We present the pseudo-code of erroneous component identification algorithm based on a specific window time $[t_s, t_e]$ in Algorithm 5.3.

Algorithm 5.3: Erroneous component identification in large-scale system based on sentiment scores

```
Input: Components $(C_i, ..., C_k)$, Unlabelled messages logs $(x_1, \cdots, x_n)$, Sentiment
       Lexicon Items$(f_i, w_i)$, detection threshold $\varphi$ , Start time $t_S \leftarrow t_s$ , End time
       $t_E \leftarrow t_e$
Output: $C_{i\mathbf{posScore}}$, $C_{i\mathbf{negScore}}$, $C_{i\mathbf{SentiScore}}$, $C_{i\mathbf{State}}$
1  initial $\varphi \leftarrow \varphi$, $C_{i\mathbf{Logs}} \leftarrow$ " "
2  Parsing messages logs $(x_1, \cdots, x_n)$
3  for $i$ = 1 to $k$ do
4      for $j$ = 1 to $n$ do
5          $C_{i\mathbf{Logs}} \leftarrow$ Concatenate $(x_j, x_{j+1})$ ; $x_j \in C_i$ && $\mathbf{t_s} \geq t \leq \mathbf{t_e}$
6  for $i$ = 1 to $k$ do
7      $C_{i\mathbf{posScore}} \leftarrow \frac{\sum_i w_i}{\sum_i f_i}$ ;   $w_i > \mathbf{0}$
8      $C_{i\mathbf{negScore}} \leftarrow \frac{\sum_i w_i}{\sum_i f_i}$ ;   $w_i < \mathbf{0}$
9      $C_{i\mathbf{SentiScore}} \leftarrow C_{i\mathbf{posScore}} + C_{i\mathbf{negScore}}$
10     if $C_{i\,SentiScore} < \varphi$ then
11         $C_{i\mathbf{State}} \leftarrow$ Erroneous;
12     else
13         $C_{i\mathbf{State}} \leftarrow$ non-Erroneous;
```

Phase I: Component's Log Parsing

Unlike the log parsing phase in the error detection phase, the component's log messages should not be filtered out, in order to keep the spatial and temporal information. Moreover, in this phase, the system administrator are allowed to specify the time window in which the component logs are grouped together. Our model aims to assign sentiment polarity scores based on what each component logs about its health in a specific time.

Phase II: Component Sentiment Scores Assignment

In this phase, the system's components are associated with sentimental polarity scores using an assignment method similar to that used in the second phase of error detection presented previously. In particular, all negative features and positive features indicating error correction learned from our model are used to assign each component a sentiment score for a specified time window. It is composed of three scores: positive, negative, and overall sentiment score (denoted as $Ci_{\textbf{posScore}}$, $Ci_{\textbf{negScore}}$, and $Ci_{\textbf{sentiScore}}$, respectively), whose calculations are similar to those of the phase II in the error detection (see Formula (5.7), (5.8) and (5.9)) normalized by Hutto Formula (5.10).

Phase III: The Erroneous Component Identification

Once the system components are associated with sentiment polarity scores, the $Ci_{\textbf{SentiScore}}$ and detection threshold φ, are used to detect whether these components are erroneous or not. A component is classified as erroneous when it is attached with $Ci_{\textbf{SentiScore}} < \varphi$ for consecutive time windows and as non-erroneous otherwise. This model can be plugged into each system's components to generate sentiment scores over the time windows customized by administrators time windows. It works as an assistant tool that continuously alerts the administrator of erroneous components within negative scores and positive scores as the components' issues are corrected.

5.4 Experimental Evaluation

We perform the evaluation by automatically generating a sentiment lexicon from IBM Blue Gene systems' distinctive log message templates: BlueGene/L log templates(2007), BlueGene/P Intrepid log templates(2012), and BlueGene/Q MIRA log templates. Specifically, we carefully evaluate the viability of our error detection sentiment-based approach for three systems - Mira (4.2.1) (year 2017&2020), Ranger (4.2.2), and Lonestar4 (4.2.3), respectively. Our model is implemented using Python 3.8. , Pandas [24], Keras [7] with a TensorFlow [17] backend. We verified this test over an environment with the 2.2 GHz 6-Core Intel Core i7 processor and 16 GB 2400 MHz DDR4 memory. The operating system is MacOS Catalina version 10.15.x.. In the following, we first describe our evaluation indicators and then discuss the evaluation results.

5.4.1 Evaluation Metrics

Since our sentiment model not only detects erroneous logs but also accurately identifies non-erroneous logs, we utilize the weighted-average of (precision, recall, F1-score) and Matthew's correlation coefficient (MCC) to measure our sentiment model's performance. The precision, recall, F1-score are calculated for each category (faulty and non-faulty). So, for a given detection category *i*, we define: (i) True Positive$_{\mathrm{i}}$ (TP$_{\mathrm{i}}$) is the number of logs correctly detected as belonging to *i* category, (ii) False Positive$_{\mathrm{i}}$ (FP$_{\mathrm{i}}$) is the number of logs incorrectly detected as belonging to *i* category, (iii) False Negative$_{\mathrm{i}}$ (FN$_{\mathrm{i}}$) is the number of logs detected as not belonging to *i* category but, in fact, it does, and (iv) True Negative$_{\mathrm{i}}$ (TN$_{\mathrm{i}}$) is the number of logs correctly detected as not belonging to *i* category [241]. Based on TP$_{\mathrm{i}}$, FP$_{\mathrm{i}}$, FN$_{\mathrm{i}}$, and TN$_{\mathrm{i}}$, we then calculate precision$_{\mathrm{i}}$ and recall$_{\mathrm{i}}$ for each category *i* as follows:

$$Precision_i = TP_i/(TP_i + FP_i) \tag{5.11}$$

$$Recall_i = TP_i/(TP_i + FN_i) \tag{5.12}$$

and the F1-score$_{\mathrm{i}}$, which is a balanced harmonic average of recall and precision, as shown below:

$$F1\text{-}score_i = 2 \times \frac{recall_i \times precision_i}{recall_i + precision_i} \tag{5.13}$$

Lastly, the overall three scores (precision, recall, F1) are weighted by each category's support and averaged. The recall, precision and F1-scores are among the most popular statistical measures for a binary classification task. However, these measures can show overoptimistic results especially on imbalanced datasets. Matthew's correlation coefficient is a more reliable metric which produces a high score only if it obtains a good result in all the four confusion matrix categories (true positive, false positive, false negative and true negative), proportionally to both the size of positive and negative elements in the dataset. MCC is defined as [71]:

$$MCC = \frac{TP \times TN - FP \times FN}{\sqrt{(TP+FP)(TP+FN)(TN+FP)(TN+FN)}} \tag{5.14}$$

MCC returns a score between -1 to 1. A score of 1 represents a perfect detection. A score of 0 represents a random detection. A score of -1 represents total disagreement between detection and observation.

5.4.2 Evaluation of Error Detection

Learning Sentiment Lexicon for the IBM Blue Gene system

We use the SGDLR technique to automatically construct the sentiment lexicon based on the IBM Blue Gene system's RAS log message templates. We used $3k$ RAS log message templates as our dataset (detailed in Chapter 4, section (4.2.1)), whose labels are automatically inferred from the severity level field contained in a RAS log file. The RAS events are classified into two categories: faulty messages and non-faulty messages. The former include warnings, failures, and fatal levels, and the latter indicate informational messages to show the system software's progress or correction of errors.

We employ k-fold cross-validation with over-sampling methods to address the imbalance within RAS dataset. After the NLP text preprocessing phase, we adopt the Term Frequency-Inverse Document Frequency (TF–IDF) to transform terms (features) of RAS templates from text format to numerical vectors. The SGDLR algorithm with L1 regularization and default parameters are then employed to learn the sentiment lexicon items of the IBM Blue Gene systems by training RAS log template feature vector with their associated labels to obtain the dense weights vector W (sentiment scores for lexicon items). Our model obtains around ∼932 discriminative features associated with their sentiment scores. Figure 5.1 shows ∼60 lexicon items associated with their sentiment intensity scores learned automatically. It is clearly observed that the learned sentiment lexicon of Blue Gene systems consists of the system developers' negative sentiments that show the systems' issues and the positive sentiments indicating that the system problems have been corrected or the system components work as expected. Moreover, the Blue Gene lexicon contains more negative sentiments than positive and neutral sentiments, demonstrating that the abnormal system issues are generally paid more attentions in the logs. The Blue Gene lexicon contains ∼664 negative sentiments, whereas there are ∼268 positive ones.

Mira Error Detection Performance

After the Blue Gene lexicon's items are generated from the previous phase, we evaluate its efficacy by detecting error messages using the entire year 2017 of the Mira RAS logs. The year-2017 RAS log contains 16.5 million messages. We first filter out the duplicated messages by using the open source toolkit - LogAider [99] based on the spatial or temporal correlations. The total number of messages is thus significantly reduced from $16,772,894$ to $2,380,211$. Then, we preprocess the log messages' content by the NLP techniques. After that, we classify all events based on their severity attributes into two categories - faulty and non-faulty - in order to obtain the ground truth for evaluating our error detection method. Our error detection sentiment-based approach takes the filtered $2,380,211$ messages each of which is assigned a sentimental polarity score, and detects the faulty messages based on comparing the associated scores with detection threshold φ. Figure 5.2 presents the evaluation results about error detection accuracy based on the RAS log of year 2017 with different lexicon absolute threshold μ values(i.e., the number of sentiment lexicon items) with detection threshold $\varphi = 0$. Experiments show that our machine learning-based sentiment lexicon achieved excellent error detection accuracy for the RAS log data of the year 2017, 99%, 99%, 99% of recall, precision, and f1-score, respectively, at lexicon absolute threshold $\mu = 0$, 1, or 2, and detection threshold $\varphi = 0$.

In order to assess the effectiveness of our detection methodology under different conditions, we evaluate our sentiment lexicon approach with different values of μ and φ, which also aims at exploring the best threshold values with respect to high true positives and true negatives. We observe that a

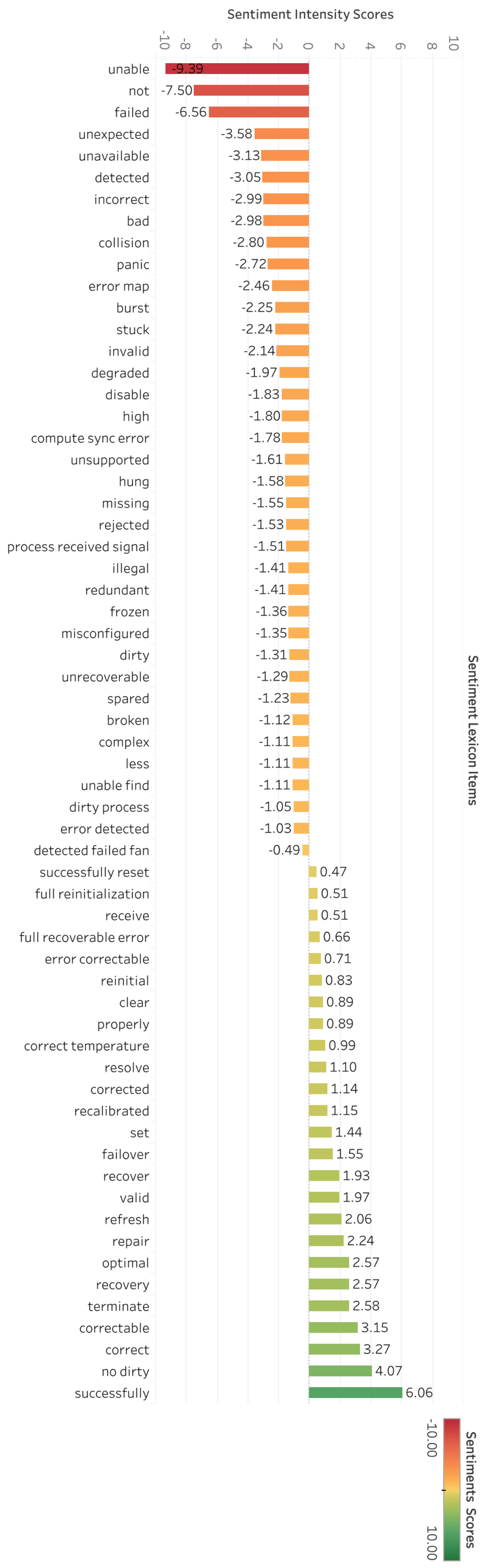


Figure 5.1: $\sim$ 60 of IBM Gene systems lexicon' items associated with their sentiment intensity scores

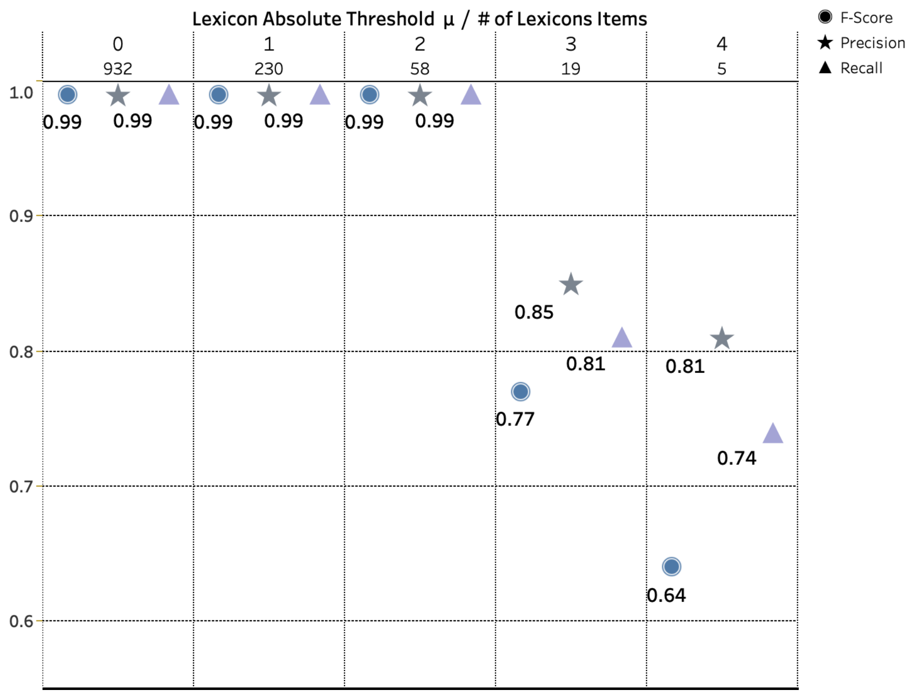


Figure 5.2: Detection with different lexicon absolute threshold μ, with $\varphi = 0$

small change to the lexicon absolute threshold μ (i.e., the number of sentiment lexicon items) and detection threshold φ may have a significant impact on the detection result. In general, the detection accuracy increases as the former threshold value decreases, meaning that using fewer lexicon items affects the detection accuracy. Moreover, the same detection accuracy is reached when only $\sim$ 58 sentiment items are used (i.e., μ=2), and when using all the 932 features of our learned lexicon. This result verifies the fact that developers often use only a few sentiment words to log system issues and operations, as opposed to similar NLP sentiment analysis tasks that contain a high number of sentiment words.

We achieve a good error detection for the latter threshold value as the φ is chosen within the range between 0 and -0.8; this means the developers use high-intensity sentiments to log the system issues (see Figure 5.3). Moreover, we observe that our model's misclassifications occur due to two reasons: the first reason is the developer's classification of some logs with the incorrect severity. For example, the severity level is INFO for the '`nd receiver link error`' message and is ERROR for the '`correctable ecc error threshold`' message in the RAS log. However, our model correctly classifies the former as faulty since this is what is

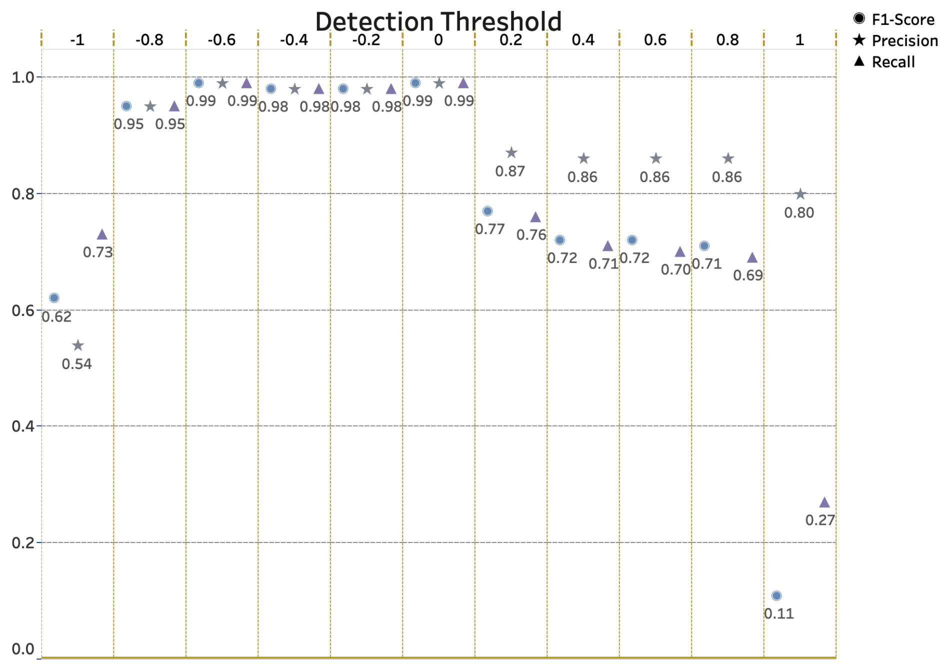


Figure 5.3: Mira error detection performance with different detection threshold φ, with $\mu = 0$

reflected through the feature **error**, and classifies the latter as non-faulty, since it shows the '`ecc error threshold`' is corrected. The second reason is that system developers tend to log some system events with unstructured text embedded with mixed negative and positive sentiments. For instance, the log message '`recoverable error message failed ecc parity error drill down error recoverable overable error cache parity error`' contains several negative and positive sentiments. Therefore, our model solves not only the problem of labeling the systems' logs with no severity levels but also fixes the misclassified severity levels within systems containing this feature.

Comparing Our Approach with ML/Deep Learning

New types of unlabelled logs are generated because the operators of large-scale systems (e.g., clusters, data centers) perpetually upgrade on the system components (software/hardware) and service to add new features, fix bugs, or enhance performance [340]. Furthermore, there are several large-scale systems (i.e., target systems) with non-labeled logs, such as Ranger (4.2.2) and Lonestar4 (4.2.3). Classifying the massive number of unlabeled logs (i.e., millions) of target systems manually into faulty and healthy is infeasible. Consequently, our approach can be used to detect potential faults/failures (classify logs to be faulty and non-faulty) automatically for the systems with non-labeled logs.

To demonstrate our approach's effectiveness and generality on cross-systems, we evaluate our sentiment lexicon's performance on logs of three large-scale systems to detect potential errors. We compare our solution with many state-of-the-art machine/deep learning classification techniques, including Random Forest (RT), Extreme Gradient Boosting (XGBoost), Multinomial Naive Bayes (Multinomial NB), K Nearest Neighbor (KNN), Long Short-Term Memory (LSTM), which have been commonly adopted by state-of-the-art approaches for error detection/prediction. For fairness, we train the five machine/deep learning models by using the same data we employed to extract our sentiment lexicon, and then evaluate those models' performance versus our sentiment lexicon model's performance on logs of three large-scale systems: the first six months of 2020 Mira (4.2.1) filtered RAS logs, and all distinctive log messages from Ranger (4.2.2) and Lonestar4 (4.2.3) (i.e., target systems).

Table 5.1: Scores (Recall, precision, f1-score, and MCC) of our lexicon and ML models on three systems(Mira, Ranger, and Lonestar 4)

	RAS Mira				Ranger			
Technique	Recall	Precision	F1-score	MCC Score	Recall	Precision	F1-score	MCC Score
Random Forest	96%	96%	96%	91%	55%	74%	51%	31%
XGBoost	96%	96%	96%	91%	50%	74%	44%	26%
Multinomial NB	96%	96%	96%	91%	45%	71%	36%	18%
KNN	96%	96%	96%	91%	50%	66%	46%	19%
LSTM	98%	98%	98%	96%	76%	80%	76%	55%
Our Sol.	99%	99%	99%	98%	94%	94%	94%	87%

	Lonestar4				Average of 3 Systems			
Technique	Recall	Precision	F1-score	MCC Score	Recall	Precision	F1-score	MCC Score
Random Forest	57%	78%	57%	35%	69%	83%	68%	52%
XGBoost	52%	78%	51%	32%	66%	83%	64%	50%
Multinomial NB	45%	75%	41%	23%	62%	81%	58%	44%
KNN	54%	71%	55%	25%	67%	78%	66%	45%
LSTM	80%	80%	79%	50%	85%	86%	84%	67%
Our Sol.	95%	95%	95%	87%	96%	96%	96%	91%

As presented in Table 5.1 and Figure 5.4, the results reveal that using our lexicon achieves an average MCC score and f1-score of 91% and 96% respectively in error detection in the three large systems' log messages, whereas the best machine/deep learning model (LSTM) obtains only 67% and 84% respectively. The other models RF, KNN, XGBoost, and Multinomial NB achieve an average MCC score of 52%, 45%, 50%, and 44%, and an average f1-score of 68%, 66%, 64%, and 58%, respectively.

As mentioned above, new types of unlabelled message logs in such HPC

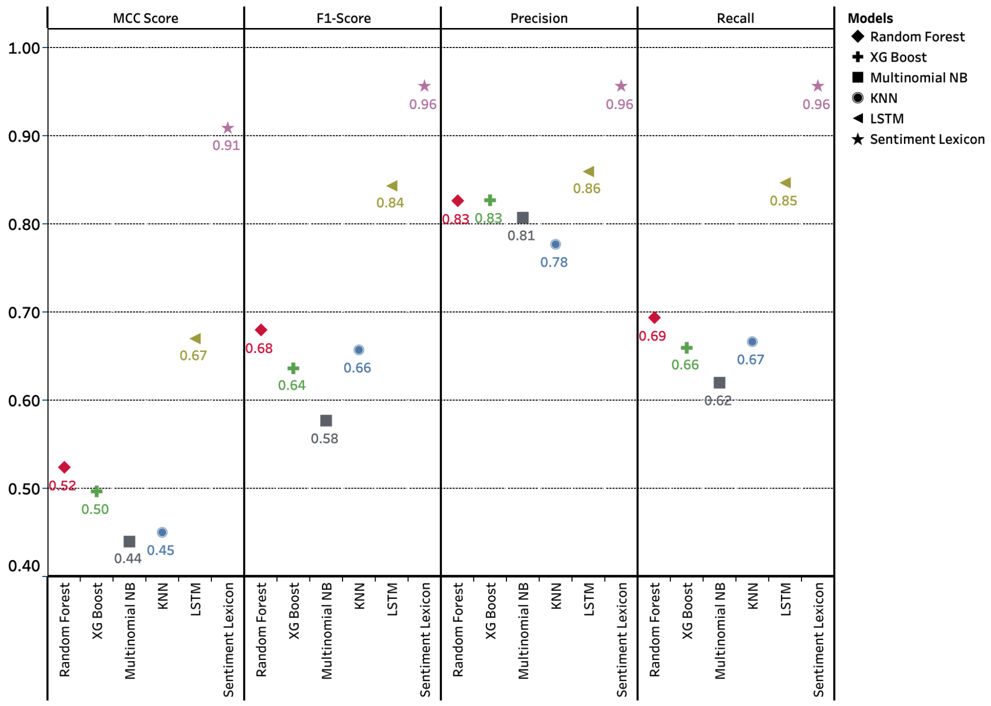


Figure 5.4: The average of detection performance of our lexicon and ML models on all three systems

systems were generated because of their upgraded or added components. It is non-trivial to manually classify a large number of new system log messages induced by the system upgrades or repairs. To cover this case, we evaluate our sentiment lexicon in classifying the latest RAS logs of IBM Blue Gene Mira 2020 (i.e., test dataset) and compare its performance with the other machine/deep learning techniques. The IBM Blue Gene systems' distinctive log message templates: BlueGene/L (2007), BlueGene/P (2012), and BlueGene/Q log templates are combined as our training dataset to learn the lexicon and train other five machine/deep learning techniques. The results (see Figure 5.5) show that our sentiment lexicon achieves the best MCC and F1 score among the six methods, achieving high MCC and F1 scores of 98% and 99% respectively. All other techniques achieve satisfying accuracy because the training runs on old RAS message templates and tests on Mira 2020 RAS logs' messages. The deep learning technique, LSTM, also achieves a high MCC score of 96% and F1 scores of 98%, respectively, and all other ML models achieve MCC scores (91%) and f-scores (96%). In general, our sentiment lexicon successfully identified the developers' sentiments from old generations Blue Gene supercomputers to use them for classifying new types of logs Mira 2020 RAS logs.

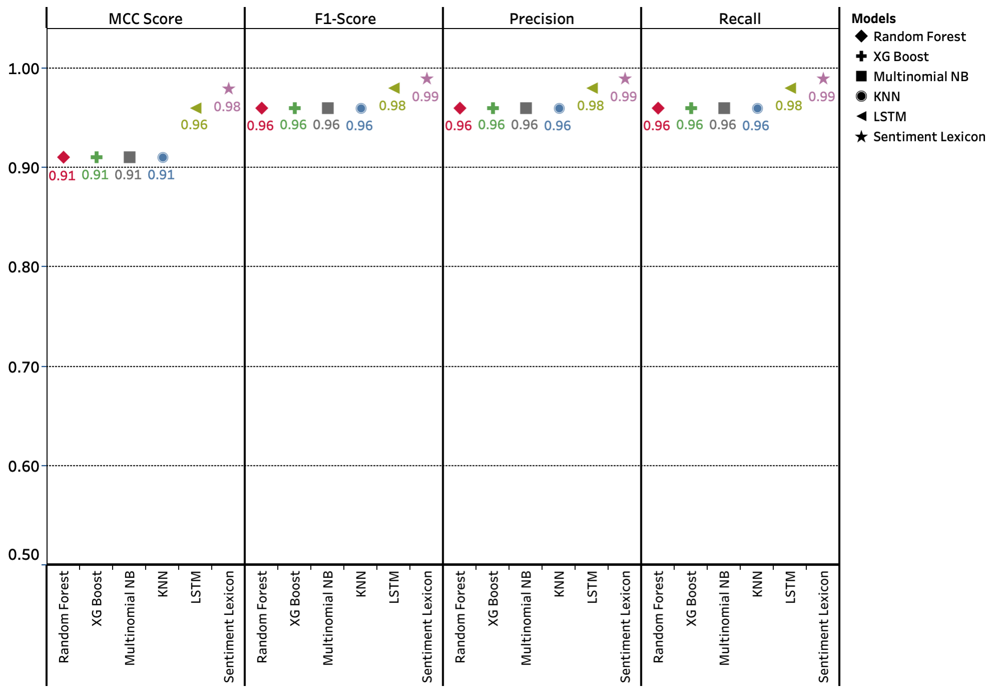


Figure 5.5: Detection performance of our lexicon/ML models on 2020 Mira RAS logs

As we stated before, our approach's primary goal is accurately identifying individual errors automatically for those HPC systems with a huge number of non-labeled logs. For example, the Ranger and Lonestar4 logged around 65 million and 12 million unlabelled messages in July 2011 and February 2013, respectively. To demonstrate the effectiveness of our solution in addressing this point, we utilize our sentiment lexicon features learned on labeled log messages from the source system (i.e., IBM Blue Gene) to detect errors in the target systems unlabelled log messages (i.e., Ranger and Lonestar4). As illustrated in Figure 5.6 and Figure 5.7, our lexicon achieves much higher scores in detecting error messages of Ranger and Lonestar4 logs than other solutions, even though they are different HPC systems developed by different companies with distinct logging methodologies. Our lexicon can detect the majority of errors accurately in Ranger and Lonestar4 unique logs. In absolute terms, the MCC scores reach up to 87% and 87%, the f-scores reach 94% and 95%, the recalls can get up to 94% and 95%, and the precisions achieve 94% and 95%, respectively. For other solutions including LSTM, RN, KNN, XGBoost, and Multinomial NB models, on Ranger Logs, the MCC scores are 55%, 31%, 19%, 26%, and 18%, respectively. The f1-scores are 76%, 51%, 46%, 44%, and 36%, respectively. On Lonestar4 Logs, those models get the MCC score only 50%, 35%, 25%, 32%, and 23%, respectively, and the f1-score only 79%, 57%, 55%, 51%, and 41%, respectively. The major limitations of detection accuracy on target systems for five machine/deep learning techniques are analyzed as follows: KNN does not work well because the log messages are imbalanced and contain many outliers, and LSTM, RF, and XGBoost are prone to overfitting, where Multinomial NB assumes the logs' features are independent, despite this not being the case.

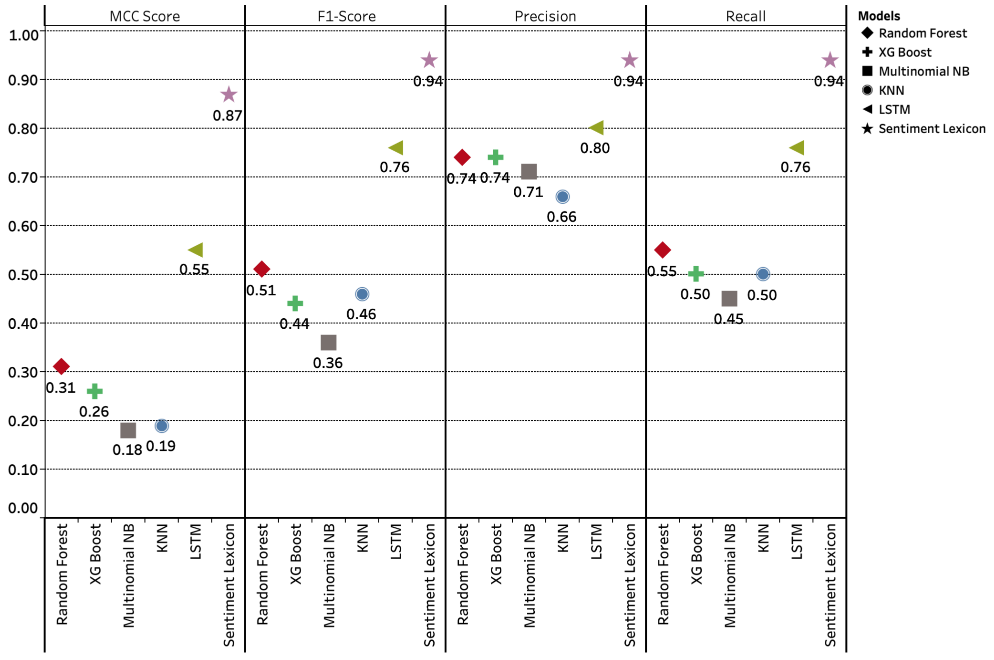


Figure 5.6: Detection performance of our lexicon and ML models on Ranger logs

These results demonstrate the reusable sentiment lexicon's benefits of deploying it as an unsupervised log analysis approach on different (target) systems. The key reason our solution outperforms other ML methods is that it successfully extracts developers' sentiment features hidden in labeled log messages from one source system (i.e., IBM Blue Gene) and transfers these features with their weights as lexicon items to detect errors in the target systems with unlabelled log messages (i.e., Ranger and Lonestar4. This verified the fact that developers of different systems really adopt similar sentiment features in their logging methodology. Thus, we can utilize a few system logs

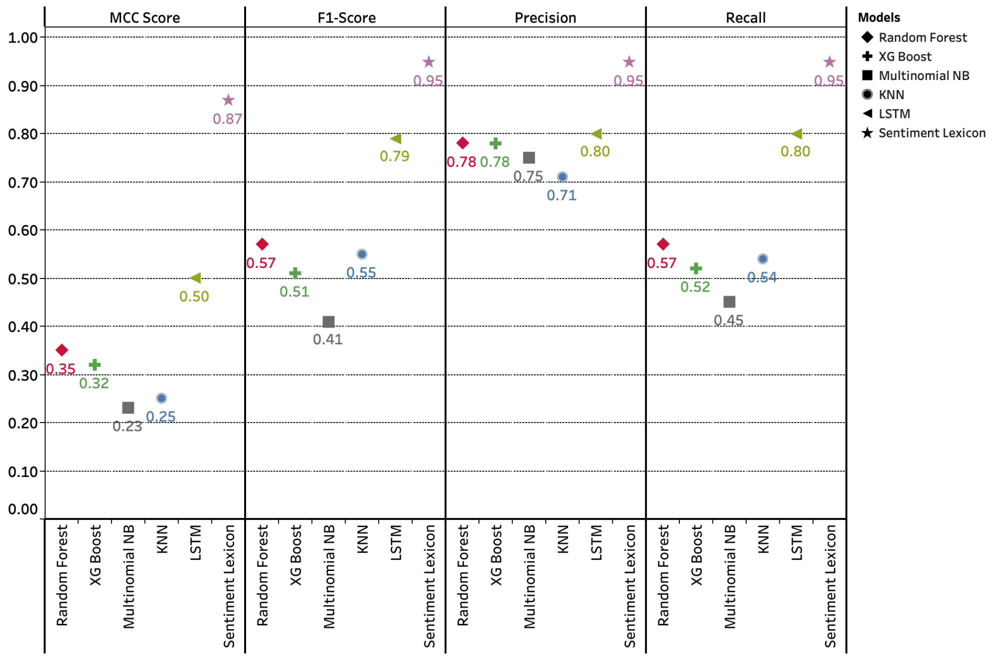


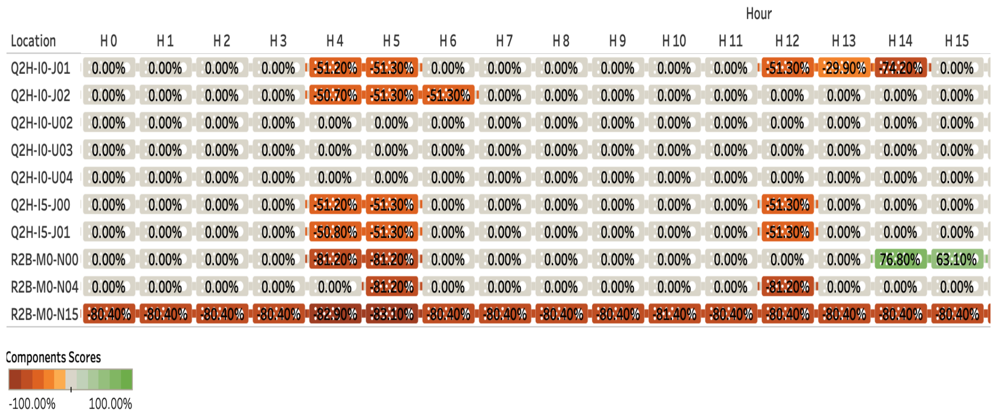
Figure 5.7: Detection performance of our lexicon and ML models on Lonestar4.

with severity levels to automatically extract their sentiment features and label the logs of other target systems with non-labeled logs. These promising results motivate us to collect more logs from different systems and generate a general sentiment lexicon using our technique to detect hardware and software issues in our future work.

5.4.3 Evaluation of Erroneous Component Identification

We evaluate our approach of identifying erroneous components such as compute nodes (denoted Rxx-Mx-Nxx), I/O nodes (denoted Qxx-Ix-Jxx) and link modules (denoted Qxx-Ix-Uxx), based on Mira's RAS logs. We present the evaluation results based on one-day period (27-March-2017) due to massive components involved. We employ our sentiment lexicon learned in Section 5.3.1 and detection sentiment score φ=0 to associate each component with 24 sentiment scores for the 24 hours, based on the hourly log messages regarding these components. Each score is composed of positive and negative scores, which are then summed up to produce the overall sentiment scores. Thus, each component is attached with a total of 72 sentiment scores (24 positives + 24 negatives + 24 overall scores).

Figure 5.8 demonstrates the states of some components in Mira. A gray color indicates 'working as expected by logging informational messages or logging nothing'; a red color indicates that the component is experiencing some abnormal events that affect its productivity; a green color indicates that the issues have been corrected, or silent errors have been fixed.

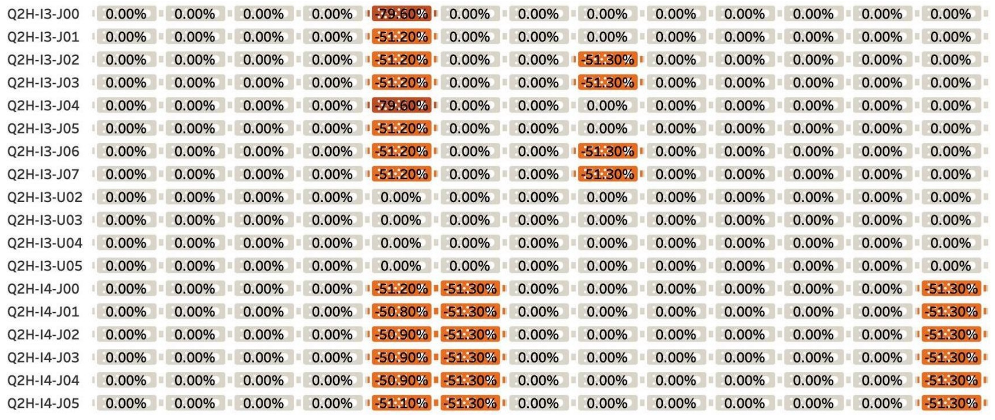


	Hour															
Location	H0	H1	H2	H3	H4	H5	H6	H7	H8	H9	H10	H11	H12	H13	H14	H15
Q2H-I0-J01	0.00%	0.00%	0.00%	0.00%	-51.20%	-51.30%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	-51.30%	-29.90%	-74.20%	0.00%
Q2H-I0-J02	0.00%	0.00%	0.00%	0.00%	-50.70%	-51.30%	-51.30%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%
Q2H-I0-U02	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%
Q2H-I0-U03	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%
Q2H-I0-U04	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%
Q2H-I5-J00	0.00%	0.00%	0.00%	0.00%	-51.20%	-51.30%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	-51.30%	0.00%	0.00%	0.00%
Q2H-I5-J01	0.00%	0.00%	0.00%	0.00%	-50.80%	-51.30%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	-51.30%	0.00%	0.00%	0.00%
R2B-M0-N00	0.00%	0.00%	0.00%	0.00%	-81.20%	-81.20%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	76.80%	63.10%
R2B-M0-N04	0.00%	0.00%	0.00%	0.00%	0.00%	-81.20%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	-81.20%	0.00%	0.00%	0.00%
R2B-M0-N15	-80.40%	-80.40%	-80.40%	-80.40%	-82.90%	-83.10%	-80.40%	-80.40%	-80.40%	-80.40%	-81.40%	-80.40%	-80.40%	-80.40%	-80.40%	-80.40%

Components Scores

-100.00% 100.00%

Figure 5.8: Illustration of The Erroneous Component Identification on Mira



Q2H-I3-J00	0.00%	0.00%	0.00%	0.00%	-79.60%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%
Q2H-I3-J01	0.00%	0.00%	0.00%	0.00%	-51.20%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%
Q2H-I3-J02	0.00%	0.00%	0.00%	0.00%	-51.20%	0.00%	0.00%	-51.30%	0.00%	0.00%	0.00%	0.00%	0.00%
Q2H-I3-J03	0.00%	0.00%	0.00%	0.00%	-51.20%	0.00%	0.00%	-51.30%	0.00%	0.00%	0.00%	0.00%	0.00%
Q2H-I3-J04	0.00%	0.00%	0.00%	0.00%	-79.60%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%
Q2H-I3-J05	0.00%	0.00%	0.00%	0.00%	-51.20%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%
Q2H-I3-J06	0.00%	0.00%	0.00%	0.00%	-51.20%	0.00%	0.00%	-51.30%	0.00%	0.00%	0.00%	0.00%	0.00%
Q2H-I3-J07	0.00%	0.00%	0.00%	0.00%	-51.20%	0.00%	0.00%	-51.30%	0.00%	0.00%	0.00%	0.00%	0.00%
Q2H-I3-U02	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%
Q2H-I3-U03	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%
Q2H-I3-U04	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%
Q2H-I3-U05	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%
Q2H-I4-J00	0.00%	0.00%	0.00%	0.00%	-51.20%	-51.30%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	-51.30%
Q2H-I4-J01	0.00%	0.00%	0.00%	0.00%	-50.80%	-51.30%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	-51.30%
Q2H-I4-J02	0.00%	0.00%	0.00%	0.00%	-50.90%	-51.30%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	-51.30%
Q2H-I4-J03	0.00%	0.00%	0.00%	0.00%	-50.90%	-51.30%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	-51.30%
Q2H-I4-J04	0.00%	0.00%	0.00%	0.00%	-50.90%	-51.30%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	-51.30%
Q2H-I4-J05	0.00%	0.00%	0.00%	0.00%	-51.10%	-51.30%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	-51.30%

Figure 5.9: Mira Erroneous Component Identification (Q2H-I3-J00∼Q2H-I4-J05)

We observe that majority of Mira components experienced no issues (e.g., Q2H-I0-U02, Q2H-I0-U03, and Q2H-I0-U04). Negative sentiment scores were attached to a few components (e.g., Q2H-I0-J01, Q2H-I0-J02, Q2H-I5u-J00, Q2H-I5-J01, and R2B-M0-N00) for consecutive hours, which validates the high accuracy of our algorithm because each of them crashed due to one or two fatal events according to the RAS log. Moreover, some components (e.g., the node R2B-M0-N15) exhibit high negative scores for long consecutive hours since it was suffering from recurrent abnormal events. Some components such as R2B-M0-N00, exhibit positive sentiment scores in later hours (i.e., the 14th and 15th), as some correction events were triggered. Our technique can assist the Mira system's administrator to isolate these faulty components until the problems are fixed, since there are 152 fatal events occurring in that day, and our model highlights all the components that triggered them.

Furthermore, as shown in Figure 5.9 from sentiment scores attached to Q2H-I4-J00 $\sim$ Q2H-I4-J05, we first observe that components are associated with similar sentiment scores for a consecutive or recurrent time windows. This can be explained by the fact that components generate the same logs over those time windows to convey that they are still facing the same issues. Second, we also observe that neighboring components are assigned with similar sentiments scores within similar time windows, indicating that large-scale system components that exhibit similar behaviors generate similar logs. Thus, one important observation is that the issues encountered by similar components may result in the same sentiments, and thus similar sentiment scores, enabling our technique to be deployed by Mira's administrators to detect faulty components.

Summary

In this chapter, we leverage the inherent meaning behind the log messages and propose a novel sentiment analysis-based approach for the error detection in large-scale systems, by automatically mining the sentiments in the log messages. Our contributions are four-fold. (i) We develop a machine learning (ML) based approach to automatically build a sentiment lexicon, based on the system log message templates. (ii) Using the sentiment lexicon, we develop an algorithm to detect system errors. (iii) We develop an algorithm to identify the nodes and components with erroneous behaviors, based on sentiment polarity scores. (iv) We evaluate our solution vs. other state-of-the-art machine/deep learning algorithms based on three representative supercomputers' system logs. Experiments show that our error detection algorithm can identify error messages with an average MCC score and f-score of 91% and 96% respectively, while state of the art ML/deep learning model (LSTM) obtains only 67% and 84%.

Chapter 6

Clairvoyant: A Log-Based Transformer-Decoder for Failure Prediction in Large-Scale Systems

Preface

The transformer-based pre-trained language models, such as GPT-2/3, have revolutionized the representations and transfer learning of the NLP downstream tasks; however, their applications to areas such as failure prediction in HPC systems are unknown.

System failures are expected to be frequent in the exascale era such as Petascale systems. The health of such systems is usually determined from challenging analysis of large amounts of unstructured & redundant log data. In this chapter, we leverage log data and propose *Clairvoyant*, a novel self-supervised (i.e., no labels needed) model to *predict node failures in HPC systems* based on a modern deep learning approach called transformer-decoder and the self-attention mechanism. Clairvoyant predicts node failures by (i) predicting a sequence of log events and then (ii) identifying if a failure is a part of that sequence. We carefully evaluate Clairvoyant and another state-of-the-art failure prediction approach – Desh, based on two real-world system log datasets. Our experiments show that Clairvoyant outperforms Desh with faster training and prediction time.

6.1 Introduction

Failures[1] in HPC systems can occur as a result of the scale and design complexity of the systems or due to faults occurring elsewhere in the system. Such failures typically lead to a significant computational overhead which, in turn, may have severe impact of system throughput.

In HPC systems, popular proactive failure management techniques are used such as task migration and checkpointing/restart. However, both techniques are expensive procedures and need to be used only when required, e.g., the computational overhead associated with these techniques may be exacerbated if they are wrongly triggered due to wrong failure prediction. Thus, it is important to develop efficient failure prediction techniques so that the overhead can be kept tractable. Effectiveness of current failure prediction approaches show a true positive rate of 50% in terms of actual failure identification and a false positive rate of less than 10%, meaning that the overhead of proactive techniques can be bounded [91, 102].

The SW of these HPC systems, such as OS and parallel file systems, typically generate a large volume of valuable log messages that are recorded in a centralised log file. These log messages typically capture the health states of every component (e.g., nodes). For example, a log message may state that the memory of a particular node has been corrupted. As such, these event logs are critical for system administrators to assess the state of the system.

Although log files are nontrivial for analysis (e.g., they are often unstructured, duplicated or even incomplete [77]), extensive research on failure-related analysis using HPC system logs has been undertaken such as detecting anomalies (e.g., [54], [161], [59]), diagnosing the root causes of failures (e.g., [77, 88, 93]), and detecting the errors that lead to system failures (e.g., [34, 148, 241, 346]).

While error detection is important at system runtime, not all errors will lead to system failure due to in-built recovery procedures such as the use of ECCs. As such, any premature triggering of an error recovery technique would likely introduce extra overhead. Accordingly, to mitigate the impact of system failures

[1]In the context of HPC systems, we will use node failures and system failures interchangeably.

on applications, it is critical to develop an efficient failure prediction mechanism alongside proactive failure management techniques [126]. Unfortunately, the failure prediction tools that determine when proactive failure management techniques should be activated is still insufficient [91]. This necessitates the development of failure prediction techniques that can flag impending failures ahead of time. Techniques that have been employed for failure prediction in HPC systems are, for example, support vector machines (SVM) [123], principal component analysis (PCA) [204], learning message patterns [315], Bayesian networks for hierarchical online failure prediction [266], and hidden semi-Markov models (HSMMs) [279]. Despite these contributions, these solutions have limited prediction accuracy or suffer from high computational overhead.

The proposed Long Short-term Memory (LSTM) and Bidirectional Long Short Term Memory (Bi-LSTM), used in [91] and [131] respectively, have been the most effective techniques for log-based failure prediction. However, they both suffer non-trivial weaknesses, e.g., due to recurrence learning, it is difficult to parallelize those approaches, leading to long training time. Another problem is the vanishing gradient problem, that causes the loss of earlier "memory" resulting in limited accuracy, i.e., long-range dependencies cannot be adequately captured.

As such, we develop **Clairvoyant**, a *self-supervised (no need for labels) transformer-decoder* based model to predict node failures in HPC systems by first predicting the future sequence of events (future health state) and then identifying if a failure is part of the sequence. Clairvoyant rectifies the limitations of LSTM implementations through the self-attention mechanism and parallelization. Denoting the predicted log sequence as $\mathbb{S}$ and a failure log event by $\mathcal{F}$, we then capture failure prediction of the node if $\mathcal{F} \in \mathbb{S}$, i.e., if a failure event appears in the predicted output log sequence. We run Clairvoyant on real-world datasets and the results obtained show that Clairvoyant *significantly* outperforms the state-of-the-art HPC failure predictor – *Desh* [91]. To the best of our knowledge, this chapter is the first attempt to leverage the self-attention and transformer-decoder techniques to predict node failures in HPC systems. However, different log-based studies have utilized self-attention with different transformers variants for anomaly detection and log parsing, such as Trine

[348], LAMA [144], LAnoBERT[207], NuLog[256], and [285].

We make the following contributions. (i) We develop *Clairvoyant*, a transformer-decoder based technique to predict component (node) failures in HPC systems. This is a generic model that can be applied to any other HPC systems or components since it is very common that the system failures are more or less correlated to the error messages. (ii) We evaluate the efficiency of Clairvoyant and Desh using two real-world logs from the Ranger supercomputer. The log data used in our experiments are very good representatives of large-scale HPC systems for failure analysis, as they are unlabelled, unstructured, and more complex than other HPC system logs. (iii) Our results show that Clairvoyant significantly outperforms Desh both in prediction accuracy and in training and prediction time.

Chapter structure: In Section 6.2, we present the problem formulation. Section 6.3 presents the methodology behind Clairvoyant, and we present the metrics used for performance evaluation in Section 6.4. We highlight the cluster system and its log datasets in Section 6.5, and we discuss the evaluation results in Section 6.6. A summary is provided at the end of this chapter.

6.2 Problem Formulation

Challenges in log-based failure prediction: Informally, our approach for log-based failure prediction is as follows: given a sequence of log events, predict an incoming log sequence and identify if a failure log event is in the predicted sequence. However, there are two critical challenges, which are: (i) The instant at which the failure log event appears in the predicted sequence should neither be too soon nor too late as the failure management mechanism may be triggered at the wrong time and (ii) The component (i.e, node) that is going to fail needs to be clearly identified so that failure management mechanism is triggered at right “location”.

We denote by $\mathcal{L}^r$, the set of log sequences of length at most r. Consider two sets: $\mathcal{L}^m$ and $\mathcal{L}^k, k \leq m$. The individuals in set $\mathcal{L}^k$ are called the possible extensions of the individuals in $\mathcal{L}^m$. Each individual in $\mathcal{L}^m$ can be assigned an output from $\mathcal{L}^k$, i.e., for each $s_i \in \mathcal{L}^m$, let $e_i \in \mathcal{L}^k$ be the true outcome to be predicted (i.e., the true log sequence that follows s_i). We model a (possibly randomised) predictor by a mapping $\mathcal{M} : \mathcal{L}^m \to \mathcal{L}^k$ such that $\mathcal{M}(s_i)$ is the predicted log sequence to follow s_i, i.e., $s_i \cdot \mathcal{M}(s_i)$ is a (future) predicted log sequence of length $(k + m)$, i.e., $s_i \cdot \mathcal{M}(s_i) \in \mathcal{L}^{m+k}$.

The two problems we address in this chapter can be formulated as follows:

Definition 6.2.1 (Log Prediction). Given a log sequence $s_i \in \mathcal{L}^m$, obtain a predictor $\mathcal{M}$ such that *arg min*$_{\mathcal{M}}$ $\mathcal{D}(s_i \cdot \mathcal{M}(s_i), s_i \cdot e_i)$, where $\mathcal{D} : \mathcal{L}^{m+k} \times \mathcal{L}^{m+k} \to \mathbb{R}$ represents a distance metric for log sequences of length $(m + k)$ and where $(\cdot)$ represents sequence concatenation.

In this case, we say that $\mathcal{M}$ *correctly extends* s_i if the distance is 0. Otherwise, we say that $\mathcal{M}$ *approximately extends* s_i. $\mathcal{D}$ is a distance metric on log sequences such that the distance is 0 when the logs are identical.

Definition 6.2.2 (Failure Prediction). Given a log sequence $s_i \in \mathcal{L}^m$, its extension $e_i \in \mathcal{L}^k$ and a predictor $\mathcal{M}$ that approximately extends s_i, we say that $\mathcal{M}$ accurately solves the failure prediction iff $\mathcal{F} \in e_i \Leftrightarrow \mathcal{F} \in \mathcal{M}(s_i)$. We say that $\mathcal{M}$ approximately solves the failure prediction problem if $\mathcal{F} \in \mathcal{M}(s_i) \Rightarrow \mathcal{F} \in e_i$.

It is interesting to observe that the (predicted) failure lead time is exact when $\mathcal{D}(s_i \cdot \mathcal{M}(s_i), s_i \cdot e_i)$ is 0, i.e., the failure event log does neither appear too soon nor too late. It is also worth noting that $\mathcal{D}$ will have a small value when the failure event appears at 'roughly' the correct time, i.e., at the correct place in the sequence. As such, developing an efficient mapping will help towards addressing the first challenge explained above.

So, for each node $C_j \in C$, given an input of a (previous) sequence of m log events $E_1, \ldots, E_m$ logged as node C_j's current health state, the aim of our transformer-decoder based model is to predict the sequence $E_{m+1}, \ldots, E_n$ future log events (i.e., the extension of $E_1, \ldots, E_m$), including failure events. The failure prediction is repeated for every node in $C_j \in C$ and the identity of the node that is predicted to fail will be known, thereby addressing the second challenge mentioned above regarding the failure location. The model calculates and predicts an upcoming log event probability $P(E_{(m+1):n})$ as follows:

$$P(E_{(m+1):n}) = \prod_{i=m+1}^{n} P(E_i | E_1, ..., E_{i-1}) \tag{6.1}$$

6.3 Methodology for Clairvoyant

In this section, we first provide a high level overview of the proposed approach followed by a detailed description.

An Overview of the Proposed Approach Due to the serious limitations of existing techniques (e.g., LSTM based methods) and challenges of the failure prediction problem, novel and scalable approaches are needed. *Transformer* neural network has made tremendous progress, primarily in Natural Language Processing (NLP) tasks (e.g., text prediction) and tackled LSTM limitations, through the *self-attention mechanism* and *parallelization* processing.

These properties benefit log-based analysis in multiple ways: (i) The self-attention mechanism emphasizes the important part of the input data and fades out the rest. Focusing on log-based analysis where, by analogy, we consider an event log entry as a word and a sequence of log entries as a sentence; self-attention will help focus on the important event log entries while moving focus away from irrelevant events. (ii) The self-attention feature is amenable to *parallelization*, meaning that training and prediction time can be drastically reduced, compared to LSTM.

Driven by the self-attention and parallelization learning – the crux mechanisms of the transformers neural network [305], we develop a novel approach namely *Clairvoyant* based on the transformer-decoder variant [272] to predict HPC nodes' failures by first predicting the future sequence of events (future health state) for each node and then identifying if a failure is part of the sequence. Predicting a compute node's (or a component's) failures ahead is achieved through accurately predicting the forthcoming log events $E_{m+1}, ..., E_n$ based on the previous log events $E_1, \ldots, E_m$ by that node. Our proposed transformer-decoder-based technique can be deployed in real-time to assist the large-scale systems administrator as nodes' failure predictor. As shown in Figure 7.1[2], the proposed model is based on a transformer-decoder consisting of a stack of attention blocks, preceded by log message preprocessing and an input embedding, followed by a log events prediction. We can incorporate these steps as two main phases; log message preprocessing and log events (log sequence) learning&(prediction), which are described in detail as follows.

[2]For simplicity, Figure 1 shows the prediction phases for one node.

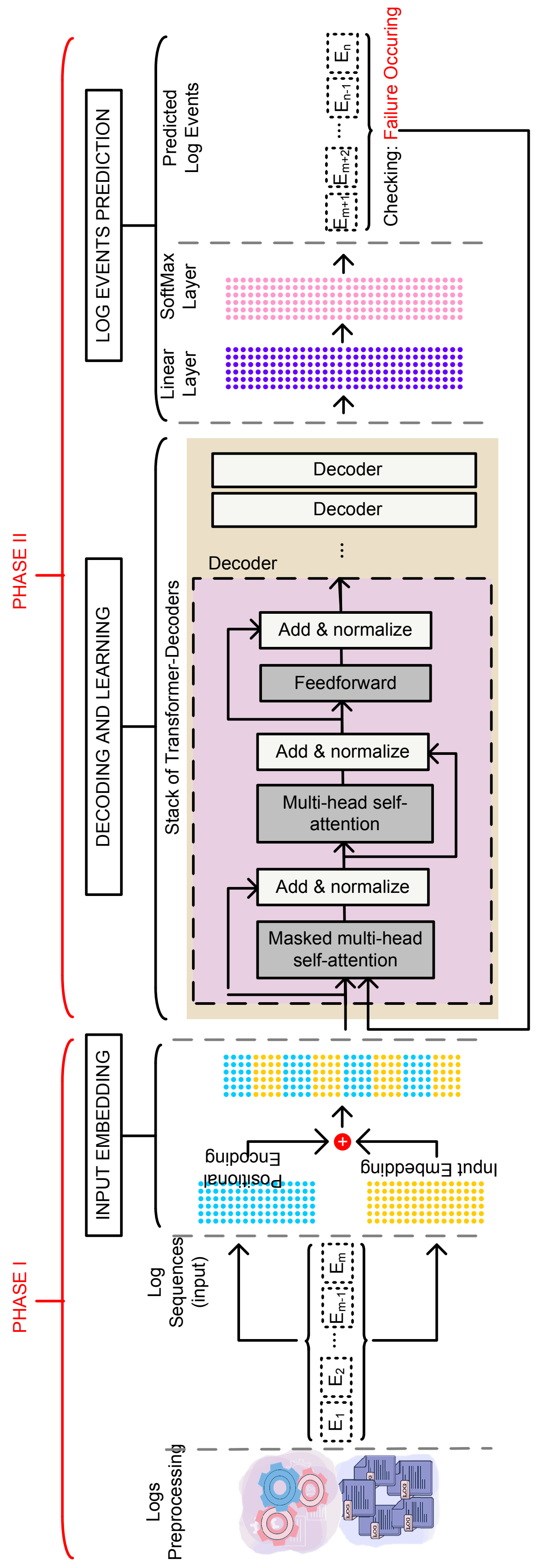


Figure 6.1: Failure and Health State Prediction Phases for Each Component (i.e., node)

6.3.1 Phase I. Log Message Preprocessing

In the first phase, standard NLP methods are used to clean textual log messages from all alphanumeric words, punctuation, stop words, variables that are not strings from log messages. After that, the duplicate messages are removed based on a time window as determined by an expert. Then, (unique) text log messages are mapped onto corresponding log event IDs based on the unique events (templates) from log message preprocessing. Next, each node's log event IDs are concatenated into sequences (log event sequences) sequentially based on their timestamps, where each sequence contains 1024 events at most. Hence, the number of sequences from each node can be calculated by dividing the number of log events generated by that node on 1024. Each node's sequences of log events is tokenized by breaking them up and transforming them to their associated indices (i.e., numbers). Those indices are generated by taking all events present in the log data and creating a vocabulary dictionary. The decoder blocks is fed by nodes' log sequences one after another. Besides, transformer-decoder can, in parallel, perform until 1024 log events within the input sequence, which is an advantage over the recurrent neural network (RNN) architectures such as Long Short Term Memory networks (LSTM). Also, the Byte Pair Encoding (BPE) technique is employed in transformer-decoder architecture to tokenize the input, allowing the encoding of any unusual tokens, which are the IDs of log events in our case. BPE is a compression NLP technique that repeatedly replaces the most common pair of adjacent bytes with a new byte that does not exist in the original data [124].

6.3.2 Phase II. Log Events Learning and Prediction

As stated before, in this chapter, we aim to predict the failures of HPC system components (nodes) and the entire health state through generating a sequence of forthcoming log events based on their preceding log events sequence. Thus, our proposed approach is based on the transformer-decoder deep neural networks designed for sequence processing. The transformer's core component's self-attention mechanism considerably improves the connectivity among the elements in long sequences. Accordingly, we employ transformer-decoder neural network, a stack of decoder attention blocks preceded by an input layer to embed the sequence of real-time log events logged by the HPC node, and followed by linear and softmax layers to predict failures by two steps: predicting the future sequence of events and then identifying if a failure is part of the sequence. More design details are described in the following text. We refer the readers to read [272] for detailed background of the *transformer variant* which we will use to build our model.

Step 1: Input Embedding

This step incorporates two types of encoding: log event embedding and log event's positional encoding, which are merged element-wise by dot-matrix multiplication:

Log Embedding: Each log event ID in input sequences is mapped into a vector of d_{model} dimension size, with continuous numeric values to represent that event learned through neural networks. By the end of the training, log event vectors' values represent the relation and dependency among these events.

Log Positional Encoding: Transformer avoids the recursion mechanism that is employed by RNNs, in order to enable parallel computation to minimize training time as well as the reduction in performance caused by lengthy dependencies. To this end, input embedding is associated with positional embedding to encode the order of the tokens (in our case, log events) and determine distances between the log events in the log sequences to the decoder blocks. The log event position i is encoded using sin and cosine periodic functions as follows:

$$\begin{cases} P_{(pos,2i)} = sin(\frac{pos}{10000^{\frac{2i}{d_{model}}}}) \\ P_{(pos,2i+1)} = cos(\frac{pos}{10000^{\frac{2i}{d_{model}}}}) \end{cases} \tag{6.2}$$

The even positions in the input vectors of log events in the sequence are calculated via the sin function, and the cos function is used for the calculation of odd positions. The benefit of utilizing sin and cos functions for positional encoding is periodicity. So, whether the model is learning on any length of tokens (events) in a sequence, the positions will always have the same range ([-1,1]) to project each position to a unique code during parallel computation.

The positional vectors are then added to their corresponding log events input embeddings. Based on transformer-decoder architecture, each log event embedding in the input sequence incorporates one positional encoding vector for each of the 1024 positions(pos) [272] in the input; d_{model} refers to the size dimension (which is 768 in our design), and i refers to the index within the vector of log event.

Positional encoding is added to the input embedding to construct the input matrix X before being passed to the decoder stack to provide information about the position of those corresponding inputs (log events). For an input log event E_i, its embedding x_i in the input matrix X is defined as:

$$x_i = W_{embedding} * E_i + P_{E_i}, i \in 0, ..., I - 1 \tag{6.3}$$

where **I** denotes the number of log events in the input sequence, $\boldsymbol{P_{E_i}}$ is positional encoding of $\boldsymbol{E_i}$, and the $\boldsymbol{W_{embedding}} \in \mathbb{R}^{\boldsymbol{E_{size}} \times \boldsymbol{V_{size}}}$ is the log event embedding matrix with embedding size $\boldsymbol{E_{size}}$ and the log events vocabulary size $\boldsymbol{V_{size}}$.

Step 2: Decoding and Learning

In the next phase, the input matrix X, which is log events embedding vectors, is passed forward to a stack of decoders (12 decoder blocks) one after the another forming the main part of the model. These decoders are identical in their architecture and functions in which each decoder block consists of a multi-headed masked self-attention layer, feed-forward neural network (FNN) layer, and some normalization layers. Each decoder has its own weights in both sublayers (self-attention and FNN). The following details show how the decoder layers work.

(1) **Masked Self-Attention:** The masked self-attention mechanism allows the model to associate each individual log event to its preceding log events in the input sequences; this leads to understanding and capturing the relation, dependencies, and order occurrence among the log events in the input log sequences. Therefore, all associated and relevant log events in the sequence that reveal the connection with a particular log event are identified. As the correlation between those log events preceding that log event as these events receive higher scores (given more attention). The self-attention is achieved

by creating three matrices for the decoder's input sequence X (in our case, a sequence of log events embedding). As stated in the previous step, the log events embeddings are combined into the input matrix X, where each row in X corresponds to a log event in the input sentence. A Query matrix (Q), a Key matrix (K), and a Value matrix (V) are created by multiplying X by three weight matrices, Query weight matrix (W_Q), a Key weight matrix (W_K), and a Value weight matrix (W_V), are trained during the training process. The matrices (W_Q), (W_K), and (W_V) have a smaller size dimension (64) than the log events embedding vectors (768) for better performance calculation of multiheaded attention(explained later). The input matrix X is passed through three linear layers W_Q, W_K, and W_V to produce the Query (Q), Key matrix (K), and Value matrix (V) matrices, respectively, where each row associated with a log event in the input sequence is defined by the following three equations:

$$\begin{cases} Q = W_Q \cdot X + b_Q \\ K = W_K \cdot X + b_K \\ V = W_V \cdot X + b_V \end{cases} \tag{6.4}$$

After the three matrices are created, several calculations are conducted to generate the masked self-attention Z, which can be depicted in the following formula:

$$\begin{aligned} Z &= MaskAttention(Q, K, V) \\ &= Softmax(mask(\frac{Q.K^T}{\sqrt{d_k}})V \end{aligned} \tag{6.5}$$

The masked self-attention is calculating the score for each log event against the preceding log events in the sequence by multiplying the dot product of that log event's query vector with its key $q_x \cdot k_x$, where q_x and k_x refer to the vectors of Q and K, respectively. As the correlation between the log event and its preceding log events increases, these events receive higher scores (given more attention). Then, those scores are divided by the square root of the dimension of the key vectors. The results are then passed to a softmax layer and they will be normalized all positive numbers with a sum being equal to 1. The obtained score from the softmax operation decides how much each individual event receives attention (focus) with respect to its current position in the input sequence. The relevant log events receive higher scores than other irrelevant ones. Next, the softmax scores are multiplying by each value vector v_x. This process keeps the relevant log events gaining high scores in the previous step and opting out unrelated log events because they are multiplied by tiny scores. Lastly, the output of the self-attention layer for that log event at that position is calculated by summing up the weighted value vector and send this vector along to the FNN layer. All these processes are performed in the form of matrix calculation in parallel for all sequence log events.

A "multi-headed" attention technique is employed (8 attention heads) to improve the self-attention layer's performance for two reasons. First, the ability of the transformer can be increased to extend attention to various positions. Second, the input embeddings can be projected into a varied representation subspace. Multiplying the input matrix X by the 8 multi-headed attention separate sets W_Q, W_K, and W_V weight matrices produces 8 sets of Query (Q), Key matrix (K), and Value matrix (V) matrices, respectively. Then, 8 different Z_i matrices are obtained. The FNN layer is expecting a single matrix to handle, thus the Z matrices are concatenated and multiplied with an additional weight matrix W_O to obtain the attention layer's output Z matrix that captures the information from all multi-heads.

(2) **Residual Connection and Normalization Layers:** Each transformer-decoder contains residual connection and normalization layers to make the training (learning) more effective. The layer normalization is calculated as:

$$Norm_{layer}(Z) = \gamma \frac{Z - \mu}{\sigma} + \beta \tag{6.6}$$

where γ and β are learnable parameters, μ and σ are the mean and standard deviation of the Z' vector's elements [272, 305].

(3) **Feed-Forward Neural Networks:** Each transformer-decoder also contains two-layer feed-forward networks with a ReLU activation function applied to each position separately and identically. The first layer is the input layer to receive the output of the preceding layer, hidden layers that capture the hidden correlations among those input log events. The second layer is the

output layer to pass forward what has been captured to the next step. Given a log sequence of vectors $h_1, ..., h_n$, the calculation of a position-wise FNN layer on a h_i is represented as:

$$FNN(h_i) = ReLU(h_i \cdot Z_{normalized} + b^1) \cdot W^2 + b^2 \tag{6.7}$$

where the $Z_{normalized}$, W^2, b^1, and b^2 are learnable parameters.

The other decoders work as the first decoder, and the output of each decoder is sent to the next decoder. The output of the final decoder is passed on to the linear and softmax layers.

Step 3: Log Events and Failure Prediction

The vector of float values returned by the last decoder in the stack is transformed into a vector (logits vector) via a dense linear layer whose size is equal to the number of unique log events (vocabulary size). For instance, in our case, if there are V unique log events in log dataset, this would result in a logits vector of V cells values where each cell corresponds to a unique log event score. Finally, those scores are converted into positive probabilities with a total sum of up to 1 through a softmax layer, and the index of logit cell with the highest probability is selected. Based on the model vocabulary, the log event associated with that index cell is predicted at this time step as the output (the predicted log event). A log sequence of desired length of events is predicted via a loop that begins by predicting the next event based on its preceding events, then appends it to the input log sequence, and continues to return the subsequent events. The outcome of this looping process is a prediction of the health state of the entire HPC component system. Consequently, a failure is predicted if it appears in the predicted log sequence (our main goal). This means that Clairvoyant predicts a node failure by first predicting its forthcoming sequence of events and then identifying if a failure is part of the sequence.

6.4 Evaluation Metrics

To demonstrate the viability of Clairvoyant for failure prediction of HPC systems, we examine two large datasets, obtained from two different logging mechanisms operational at different times, from the same HPC system, namely Ranger (4.2.2). In this section, we mainly introduce our evaluation metrics, and then we present the system, datasets, the evaluation results in the next sections.

Clairvoyant predicts not only future failures for HPC nodes but also the entire health state of every node. Thus, our model will be evaluated in two aspects. First, we evaluate the accuracy with which our model generates (i.e., predicts) log events of HPC nodes before the actual log events are generated. Second, we evaluate the accuracy with which our model predicts nodes' failures. For that purpose, two prediction-accuracy metrics supported by standard metrics including recall, precision, F1-score, Matthew's correlation coefficient (MCC), false-positive rate and false-negative rate, are utilized to evaluate our model.

Evaluating text generation (i.e., text prediction) in the NLP domain remains challenging because the generation task is open-ended. However, after a careful analysis of different text prediction evaluation metrics, we make use of the following metrics, namely *Bleu* and *Rouge*, which will be detailed below. The reason for choosing Bleu and Rouge is that they complement each other for the text prediction task as Bleu measures the *precision* of generated text (log events in our case) while Rouge measures *recall.* We detail the metrics below.

Bleu (Bilingual Evaluation Understudy [263])

Bleu is a precision-based metric calculated by comparing the degree of similarity between a text candidate to one or more text references (in our case, just one reference). It was initially designed as a translation evaluation metric, but later it was used to evaluate text generation tasks, more specifically, to compare a generated text sequence (candidate) against a reference sequence. A Bleu score ranges from 0 to 1, with 0 indicating a complete mismatch and 1 a perfect match, and 0.6 or above indicating a good result.

In this chapter, we use Bleu to measure how many log events predicted by our model (candidate) appear in (i.e., overlap with) the actual log events generated by the HPC system (reference).

The Bleu metric is defined as shown in Eq 6.8 [263]:

$$Bleu = BP \times e^{(\sum_{n=1}^{N} w_n \log p_n)} \tag{6.8}$$

where BP indicates the brevity penalty. $BP = 1$ when $r \leq c$ (Long candidates are not penalised, and only penalise short candidates) and $BP = e^{(1-\frac{r}{c})}$ when $r > c$, and where r indicates the length of the reference sequence (events generated by the HPC system) while c indicates the length of the candidate sequence (log events predicted by the model). N represents the length of the ngrams; $w_n = \frac{1}{N}$ indicates the positive weights, and p_n is the modified precision score as defined in Eq 6.9 [263].

$$p_n = \frac{\sum\limits_{C \in \{Candidates\}} \sum\limits_{n\text{-}gram \in C} Count_{clip}(n\text{-}gram)}{\sum\limits_{\acute{C} \in \{Candidates\}} \sum\limits_{n\text{-}gram \in \acute{C}} Count(n\text{-}gram)} \tag{6.9}$$

where $Count(ngram)$ is the number of ngrams for the candidate in the test set, and $Count_{clip}(ngram)$ is the number of clipped ngrams for the candidate log sequence.

Rouge (Recall-Oriented Understudy for Gisting Evaluation N-gram Co-Occurrence Statistics) [216])

Rouge is a recall-based metric calculated by counting the number of text reference ngrams (in our case, just one reference) that appear in the text candidate (i.e., predicted sequence). A Rouge score can range from 0 to 1, with 0 indicating a complete mismatch, 1 a perfect match, and 0.6 and above indicating a good result.

In our case, we use Rouge to measure the recall how many of the actual log events generated by the HPC system (reference) appear in (overlap with) log events predicted by our model (candidate), as defined in Eq 6.10 [216]:

$$Rouge = \frac{\sum\limits_{S \in \{Reference\}} \sum\limits_{(gram_n) \in S} Count_{match}(gram_n)}{\sum\limits_{S \in \{Reference\}} \sum\limits_{(gram_n) \in S} Count(gram_n)} \tag{6.10}$$

in which n indicates the number of ngrams, $Count(gram_n)$ is the count of ngrams that appear in the reference, and $Count_{match}(gram_n)$ refers to the maximum ngrams number occurring in both reference and candidate sets.

The standard metrics that are used are defined in equations (6.11) to (6.15). The symbols TP, FP, FN, and TN refer to True Positives (failures are predicted correctly), False Positives (failures are predicted incorrectly), False Negatives (failures are missed by our model) and True Negatives (non-failures correctly predicted by our model), respectively[3].

In addition, we compute the following metrics: (i) the F1-Score that indicates the overall failure prediction accuracy with regards to the weighted average between recall and precision, (ii) The Matthew's correlation coefficient (MCC) is an adequate metric as it only returns a high score if it performs well in all four confusion matrix categories (TP, FP, FN, and TN), proportionate to the quantity of positive and negative classes in the test dataset. The MCC score can range from -1 to 1 where a score of -1 indicates a complete discrepancy between the actual and predicted results, a 0 score represents a random prediction and a score of 1 indicates that the prediction is perfect.

$$(failure)Recall, Precision = \tfrac{TP}{TP+FN}, \tfrac{TP}{TP+FP} \tag{6.11}$$

$$(nonfailure)Recall, Precision = \tfrac{TN}{TN+FP}, \tfrac{TN}{TN+FN} \tag{6.12}$$

$$F1\ Score = 2\tfrac{Recall \cdot Precision}{Recall+Precision} \tag{6.13}$$

$$MCC = \frac{TP \times TN - FP \times FN}{\sqrt{(TP+FP)(TP+FN)(TN+FP)(TN+FN)}} \tag{6.14}$$

$$FPRate, FNRate = \tfrac{FP}{FP+TN}, \tfrac{FN}{TP+FN} \tag{6.15}$$

In comparison to classification tasks, the automatic evaluation of text prediction tasks is a significant challenge [278]. Even though Bleu and Rouge are two of the few main metrics for Natural Language Generation (NLG),

[3]Precision and recall are calculated for both classes (failures and non-failures) and highlighted in the experimental results section even our test datasets are balanced.

Table 6.1: Data Logs before and after the Preprocessing Phase

Log Data	Duration	From	To	# Before	# After
Syslogs	5 mon	Jan-11	May-11	43,639,722	2,346,780
RatLogs	6 mon	June-11	Nov-11	144,836,950	8,068,752

they have some drawbacks. However, and most importantly, most of these drawbacks do not apply to the problem of evaluating our model because only one reference and one candidate are used in our case. Nevertheless, for failure prediction experiments evaluation, Bleu and Rouge metrics are complemented using the standard metrics[4]. To use specific standard metrics, we use failure events and ignored all other log events from the candidate and reference.

6.5 Evaluation System, Datasets, and Soft Lockup Failure

In our experiments in this chapter, we adopt two real-world supercomputer system logs both generated by Ranger system (4.2.2), which detailed along with its two data logs in chapter 4 and have been widely used for failure analysis [34, 104, 149, 289].

The two different system logs generated based on two *different* logging frameworks, namely SysLogs (4.2.2) and Rationalized Logs (4.2.2). As shown in Table 6.1, our SysLogs dataset spans across five months (January to May 2011), while Rationalized Logs span six months (June to November 2011). We use both of them in our experiments and analysis.

Ranger Compute Node Failures (Soft Lockup Failures): Ranger administrators at TACC frequently encounter “compute node soft lockup” log events, indicating failures. A soft lockup is a state that causes the kernel of Linux OS to panic, be unresponsive, stuck, and loop endlessly, preventing other processes from being completed and eventually causing the nodes to crash. Soft lockup failures are recognized in log data by searching for the term *soft lockup*. Accordingly, the failures we aim to predict in this work are those soft lockups, which can be used to guide administrators in using mechanisms that

[4]The results show the validity of using Bleu and Rouge to evaluate the health state and failure prediction because both correlate highly with the results of standard evaluation metrics.

will reduce the number of applications from failing [73]. Several types of errors precede the failure of Ranger compute nodes (soft lock up), including Linux OS process errors, Lustre file-system errors, storage errors, network errors and software errors among others.

Figure 6.2 and Figure 6.3 show the recorded number of "compute node soft lockup" failure log entries for the Ranger logs per month. These failures are distributed over most of each month's dates; Figure 6.4 and Figure 6.5 show the dates and distribution of soft lockup failures that occurred in March and July, with a similar distribution pattern for other months. These months were chosen to showcase the distribution pattern, which also remains similar for other months. The distributions would likely exhibit a similar pattern if other months were chosen.

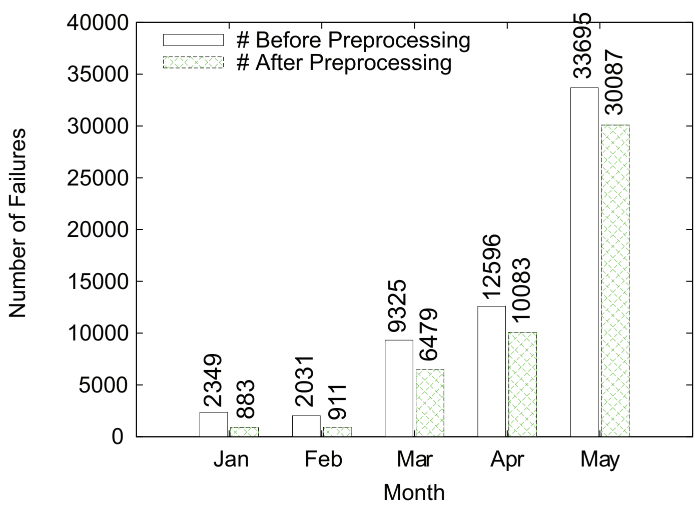


Figure 6.2: Monthly Compute Node Soft Lockup Failure Messages on SysLogs

6.6 Evaluation Results

To show the efficacy and applicability of our technique across log data, we evaluate the performance of our model on two **unlabeled** real-world log datasets with different logging frameworks, SysLogs and Rationalized Logs, to predict potential soft lockup failures. Moreover, we compare our method to a state-of-the-art deep learning prediction technique, Desh [91], the best in class, which employs LSTM to predict HPC node failures. There are good failure prediction approaches as well such as CNN-LSTM based and Bi-LSTM

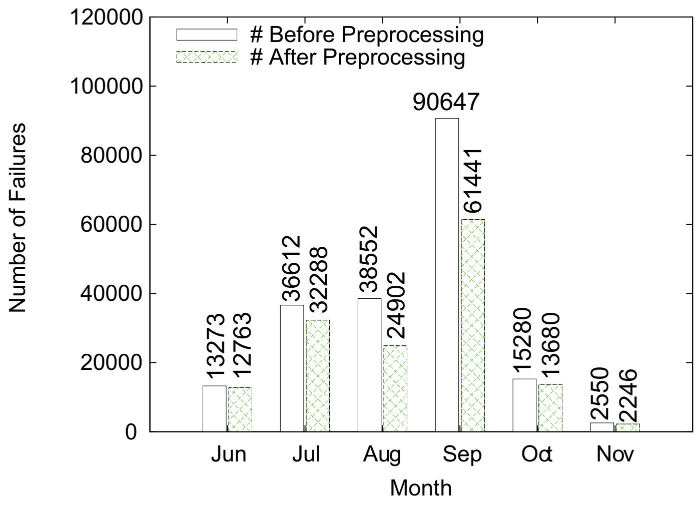


Figure 6.3: Monthly Compute Node Soft Lockup Failure Messages on Rationalized Logs

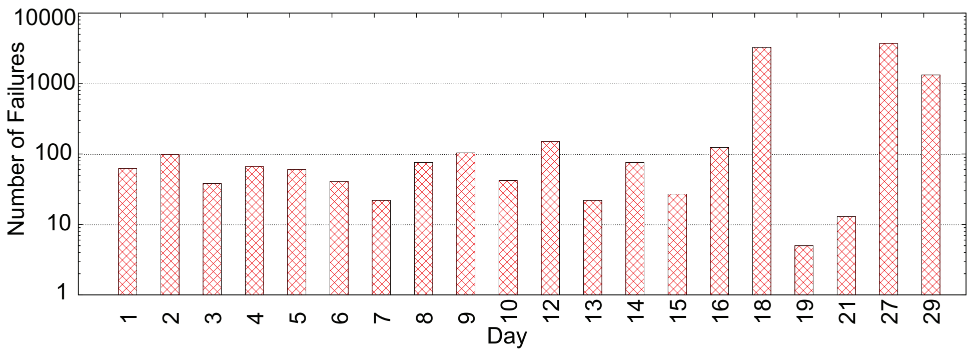


Figure 6.4: Daily Compute Node Soft Lockup Failure Messages in March

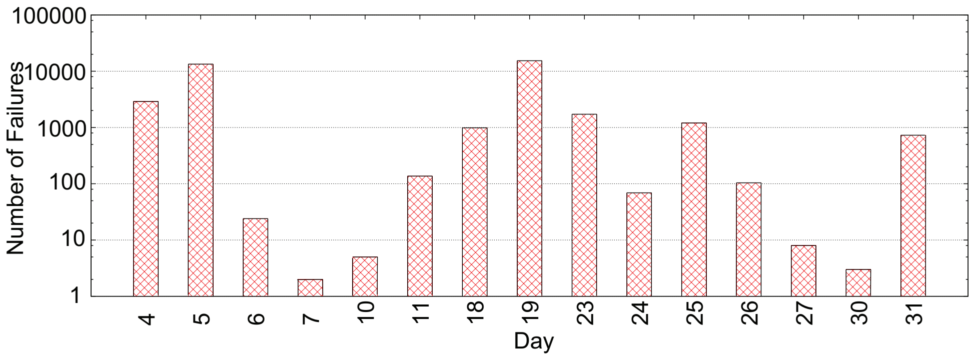


Figure 6.5: Daily Compute Node Soft Lockup Failure Messages in July

based approaches proposed in [225] and [131], respectively. However, both techniques require long training time with slower prediction because they combine two neural networks CNN+LSTM and Bi-LSTM, respectively, with similar accuracy to Desh. In addition, our model is a self-supervised learning that does not need labels, unlike the supervised learning based methods (such as SVM, Random Forest, KNN, etc.) that depends on the labels.

6.6.1 Log Data Preprocessing

We developed a log preprocessor to sort the log events based on their timestamps, clean raw log messages, and remove the duplicate ones based on the spatial and temporal correlations, as determined by an expert. After that, log messages are transformed into log sequences based on their associated nodes, as explained in the first phase of our methodology. Table 6.1 shows the quantities of both datasets' log messages before and after the preprocessing phase. 83087 log sequences are constructed from Syslogs, and 25272 log sequences are constructed from Rationalized Logs. Both logs are divided into training part and testing part. The training part accounts for 80% of the logs' data, while the testing part accounts for the remaining 20%.

6.6.2 Predicting Entire Health State of Ranger Performance Evaluation

Clairvoyant is implemented using Python 3.8., Pandas [24], Keras [7] with a TensorFlow [17] backend, PyTorch [13]. We verified the experiments over the environment of Google Colab [20] by leveraging NVIDIA Tesla V100 GPU, Intel Xeon 2.30 GHz processor, and 52 GB RAM. We build our technique using a neural network architecture similar to the GPT2 model, which has a stack of 12 transformer-decoders with 12 attention heads for each layer, 768 dimensional states to encode log events into their embeddings, 1024 feed-forward sizes, and maximum log sequence input length being set to 1024. Additionally, we implement the baseline (Desh) model as explained in [91]. The training is conducted for 10 epochs and a batch size of 16 for our model as well as the baseline model.

Training and Prediction Time Performance

Table 6.2 shows that the training time for learning using our model is drastically reduced 25.2× compared to Desh on average. Our approach requires only 1.64 and 0.71 hours in training on SysLogs and Rationalized Logs, respectively. By comparison, Desh requires at least 41 and 18 hours on both training sets, respectively. Reducing training time is critical because deploying the model online in real-time necessitates continuous training and fine-tuning parameters in case new forms of failure patterns are created. These new failure sequences can be created because the operators of large-scale systems (e.g., supercomputers and data centers) are frequently upgrading the system components (software and hardware) and services in order to add new features, repair faults, or improve performance. Also, new high-end scientific applications, such as scientific applications, which are executed on these high-performance computing (HPC) systems could create different failures patterns that the model has not learned before.

Table 6.2: Training Time Performance in Hours

	Clairvoyant		Desh	
	SysLogs	R. Logs	SysLogs	R. Logs
1 Epoch	0.16	0.07	4.17	1.82
Entire Training	1.64	0.71	41.74	18.23

Furthermore, our model predicts the forthcoming log sequence of events 15.4× faster than Desh during the testing. As illustrated in Figure 6.6, only 0.30−5.78 secs are needed to predict a log sequence chain of lengths 64 to 1024 respectively, whereas the Desh technique requires 3.36−98.00 secs, where the chain length indicates the length of the predicted log events. This can be explained by the transformer-decoder mechanism's parallelization capabilities and positional encoding, which takes substantially less training and prediction time than the RNN models such as LSTM, which lack parallel training and require sequential learning. Therefore, the prediction of upcoming events using our solution on the testing data is very suitable for the real-time failure prediction scenario that requires fast forecasting to trigger the appropriate proactive recovery actions and avoid costly failures.

Overall Learning and Log Events Prediction Performance

We employ Bleu and Rouge metrics to measure our model's overall accuracy in generating (predicting) the nodes' forthcoming log events (whether informational, errors, or failures) before the actual log events are generated, with respect to the entire system future health state. Bleu and Rouge measure the

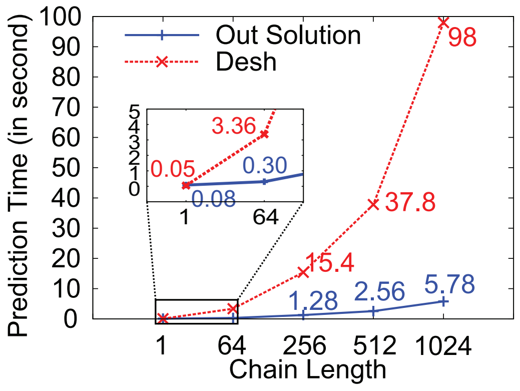


Figure 6.6: Chain Lengths(# Log Events) Prediction Time

degree of similarity (overlapping) between the candidate (predicted log events by our model and the baseline (Desh)) and the reference (log events generated by the Ranger system in realtime).

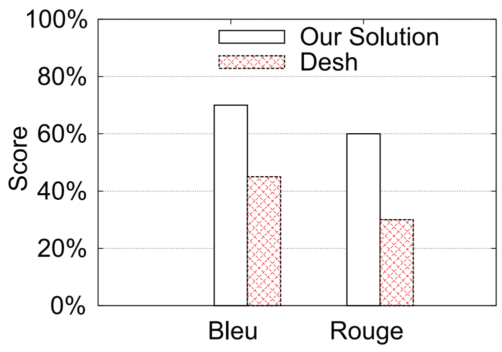


Figure 6.7: Bleu and Rouge for Entire Health State Prediction on Syslogs

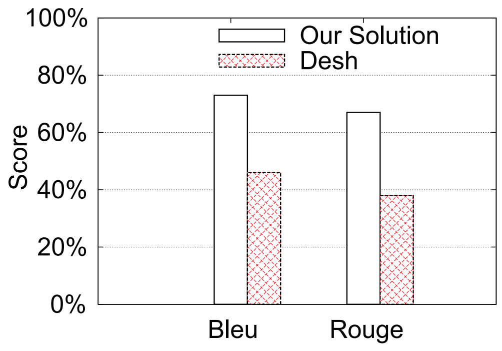


Figure 6.8: Bleu and Rouge for Entire Health State Prediction on Ratlogs

As can be seen in Figures 6.7 and 6.8, our transformer decoder-based approach achieves a Bleu score of 0.70 and 0.73 on SysLogs and Rationalized Logs, respectively, in predicting forthcoming log events. In contrast, the Desh baseline obtains only 0.45 and 0.46, respectively. This means that on average, 72% of log events predicted by our model (the candidate) appeared in the events generated by the HPC system (the reference), compared to just 45.5%by Desh.

Also, as illustrated in Figures 6.7 and 6.8, results demonstrate that our technique obtains Rouge scores of 0.60 and 0.67 on SysLogs and Rationalized Logs, respectively. Desh, however, obtains only 0.30 and 0.38, respectively. This means that on average, 63.5% of events generated by the HPC system appear in the log events predicted by our model, compared to Desh's 34%.

The Bleu and Rouge scores imply that it is difficult for Desh, an LSTM-based model, to capture long-range dependencies/correlations between events in long sequences due to a loss of memory for earlier events caused by the vanishing gradient problem. On the other hand, our technique successfully predicts the future log events sequence depending on the preceding lengthy log sequence (predicting the upcoming health state from previous& current health state). This is indicated by a high match (overlap with) between the forthcoming log events predicted by our model(candidate), and the events generated by the HPC system (reference). The key reason is that the masked self-attention mechanism, which is the crux of our model, efficiently identifies the log entries of important events while moving the focus away from irrelevant ones and capturing long-range dependencies/correlations between events in long sequences.

6.6.3 Node Failure Prediction Performance Evaluation

The main objective of this research is to predict the failures of nodes in an HPC system. As explained, our solution, Clairvoyant, predicts node failures by first predicting the future sequence of events for every node as evaluated above and then identifying if a failure is part of the sequence, as evaluated below using Bleu and Rouge, and the standard metrics mentioned in the Section 6.4. To this end, we calculate the values of the evaluation metrics by focusing on only failure prediction events, unlike the evaluation of the prediction of the entire health state, which involves multiple log events predicted by our model and the events generated by the HPC system.

Ranger often encountered compute node lockup failures; thus, our goal is to predict those soft lockup failures ahead of their occurrences, to trigger proactive

Table 6.3: Failure Prediction Performance Evaluation on Both Data Logs[5]

	SysLogs		Rationalized Logs	
	Our Sol	Desh	Our Sol	Desh
Bleu	0.89	0.56	0.91	0.59
Rouge	0.75	0.57	0.80	0.59
Failure Precision	0.97	0.76	0.99	0.86
Failure Recall	0.52	0.21	0.61	0.21
non-Failure Precision	0.67	0.54	0.72	0.55
non-Failure Recall	0.98	0.93	0.99	0.96
Overall Precision	0.82	0.65	0.86	0.70
Overall Recall	0.74	0.57	0.80	0.59
F1-Score	0.74	0.51	0.80	0.52
MCC Score	0.6	0.2	0.7	0.3
FP-Rate	0.02	0.07	0.01	0.04
FN-Rate	0.48	0.79	0.39	0.79

failure management procedures in the system. Accordingly, we demonstrate our approach's effectiveness by evaluating the performance of our technique and comparing the results with baseline results on two different logs as follows:

As presented in Table 6.3, Figure 6.9, and Figure 6.10, our model predicts upcoming failures with a Bleu score of 0.89 and 0.91 on SysLogs and Rationalized Logs, respectively. In comparison, the Desh baseline scores just 0.56 and 0.59. In other words, on average, 90.0% of Ranger failures predicted by our model (the candidate) appear in the events generated by the HPC system (the reference), compared to only 57.5% by Desh.

Moreover, results show that our technique obtains better recall accuracy. Specifically, Clairvoyant achieves a Rouge score of 0.75 and .80 on SysLogs and Rationalized Logs, respectively. Desh obtains only 0.57 and 0.59, respectively. This means that on average, 77.5% of failures generated by the Ranger appear in the log events predicted by our model, compared to Desh's 58%.

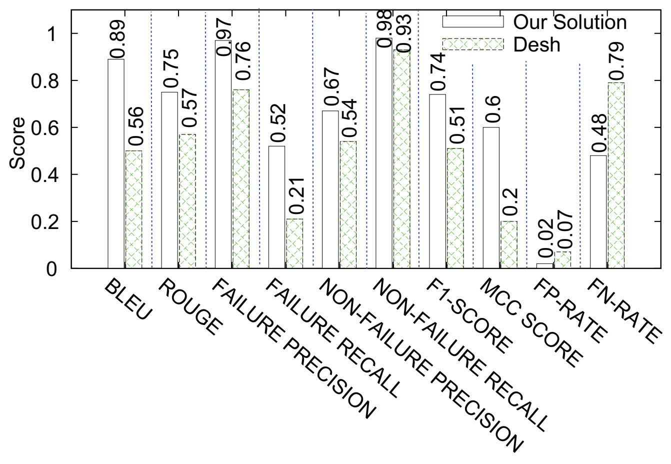


Figure 6.9: Failure Prediction Performance on SysLogs

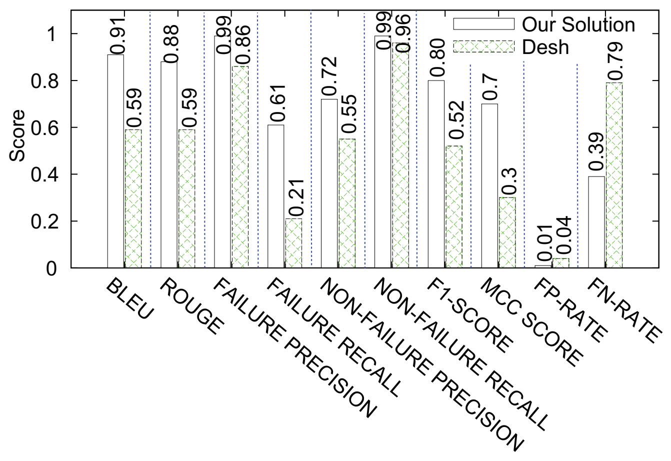


Figure 6.10: Failure Prediction Performance on RatLogs

As presented in Table 6.3, Figure 6.9, and Figure 6.10, the result of standard metrics shows that Clairvoyant predicts failures on SysLogs and Rationalized Logs with a high-precision score of 0.97 and 0.99, and the recalls can reach up to 0.52 and 0.61, respectively. In comparison, the Desh baseline achieves lower precision scores of 0.76 and 0.86 and lower recall scores of 0.21 and 0.21, respectively. Also, Clairvoyant predicts non-failure sequences correctly (benign sequences) on SysLogs and Rationalized Logs with a precision score of 0.67 and 0.72, and its recalls can reach up to 0.99 and 0.99, respectively. The Desh baseline, on the other hand, achieves lower precision scores (0.54 and 0.55) and also high recall scores (0.93 and 0.96), respectively.

We also check MCC, which is a reliable metric as it only returns a high score if it performs well in all four confusion matrix categories (TP, FP, FN, and TN), proportionate to the quantity of positive class (failure) and negative (non-failure) class in the test dataset. Moreover, we also utilize the weighted average of f1-score from the positive class (failure) and negative (non-failure) class for more accurate evaluation even our test set is balanced between the two classes. The results (see Table 6.3, Figure 6.9, and Figure 6.10) show that our model (Clairvoyant) achieves better prediction on SysLogs and Rationalized Logs with MCC scores of .6 and 0.7, and the f1-scores reach 0.74 and 0.80, respectively. In comparison, the Desh baseline achieves MCC scores of 0.2 and 0.3 and f1-scores of 0.51 and 0.52, respectively.

Moreover, false positive rate (FP-rate) and false negative rate (FN-rate) also demonstrates the substantial improvement of our model. The FP-rate shows that 7% and 4% of Desh alarms on SysLogs and Rationalized Logs are false alarms (which would cause incorrect recovery actions), respectively. On the other hand, our model only drives 2% and 1% false alarms, leading to rare incorrect trigger recovery actions. Also, based on FN-rate, Desh significantly missed real node failures on SysLogs and Rationalized Logs (both 79%), while Clairvoyant missed only 48% and 39%.

As follows, we give a detailed explanation why our model significantly advances Desh in node failure prediction. As stated before, different lengths of log sequences are observed between Ranger's node failures and their associated errors&faults (such as software& kernel OS process, file-system errors, memory&storage errors, and network errors) for each Ranger component (e.g., nodes). Those sequences contain numerous interleaved & irrelevant log events, making the failure prediction process more challenging. For example, some errors take many hours to trigger the associated failures, resulting in extended and lengthy log sequences (e.g., over 2000 events even after the preprocessing phase). Nevertheless, due to multi-head masked attention layers and the positional encoding technique, our transformer-decoder-based model outperforms the recurrent neural network baseline (Desh) in that it completely avoids recursion, processing log sentences as a whole and understanding associations between log events. In other words, our approach's effectiveness in identifying the relationship between Ranger soft lockup failures and their preceding inducing errors comes from masked attention neural networks, which is the main component of the transformer-decoder and positional encoding layer that is combined with log events embedding. Accordingly, our solution can successfully predict node failures before they occur based on evaluation scores. The baseline – Desh, however, achieved lower accuracy and slower prediction because it is an RNN-based model that requires recurring learning and sequential processing (log sequences processed event by event). Moreover, some log sequences are too long, and LSTM fails to capture the long relationship dependency range between failures and inducing error events as a result of the vanishing gradient problem, causing memory loss for earlier occurring events.

6.6.4 Node Failure Prediction Performance with Different Decoding Techniques

Transformer-decoder neural networks can be employed for text prediction with different decoding methods, including greedy search, beam search, basic sampling, top-K sampling, and top-P (Nucleus) sampling, and our model can work with each of them flexibly. This section examines the performance of two of these techniques on Rationalized Logs (similar results appeared with Syslogs) and how they affect the node failure prediction accuracy.

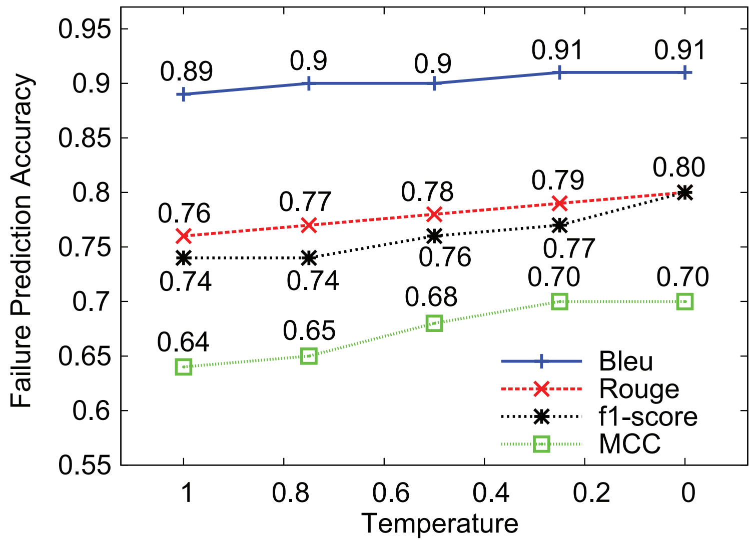


Figure 6.11: Failure Prediction with Different Temperature Scale based on Sampling Decoding

Greedy Search

In the greedy search decoding technique, the next log event is predicted as the log event with the highest probability, and the next log event is updated through the following Eq 6.16 at each time step t.

$$E_t = argmax_E P(E_t|E_{1:t-1}) \tag{6.16}$$

Using the greedy technique, our model achieves high scores of Bleu and Rouge, f1-score, MCC 0.91, 0.80, 0.80, and 0.7 respectively, in predicting future soft lockup failures in Ranger Rationalized Logs.

Sampling Decoding with Different Temperature Values

In NLP prediction task, the unpredictability of the predicted text (log events in our case) is controlled by a temperature (hyper-parameter), so we explored our model performance using basic sampling decoding with different temperature values. As shown in Figure 6.11, we observe that the failure prediction accuracy increases as the temperature value decreases, meaning that log events with high probability will be selected over the ones with low probability. Thus, we suggest using low-temperature values (≤ 0.5) to predict HPC systems to avoid predicting log events with low probability over those with high probability. In contrast, it is recommended to use ≥ 0.7 to perform well with NLP open-ended tasks.

Summary

In this chapter, we propose a novel self-supervised log-based approach called *Clairvoyant* to predict node failures. Clairvoyant solves two main problems with state of the art solution, such as Desh, by (i) being able to capture long-range dependencies and (ii) being amenable to parallelisation. To the best of our knowledge, Clairvoyant is the first attempt to leverage the transformer-decoder technique for failure prediction. Our experiments using two different datasets demonstrate a significant improvement in both prediction accuracy and learning/training performance over Desh – a LSTM-based failure predictor that has been verified as the best in class. The key findings are summarized as follows:

- Clairvoyant can obtain much higher Bleu score (0.90), Rouge score (0.78), MCC score (0.65) and F1-score (0.77) than Desh does (0.58, 0.58, 0.25, and 0.52, respectively).
- Clairvoyant is about $25\times$ and $15\times$ faster than Desh, respectively, during the training and prediction phases.

Clairvoyant significantly enhanced the reliability of HPC systems by predicting their potential failures with high accuracy and speed. However, predicting the lead time of these failures with high accuracy is crucial for proactive management solutions to be scheduled on time. Hence, there is a need to extend and improve the Clairvoyant model to predict the failures and their lead times meanwhile. The next chapter addresses this problem.

Chapter 7

Time Machine: Generative Real-Time Model For Predicting Failure and Lead Time in HPC Systems

Preface

Existing failure prediction methods, which typically seek to extract some information theoretic features to learn failure patterns, fail to scale both in terms of accuracy and prediction speed, limiting their adoption in real-time production systems. Most importantly, current failure prediction solutions do not address the problem of predicting the lead-time to failure, such as Clairvoyant [35] which was presented in the previous chapter. Specifically, Clairvoyant was developed to predict failures only. However, in this chapter, differently from existing work and inspired by current transformer-based neural networks which have revolutionized the sequential learning in the NLP tasks, we propose a *novel* scalable log-based, *self-supervised* model (i.e., no need for manual labels), called *Time Machine*[1], predicting (i) forthcoming log events (ii) the upcoming failure and its location and (iii) the expected lead time to failure. Time Machine is designed by combining two stacks of *transformer-decoders*, each employing the *self-attention mechanism*. The first stack addresses the failure location by predicting the sequence of log events and then identifying if a failure event is part of that sequence. The lead time to predicted failure is addressed by the second stack. Also, we introduce a novel synthetic minority oversampling technique

[1]A *Time Machine* allows us to travel into the future to observe the health state of HPC system and report back. Here, we travel into the log extension to report an upcoming failure.

for online time-based tasks to construct the training instances from failure sequences in the real-time which can be generalized to other domains for time-based tasks (e.g., business, healthcare, etc,). Furthermore, in fault tolerance research, Time Machine is the first framework to convert the prediction of lead times from a regression problem to a multiclass classification problem, which predicts the class for the failure lead time. This method also can be generalized to other domains for similar time-based tasks.

We evaluate Time machine on four real-world HPC log datasets and compare it against three state-of-the-art failure prediction approaches. Results show that Time Machine significantly outperforms the related works on Bleu, Rouge, MCC, and F1-score in predicting forthcoming events, failure location, failure lead-time, with higher prediction speed.

Finally, this chapter shows experimental results by evaluating the impact of our Time Machine on checkpointing-based jobs execution using a discrete event-driven simulator. Evaluation results show that the Time Machine predictor can significantly reduce the total execution overhead, and it is also immune to the inaccurate estimation of the mean time between failures (MTBF).

7.1 Introduction

When errors in the HPC system are not suitably handled, which can occur at specific components (e.g., nodes), then a failure of the system, i.e., more specifically, one or more affected components, may occur. A failure is a special event in the system and results in a special log (e.g., lockup log) to be recorded in the log file. The impact of such failures may be enormous on applications: drastic computational overhead could be introduced, such as through (partial) re-execution, thereby having severe impact on application execution. In an era of exascale computing (i.e., HPC systems executing 10^{18} floating point operations per second), failures are predicted to occur more frequently, exacerbating associated overhead.

To mitigate the impact of failure, efficient failure management strategies are required. Specifically, the development of accurate failure prediction is becoming important, to support the timely deployment of proactive recovery management techniques such as checkpointing/restart or job migration [127, 129]. Unfortunately, the effectiveness of failure prediction tools is still insufficient, thereby necessitating the development of online failure prediction techniques to flag impending failures and their lead-times with high prediction accuracy and speed and with lesser computational overhead. Some works on failure prediction exists, e.g., [92, 126].

In this chapter , we address this important problem of failure prediction by developing and applying *a transformer-decoder* model on HPC log data to build a *generative self-supervised model*, which we call **Time Machine**, to predict two important failure parameters: (i) the failure location, i.e., which nodes will crash, and (ii) the lead time to failure, i.e., how long is left before the predicted failure happens. Our designed Time Machine works as follows: (i) for the location problem, it first predicts the future sequence of logs (future health state) and then identifying if a failure event is part of the predicted sequence and (ii) for the lead time to failure problem, Time Machine reduces the time prediction problem (which is a regression problem) into a self-annotated multi-class classification problem, by predicting the class for the failure lead time. We discuss the motivation of modelling the failure lead time problem in our methodology, section 7.3.2. Note that our work introduces a novel method to construct (no need for manual labels) and augment a self-time annotated training dataset on sequential time-based (timestamps) raw data via an automatic accumulative and iterative process.

The use of generative models for failure prediction is very challenging: (i) erroneous states and failures are rare(r) events, (ii) logs are often incomplete, duplicate, and (iii) messages are alphanumeric in nature and generally lack a proper structure [77], which is very different from the context of, say, text prediction application [34]. Nonetheless, there are many state-of-the-art RNN-based failure prediction methods such as Long Short-term Memory (LSTM) [91] (Desh), Bidirectional Long Short Term Memory (Bi-LSTM) [131], and Gated Recurrent Unit (GRU) [171], which however suffer from non-trivial weaknesses: (i) long training time because of the absence of parallelization in recurrence learning, and (ii) the vanishing gradient problem with loss of earlier “memory”, which may cause limited accuracy.

To address the above limitations, we propose using transformer-decoder based model (called *Time Machine*), which can improve failure prediction accuracy and speed regarding both locations and lead times, through (i) self-attention mechanism and (ii) parallelization which are the crux of transformer neural networks. The reason why transformer-decoder model may work well on failure prediction is that it has the ability to capture long-term dependencies in sequential data (i.e., text) and understand the relationship/order between the words in the sequence (i.e., sentence, paragraph, etc.) via attention mechanism and parallelization. Similarly, the health state for HPC systems can be represented by the sequence of log events (i.e., sequential) generated by their components (e.g., compute nodes) based on timestamps associated with each log event. These consecutive log events are usually related to each other and generated one after another similar to words in the text corpus. For example, failure events usually occur after a sequence of preceding events, including errors/faults that cause those failures. As such, if we consider a log event as a word and a log sequence as a text sequence (e.g., a sentence), we can explore the auto-correlation of event sequences by the transformer-decoder model, which in turn can perform the prediction work.

Additional failure prediction models are developed for HPC systems, however these solutions are mainly based on supervised-learning, requiring extensive data labelling such as [202, 266, 349]. Most unsupervised & self-supervised solutions do not address the problem of predicting the lead-time to failure, such as Clairvoyant [35]. Specifically, Clairvoyant used one stack transformer-decoder to predict failures only. To enable the prediction of failure lead time, there are several key innovative designs proposed in our solution. Our Time Machine framework adopts a two-stack transformer-decoder architecture to predict not only failures but their lead times. The adaptation of the transformer-decoder to predict the failure lead times is based on a novel approach to self-attention: Specifically, the Time Machine framework demonstrates how the self-attention mechanism developed for text prediction is used to predict the failure lead times, by encoding/decoding log events to map each log event onto its timestamp step during the training and prediction phases. This the first work to overcome these limitations by formulating the time prediction as a *self-annotated* multi-class classification problem by predicting the class for the failure lead time. Moreover, the Time Machine can construct training instances in real-time because of our novel synthetic minority oversampling design.

We evaluate Time Machine on four real-world HPC logs and we compare it against LSTM [91] (Desh), Bi-LSTM [131], and GRU [171]. Results show that Time Machine significantly outperforms the best of them: (I) **Log events prediction**: Time Machine obtains a Bleu and Rouge score of up to 0.79 and 0.77 respectively whereas best of the three techniques only has 0.47 and 0.34. (II) **Failure Location**: Time Machine obtains a MCC and F1-score of up to 0.80 and 0.87, respectively, while the best one of the three techniques only has 0.53 and 0.71 respectively. (III) **Failure lead time**: Time Machine is also the best in class, with MCC and F1-score of up to 0.87 and 0.95, respectively. (IV) **Speed-up of training and prediction**: Time Machine is significantly faster than other approaches in both training (5.4$\sim$9.4$\times$ speed-up on average) and chain prediction (over 15$\times$ faster than the related works), making Time Machine very suitable for online failure prediction in real-time production HPC systems.

This chapter is organized as follows: Section 7.2 formulates the research problem. Section 7.3 presents our methodology. In Section 7.4 and Section 7.5, we present the HPC systems and datasets, and evaluation metrics that were used for performance evaluation. Section 7.6 shows the results of our evaluation. We provide a summary at the end of this chapter.

7.2 Problem Formulation

We formulate the research problem as below: Given a log dataset with a sequence of events, our objective is to predict the upcoming sequence of log events and determine if this sequence may contain a failure event; if yes, then predict the lead-time of the failure event as well.

Research challenges: There are two important attributes of failure prediction: (i) **Location**: the component (or node) that would fail/crash should be accurately predicted so that the failure recovery mechanism can be launched at proper “location” and (ii) **Failure lead time**: the time at which the failure log event is predicted to occur should be similar/accurate compared to the one in real-time, otherwise, the failure recovery mechanism would be triggered at wrong time.

We denote the set of log sequences by $\mathcal{L}^r$, where its length is at most r. Suppose we are given two sets: $\mathcal{L}^m$ and $\mathcal{L}^k(k \leq m)$, where the elements in the set $\mathcal{L}^k$ are possible extensions of the elements in $\mathcal{L}^m$. That is, each element in $\mathcal{L}^m$ can be assigned an element from $\mathcal{L}^k$ as an output. Accordingly, for each $\epsilon_i \in \mathcal{L}^m$, $\epsilon'_j \in \mathcal{L}^k$ indicates the true prediction outcome (i.e., the real sequence of log events following ϵ_i). Our failure prediction research is to model a mapping $\mathcal{M} : \mathcal{L}^m \rightarrow \mathcal{L}^k$, in which $\mathcal{M}(\epsilon_i)$ is the predicted sequence which follows ϵ_i, i.e., $\epsilon_i \cdot \mathcal{M}(\epsilon_i)$ is a predicted upcoming log sequence of length $(k+m)$, i.e., $\epsilon_i \cdot \mathcal{M}(\epsilon_i) \in \mathcal{L}^{m+k}$.

We formulate the two problems as follows:

Definition 7.2.1 (**Log Events Prediction**)**.** For a sequence of log events $\epsilon_i \in \mathcal{L}^m$, a predictor $\mathcal{M}$ is expected to be with the minimal distance for the log sequences of length $(m+k)$, i.e., $arg\ min_{\mathcal{M}}\ \mathcal{D}(\epsilon_i \cdot \mathcal{M}(\epsilon_i), \epsilon_i \cdot e'_j)$, where $(\cdot)$ indicates 'sequence concatenation' and $\mathcal{D} : \mathcal{L}^{m+k} \times \mathcal{L}^{m+k} \rightarrow \mathbb{R}$ is the distance measure. $\mathcal{D}$ is a distance metric on two log sequences; $\mathcal{D}$=0 means two logs are identical to each other. If the distance is 0, we claim "$\mathcal{M}$ *correctly extends* ϵ_i"; or else, we say "$\mathcal{M}$ *approximately extends* ϵ_i".

Definition 7.2.2 (**Failure Prediction**)**.** For a predictor $\mathcal{M}$ on a log sequence $\epsilon_i \in \mathcal{L}^m$ with an extension $e'_j \in \mathcal{L}^k$, we say "$\mathcal{M}$ accurately solves the failure prediction" iff $\mathcal{F} \in e'_j \Leftrightarrow \mathcal{F} \in \mathcal{M}(\epsilon_i)$. We say "$\mathcal{M}$ approximately solves the failure prediction problem" if $\mathcal{F} \in \mathcal{M}(\epsilon_i) \Rightarrow \mathcal{F} \in e'_j$.

When $\mathcal{F} \in e'_j$, we say that the extension e'_j is a *failure extension* of ϵ_i and when $\mathcal{F} \in \mathcal{M}(\epsilon_i)$, we say that $\mathcal{M}(\epsilon_i)$ is a *predicted failure extension* of ϵ_i. We also say that ϵ_i is a failure precursor sequence. Note that $\mathcal{D}(\epsilon_i \cdot \mathcal{M}(\epsilon_i), \epsilon_i \cdot e'_j)$=0 means that the predicted lead time of failure event is accurate perfectly, i.e., the failure event would occur right at the predicted moment. Also note that a small value of $\mathcal{D}$ indicates that the failure event occurrence moment is approximately correct in the sequence.

For the failure lead time [93], due to the non-determinism at the system level, it is difficult to accurately predict the *exact* failure lead time. To circumvent this challenge, we propose to model the failure lead time prediction as a multi-class classification problem. We propose a general formal definition of failure lead time as follows.

Definition 7.2.3 (**Lead time to Failure**)**.** Given a log sequence $\epsilon_i \in \mathcal{L}^m$, its extension $\epsilon'_j \in \mathcal{L}^k$ which is a failure extension of ϵ_i, the *failure lead time* of ϵ_i is the difference between the timestamp of the last event in $\epsilon_i \in \mathcal{L}^m$ and the failure event in ϵ'_j and is equal to $TS(\mathcal{F} \in \epsilon'_j) - TS(last(\epsilon_i))$, where TS denotes the timestamp function and last function returns the last element of a sequence respectively.

Let $\mathcal{L}^m$ be the instance space. Every point $\epsilon_i \in \mathcal{L}^m$ is a potential state of the log. Given a pair $\langle \epsilon_i, F(\epsilon_i) \rangle$, where $\epsilon_i \in \mathcal{L}^m$ is a failure precursor sequence, ϵ'_j is a failure extension of ϵ_i and $F(\epsilon_i)$ denotes the failure lead time of ϵ_i, we wish to learn an approximation of the unknown F, denoted by $\hat{F}$ and $\hat{F}(\epsilon_i) = TS(\mathcal{F} \in \mathcal{M}(\epsilon_i)) - TS(last(\epsilon_i))$, where $\mathcal{M}$ is a predictor that solves the failure prediction problem.

Definition 7.2.4 (**Predicted Lead time to Failure**)**.** Given a failure precursor log sequence $\epsilon_i \in \mathcal{L}^m$, its failure extension $\epsilon'_i \in \mathcal{L}^k$, a set of non-overlapping p ranges $R = \{R_1, \ldots, R_p\}, R_i \in \mathbb{Z}^+ \times \mathbb{Z}^+, R_i \cap R_j = \emptyset$ and a predictor $\mathcal{M}$ which solves the failure prediction issue approximately, we say that the *predicted failure lead time* is correct for ϵ_i if $\exists R_i \in R \cdot \hat{F}(\epsilon_i) \in R \Rightarrow F(\epsilon_i) \in R$.

7.3 Time Machine Methodology

Inspired by modern work in NLP tasks, we propose a transformer-decoder based sequential model to predict the forthcoming events, node failure, and failure lead-time in HPC systems. In general, we take the self-attention based language model as an estimator for the posterior probabilities, in which we consider the log events as input words, a sequence of log events as a sentence, and the probabilities of failure in HPC as a context-based generative probabilities. Furthermore, self-attention is friendly to *parallelization*, such that the training and prediction time can be significantly reduced by leveraging parallel techniques, compared to existing state-of-the-art failure prediction methods, such as RNN-based methods used in [91, 131, 171]. To this end, we develop a novel real-time online approach namely **Time Machine** which is fully self-supervised without the need for labeling by HPC system administrators. As shown in Figure 7.1, the architecture of our model consists of **two** transformer-decoder neural network components, and both of the two decoders are based on the transformer-decoder variant [272]. The **first** transformer component aims to predict HPC node failures with two major steps: (1) for each node, it predicts the sequence of future events (or future health state); (2) it determines weather a failure is included in the predicted sequence. The **second** transformer component aims to predict *lead times*, based on which one or more proactive fault-tolerant techniques can be accurately selected in time ahead of the failure occurrence.

As for the Time Machine methodology which adopts two stacks of transformer-decoder to predict HPC system failures and their lead-times, its framework design also includes three key innovative points:

- Transformer neural networks are employed originally for NLP tasks such as text classification, text generation, summarisation, while the **Time Machine** method is the first framework utilizing transformer architecture to predict the lead-time to failures. This method can be generalized to other domains for time-based prediction tasks.

- Time Machine introduces a novel synthetic minority oversampling technique for online time-based tasks to construct the training instances from failure sequences. This method can be generalized to other domains for time-based tasks (e.g., business, healthcare, booking business).

- In the fault tolerance research, our Time Machine method is the first study to reduce/convert the time prediction problem (a regression problem) into a self-annotated multi-class classification problem, by predicting the class for the failure lead time. Also, This method also can be generalized to other domains for similar time-based tasks (e.g., business, healthcare, booking business).

We detail our proposed framework in the rest of this section.

7.3.1 Node Failure Prediction

Phase I. Log Event Prepossessing

In the first phase, similar to the tokenization in NLP task, we first convert the log message into an ID based log event sequence: $e_1, e_2, ..., e_m$, where m is the length of event sequence, $e_i \in \{t_j | j = 1, 2, ..., T\}$ represents the i-th event, and t_j stands for all possible event types in the log event prediction. Besides, we let $m < 1024$ in order to make it possible that all input event sequences share similar length for parallel processing, which is distinct from the existing RNN methods. Moreover, the Byte Pair Encoding (BPE) method is utilized to tokenize the input to encode any unusual tokens (IDs of log entries).

Phase II. Log Events Learning and Failure Prediction

Our proposed approach aims to take the self-attention mechanism to improve the connectivity among the events in log sequences. Accordingly, we utilise transformer-decoder architecture, a stack of decoder blocks preceded by an input layer to embed the real-time log events sequence logged by the HPC system component node, and then followed by linear and softmax layers to predict failures (e.g., node crashes, networks failures) by two steps: predicting the future sequence of events and then identifying if a failure is part of the predicted sequence. More details are described in the following text. We refer the readers to read [272] for detailed background of the *transformer variant* which we will use to build our model.

We summarize the current masked language (failure and lead-time prediction in our case) model as follows:

In a typical transformer block ℓ, assuming the input feature for token e_i in $l-1$-th layer of transformer is e_i^{l-1}, the information propagation process is given by:

$$v_i^{l-1} = \text{Self-Attention}(e_i^{l-1} | e_1^{l-1}, e_2^{l-1}, ..., e_m^{l-1}) \tag{7.1}$$

$$\Phi(v_i^l) = \varphi(W^l v_i^{l-1} + b^l) \tag{7.2}$$

$$e_i^l = \text{LayerNorm}(\Phi(v_i^l) + e_i^{l-1}) \tag{7.3}$$

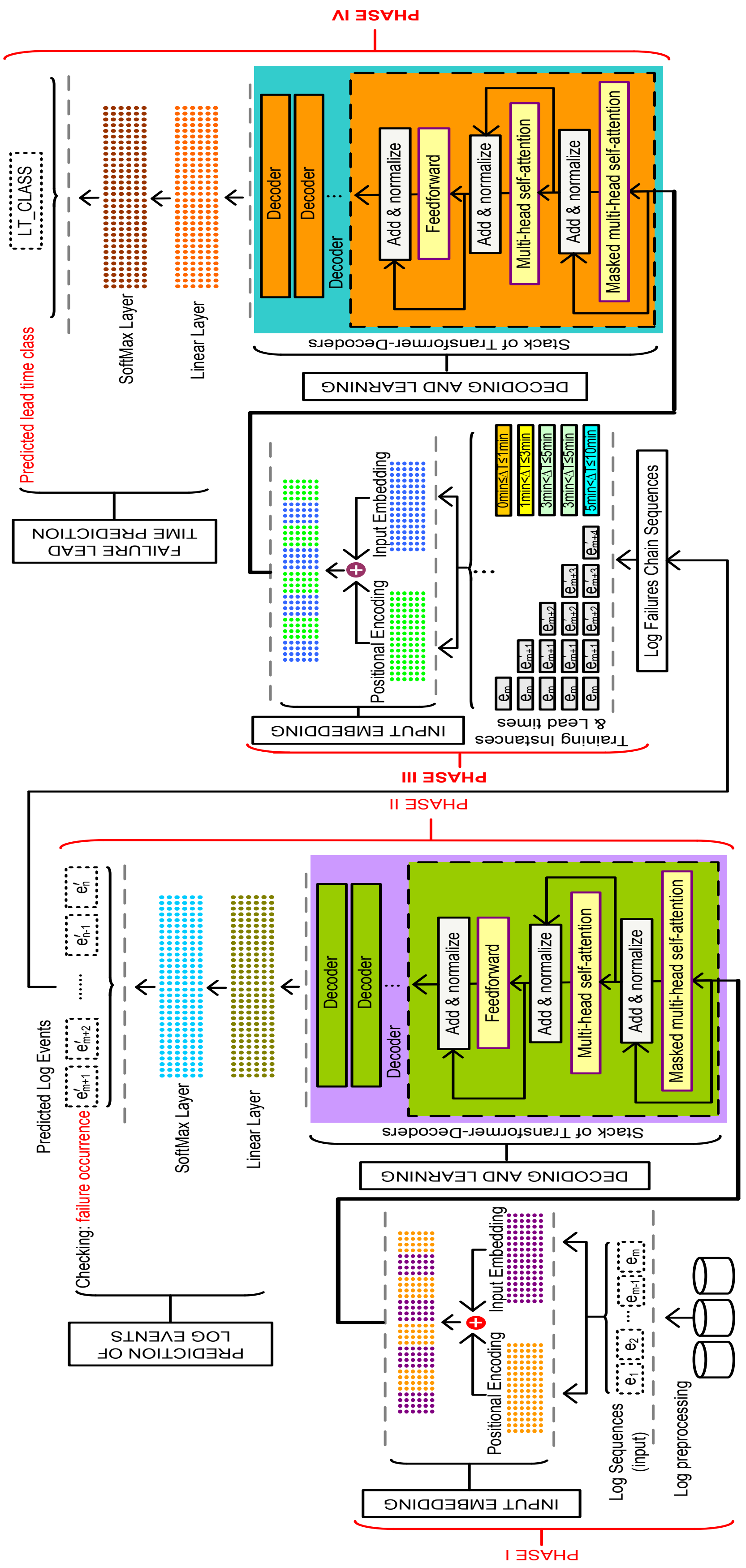


Figure 7.1: Illustration of Health State/Failure/Lead Time Prediction Phases in Time Machine

where e_i^l is the learned feature for e_i in l-th layer, v_i^l is the corresponding value vector in the regard of the self-attention mechanism according to e_i^l, φ is an element-wise nonlinear function applied to a feed-forward layer, whose weight matrix, $W^l \in \mathbb{R}^{n_l \times n_{l-1}}$, transforms the feature dimension from n_{l-1} to n_l, Self-Attention(e^{l-1}) returns the weighted value vector of all input representations where weights are derived by multiplying the query vector of the current input e^{l-1} with the key vectors from other inputs. Between every two transformer blocks, there is a skip-connection and a layer normalisation. The former mechanism bypasses the transformer block ℓ and adds the input e^{l-1} directly to the output v^l of this block, while the latter normalises the input across the feature dimension.

STEP 1: Transfer Learning Based Sequence Prediction

The main idea in pre-trained language model, such as GPT-2 [272], aims to predict a particular word based on its context by: $P(e_i|e_1, e_2, ..., e_{i-1}, e_{i+1}, ..., e_m)$. However, in the prediction of HPC log events, the tokens that follow the expected prediction e_i are unseen by the model. Furthermore, the vocabulary used to represent log event types is much smaller than that used in typical NLP tasks, which may lead to overfitting if we simply train an over-parameterised model. Therefore, we propose to use pre-trained model (GPT-2) from NLP, which is almost isotropic, to initialise the transformer-decoder model to predict the future log event by fine-tuning the parameters on HPC dataset. Here, we define the probability of future log event by Softmax $P(\hat{e'}_{m+i}|e_1, e_2, ..., e_m) = \text{Softmax}(\text{FNN}(e_i^L|e_1, e_2, ..., e_m))$, where the FNN stands for the feedforward neural network which is a linear layer to transform the last decoder outputs L into higher-dimensional logits vectors using a linear transformation and non-linear activation function. Softmax layer is the final layer applying softmax function to the linear layer's output and converts it to a probability distribution. According to definition 7.2.1, we use the cross-entropy as the metric to measure the distance between distribution in loss function,

$$\mathcal{L} = \sum_{i=1}^{n-m} P(e'_{m+i}) \cdot \log(P(\hat{e'}_{m+i}|e_1, e_2, ..., e_m)) \tag{7.4}$$

where the estimated probabilities of $P(\hat{e'}_{m+i})$ are defined by the softmax function with the learned vectors along to the last FNN layer, and the $P(e'_{m+i})$

is the true output from training corpus, which is an advantage that, in such learning architecture, we do not require a specific annotation for self-supervised learning. The sequence of log events can be generated from large scale of raw data automatically. In this way, the pre-trained language model can be easily adapted to the log events prediction task in real-time.

STEP 2: Failure Prediction: Based on the prediction of log events, we can generate a log events sequence by a given $\{e_1, e_2, ..., e_m\}$, marked as: $\{e_1, e_2, ..., e_m, e'_{m+1}, ...e'_n\}$, where $\{e_i | i \leq m\}$ is the given event and $\{e'_j | m+1 \leq j \leq n\}$ is the predicted event. According to definition 7.2.2, the failure prediction aims to identify if the e'_j is the failure extension of e_i. Hence, we can convert the generated event e'_j to the unique ID to check if it is the failure. Here, we suppose the whole vocabulary of log events is $\mathcal{V}$, which contains two subsets, the failure events $\mathcal{V}^f$, and the normal events $\mathcal{V}^n$, where $\mathcal{V}^f \cup \mathcal{V}^n = \mathcal{V}$ and $\mathcal{V}^f \cap \mathcal{V}^n = \emptyset$. Then, one can easily quantify that a predicted log event is a failure if it is a member of failure events set ($e'_j \in \mathcal{V}^f$).

7.3.2 Predicting Lead Times to The Node Failure

One key novelty in this chapter that is significantly different from existing transformer based sequence models used for NLP tasks, is predicting lead times for the failure events such that appropriate proactive methods could be triggered in time, which is handled by Phase III and Phase IV.

Phase III: Failure Sequences Construction for Lead-Time Prediction

In order to predict the node failures' lead times, the first and foremost step is establishing and preparing a dataset based on failure chains (i.e., the Phase III as presented in Figure 7.1). Our framework can be easily deployed for HPC systems in real-time, because the training/testing datasets from the failure chains and associated labels (i.e., lead times) are created automatically (**no need for manual labelling**) based on log events' timestamps.

We introduce a **novel synthetic minority oversampling technique** for online time-based tasks to construct the training instances in the real-time from failure sequences as follows. In our model, predicting a node's failures ahead is achieved through accurately predicting the forthcoming log events $\{e'_{m+1}, e'_{m+2}, ..., e'_n\}$. Without loss of generality, we assume that the predicted log events sequence ends with a failure event since the motivation of our proposed method aims to predict the failure. Hence, based on the given log events sequence $\{e_1, e_2, ..., e_m\}$ and the proposed events/failure prediction methods, we have a failure chain of $\{e'_{m+1}, e'_{m+2}, ..., e'_n\}$, where $\exists e'_j \in \mathcal{V}^f$.

To predict the lead time for any concrete failure chain, we then use the timestamp $TS(\cdot)$ to estimate the lead time when the failure appears for a given sequence of log events. Intuitively, we only need to predict the $TS(e'_j)$, where $\{e'_j \in \mathcal{V}^f\}$. However, considering that the size of $\mathcal{V}^f$ is limited, it is essential to design a smoothing method to overcome the potential risk caused by over-fitting. Hence, we propose to utilise the transformer-decoder based method to approximate the $TS(\cdot)$ for both $e'_j \in \mathcal{V}^n$ and $\mathcal{V}^f$, and take advantage of sequential model to guarantee the latent pattern $TS(e'_i) < TS(e'_j)(i < j)$ is true, to achieve both reasonable and stable lead-time prediction. Specifically, to make the trade-off between efficiency and accuracy, we convert the prediction of lead times from a regression problem to a multi-class classification problem which predicts the class for the failure lead time. Such a design is motivated by the fact that there are only a few proactive recovery techniques used in practice (e.g., less than 10 techniques), and each technique requires a specific lead time to launch. Moreover, the correction/proactive actions generally require approximately estimated lead times instead of the exact lead times. Accordingly, we define 6 lead-time classes $\hat{y}_j$ in our study: $\hat{y}_j \in \{$[0min,1min], (1min,3min], (3min,5min], (5min,10min], (10min,15min], (15min,∞)$\}$. Our model is flexible in increasing/decreasing lead time classes based on the system recovery mechanism.

We use an example to describe how to construct the training instances. Without loss of generality, suppose a failure chain contains 6 log events including the failure event with associated timestamps: $((e_m, 01{:}00{:}00), (e'_{m+1}, 01{:}00{:}30), (e'_{m+2}, 01{:}01{:}00), (e'_{m+3}, 01{:}03{:}10), (e'_{m+4}, 01{:}04{:}55), (e'_{m+5}, 01{:}07{:}16))$, where the $e'_{m+5} \in \mathcal{V}^f$ is the failure event. The training instances and their associated lead time classes are constructed automatically in terms of the failure chain iteratively, as illustrated in Figure 7.2.

As all log sequences training instances are created from the failure chains

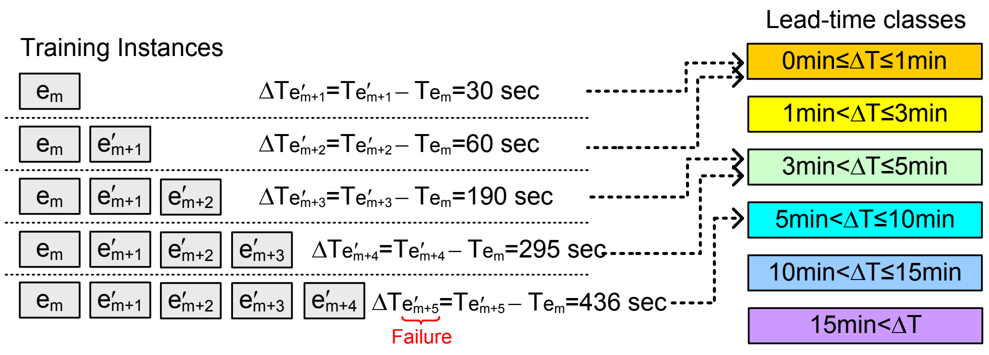


Figure 7.2: Lead Time Training Instances Construction

as described above, the lead-times have been associated/mapped to the corresponding lead-time classes. Note that this process is conducted during runtime model deployment, and all log event instances are assigned the same tokens as discussed in 7.3.1 before being embedded/fed into the second transformer-decoder stack.

Phase IV: Lead-Time Learning and Prediction

Based on the prediction of log events, we can generate a log events sequence by a given $\{e_1, e_2, ..., e_m\}$, marked as: $\{e_1, e_2, ..., e_m, e'_{m+1}, ...e'_n\}$, where e_i is the given event and e'_j is the predicted event. According to sequence generation in section 7.3.2, the lead-time prediction aims to identify the label of y_j for the failure extension of e_i with length j, a.k.a $\{e_1, e_2, ..., e_m, e_1, ...e_j\}$. Hence, we convert this task into sequence classification, in which we employ the fine-tuned transformer-decoder-based model to extract the last representation R_{e_j} of the token e_j, to approximate the posterior probability according to the failure label in real-world datasets, by a softmax probability:

$$P(y_j) = \text{Softmax}(\text{FNN}(\text{decoder}(R[e'_j]|e_1, ..., e_m, e'_{m+1}, ..., e'_j))) \tag{7.5}$$

In general, any loss function or pre-trained language model can be deployed for the approximation. Without loss of generality, we choose the cross-entropy as loss function and the GPT-2 as decoder in our implementation. In summary, $\forall$ log events sequence $e_1, e_2, ..., e_m$, we can predict the failure extension of $e'_{m+1}, ..., e'_n$ by minimising the loss function defined by equation.7.4. According to equation.7.5, we can then predict the lead time to failure of $TS(e'_j) - TS(e_m)$. This proposed framework requires no annotation or supervised signal but facilitates optimising the process which can select the lowest computation cost correction/recovery mechanisms to correct and fix HPC system errors before the failures occur.

7.3.3 Featuring Real-Time in Time Machine

Deploying the *Time Machine* online in real-time requires fine-tuning the model parameters in case new log sequences and failure patterns are encountered. Thus, the teacher forcing technique [317] is proposed to complement *Time Machine* in real-time. The integration between the *Time Machine* and teacher forcing approach enables online training, learning, and prediction by using ground truth input (i.e., the real log events generated by the HPC system in real-time) instead of our model output (the log events predicted by *Time Machine*) from a previous time step as an input. This integration can cope with any new types of log sequences and emerging failure patterns because of various cases, such as upgrade of the HPC system components (i.e., software, hardware, services). The new jobs (e.g., applications) running on HPC systems can also induce new log patterns that have not been met before. Moreover, the teacher-forcing technique forces the real-time log event learning/prediction under the *Time Machine* to be conducted on correct log events (i.e., the correct log sequences generated by the system) rather than log sequences predicted ahead by *Time Machine* (which may be incorrect prediction).

End-to-end use: Once the model is trained, it is deployed onto a supercomputer to predict its failures in real-time. As the jobs are launched on the allocated compute nodes, log messages will be generated on system components and mapped to associated log IDs by our log parser. Then, our model simultaneously begins to predict a sequence of log events for each component (e.g., compute node) using the first stack of transformer-decoder. The predicted log sequence will be updated as a new event is generated on each component. If the predicted sequence contains a failure event, our model immediately calculates its predicted lead-time class to the actual occurrence based on the log sequence, via the second stack of transformer-decoder in our framework. This can inform a failure handling mechanism, e.g., checkpointing. Depending on the predicted lead-time to the failure, a low-cost recovery mechanism is chosen to trigger at an appropriate time.

7.4 Production Systems and Datasets

Table 7.1 shows the four **unlabeled** log datasets and their supercomputers characteristics which used in our experiments in this chapter. The four system logs are generated from three different real-world supercomputers clusters. Specifically, these system are of various scales (from 200 nodes to 5600 nodes), various interconnects (Infiniband and Aries Dragonfly), different file systems (Luster, MarFS, etc.), different processors, and different logging mechanism. The log datasets are (i) Syslogs (4.2.2), (ii) Rationalized Logs (abbreviated as RatLogs) (4.2.2). Both Syslogs and RatLogs are collected form Ranger supercomputer(operated by Texas Advanced Computing Center (TACC)) at different time. (iii) Cray XC30 logs (4.2.4) generated by Cray XC30 supercomputer , and (iv) Cray XC40 logs generated from Cray XC40 (Mutrino) (4.2.5) supercomputer which was managed under a joint effort between Los Alamos National Laboratory (LANL) and Sandia National Laboraties (SNL). Mutrino, sited at SNL. Both Cray XC30 logs and Cray XC40 logs consist of two different types of logs (console and message). More details about the four system logs and their supercomputers clusters are found in Chapter 4, Section (4.2).

7.5 Evaluation Metrics

Time Machine predicts (i) forthcoming log events (the entire health state of each node in the system) (ii) the node failure, and (ii) the expected lead time to failure. Therefore, our model is evaluated in three aspects. (1) We evaluate the accuracy by comparing the log events predicted by *Time Machine* versus the actual log events generated in reality by the four HPC systems using two text generation metrics (*Bleu* and *Rouge*). *Bleu* and *Rouge* metrics can complement each other for the NLP text generation (upcoming log events prediction in our case) tasks evaluation. Specifically, they correspond to the precision measure and recall measure, respectively. More details about these two metrics and how we use them to evaluate our model performance are found in the previous chapter 6.4. (2) We evaluate the prediction accuracy of our model regarding the nodes' failure events using several standard metrics including recall, precision, F1_score, MCC, false-positive rate, and false-negative rate. We removed the log events predicted by our model (candidate) and the actual log events generated by the HPC system (reference) except failures in order to employ these metrics. More details about these metrics are also found in the previous chapter 6.4. (3) We evaluate the prediction accuracy of our model regarding the lead-time to node failures based on standard metrics: recall, precision, F1_score, and MCC. In this case, however, TP, FP, FN, and TN refer to True Positive (the actual time class is predicted correctly), False Positive (the actual time class is predicted incorrectly), False Negative (the actual time class is missed by our model) and True Negative (the negative class is predicted correctly), respectively.

7.6 Performance Evaluation

To show the efficacy and applicability of our failure&lead-time prediction method, we carefully evaluate the performance of our model on four real-world supercomputer system logs: (i) SysLogs (4.2.2), (ii) Rationalized logs (4.2.2), (iii) Cray XC30 logs (4.2.4), and (iv) Cray XC40 logs (4.2.5). They were logged by three different supercomputers and four different logging mechanisms at different operational times, which are all **unlabeled**. We compare our approach to three state-of-the-art deep learning prediction techniques (a.k.a., baselines in the following text): Desh (LSTM) [91], Bi-LSTM [131], and GRU [171]. These three related works employ LSTM, Bi-LSTM, and GRU neural networks to predict HPC failures, respectively, and they have been verified as the best in class. We do not compare our work to traditional machine learning (ML) (e.g., Random Forest, SVM, DT, KNN) for two reasons. First, our model is a self-supervised learning that does not need labels whereas ML methods depend on labeled data (i.e., supervised learning-based techniques). Second, even ML algorithms can be utilised for classification (e.g., anomaly detection) and regression tasks, however, they are not designed to resolve text generation (prediction) tasks which is our research problem.

Time Machine is implemented using Python 3.8, Pandas [24], Keras [7] with a TensorFlow [17] backend, PyTorch [13]. We verified the experiments over the environment of Google Colab [20] by leveraging NVIDIA Tesla V100 GPU, Intel Xeon 2.30 GHz processor, and 52 GB RAM. In what follows, we show and discuss the major evaluation results.

7.6.1 Log Data Preprocessing

We preprocess the log data by sorting the log events according to timestamps, cleaning raw messages, and removing the duplicate messages in terms of the spatial and temporal correlations. Consequently, these log messages are converted to log sequences regarding their associated nodes, which corresponds to the phase I of our methodology. As shown in Table 7.1, the quantities of the datasets' log messages are reduced significantly after the preprocessing step. Specifically, a total of 83087, 25272, 127161, and 49391 log sequences are constructed from Syslogs, Rationalized logs, Cray XC 30 logs, and Cray XC 40 logs, respectively. Each of the four logs is divided into training part and testing part. The training part accounts for 80% of the logs' data, while the testing part accounts for the remaining 20%.

Table 7.1: Analysis Logs before and after the Preprocessing Phase

Log name	Duration	# raw logs	# filtered logs	Percentage decrease
Syslogs	5 mon	43.6 m	2.3 m	94.7 %
RatLogs	6 mon	361 m	8.1 m	97.8 %
Cray XC30	1 mon	133 m	15.3 m	88.5 %
Cray XC40	16 mon	237 m	5.9 m	97.5 %

7.6.2 Training and Prediction Time Performance

Time Machine remarkably decreases the overall training time compared to the three state-of-the-art prediction approaches (LTSM (Desh), BiLSTM, and GRU). The overall training time includes two parts: (i) the training time in the regard of the prediction of the log event patterns and (ii) the training time for the prediction of the lead time to node failures. For the four HPC Systems data logs, *Time Machine* takes only 3.53 hours for the overall training on average, while other related works (LTSM, BiLSTM, and GRU) require 14.54 hours, 25.53 hours, and 13.22 hours, respectively. Also, our model is 15× faster over all baseline solutions in predicting the forthcoming log sequence of events. The training and prediction time speed-up results are detailed as follows.

Log Events Training Time Performance

The training time for learning to predict the log sequences and identifying failure patterns, which is addressed by the first transformer-decoder stack, is drastically reduced 5.4 ∼ 9.4× compared to three other state-of-the-art methods on average as shown in Table 7.2. *Time Machine* requires only 0.7∼3.83 hours in training, while LTSM, BiLSTM, and GRU require 3.79∼20.75 hours, 6.69∼36.66 hours, and 3.66 ∼18.88 hours, respectively.

Table 7.2: Log Events Training Time Performance in Hours

	Time Machine	**LSTM**	**Bi-LSTM**	**GRU**
SysLogs	1.60	8.86	15.30	7.89
Rationalized Logs	0.70	3.79	6.69	3.66
Cray XC 30	3.83	20.75	36.66	18.88
Cray XC 40	1.17	6.31	10.33	5.74
Average	1.83	9.93	17.25	9.04

Lead Time Prediction Training Time Performance

The training time for learning to predict the lead time to node failures, which is addressed by the second transformer-decoder stack, also drastically decreased 3.74 ∼ 7.23× compared to three other state-of-the-art strategies, as illustrated in Table 7.3. Based on all the four HPC systems data logs, *Time Machine* requires only 1.71 hours of training on average, while the other related works (LTSM, BiLSTM, and GRU) require 4.61 hours, 8.29 hours, and 4.17 hours, respectively.

Log Events Prediction Time Speed-up Performance

Our model has the highest speed on the prediction of forthcoming log sequence of events (chain lengths), as shown in Figure 7.3. As evaluated in our experiments, the speedup of our model in predicting the forthcoming log sequence

Table 7.3: Lead Time Training Time Performance in Hours

	Time Machine	**LSTM**	**Bi-LSTM**	**GRU**
SysLogs	1.2	3.17	5.91	2.84
Rationalized Logs	0.54	1.49	2.69	1.34
Cray XC 30	3.86	10.42	18.32	9.44
Cray XC 40	1.24	3.35	6.24	3.09
Average	1.71	4.61	8.29	4.17

of events is 15× faster over all baseline solutions (LSTM (Desh), Bi-LSTM and GRU). The Figure 7.3 shows that only 5.78 secs are needed to predict a log sequence with the chain length of 1024, whereas state-of-the-art methods require 96∼167 secs.

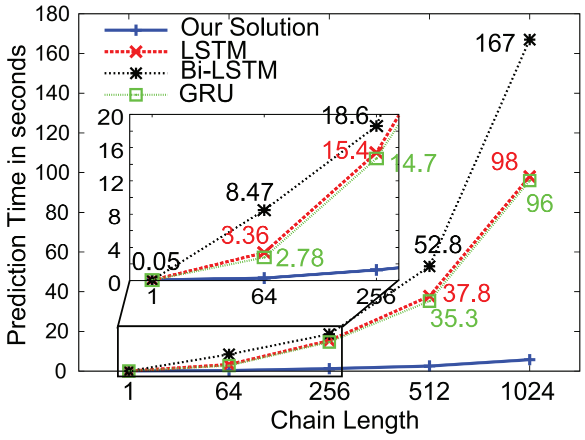


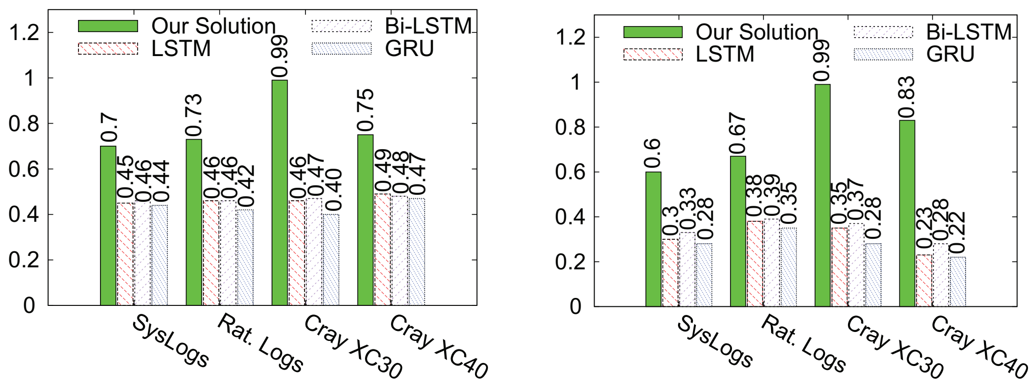
Figure 7.3: Prediction Time of Chain Lengths

The low training time and high prediction performance of our model are attributed to the transformer-decoder mechanism's parallelization capability and positional encoding. More specifically, it takes considerably less time than the RNN models (baselines) because the RNN model lacks parallel training and requires sequential learning. On the one hand, optimization of training time in HPC systems is significant because deploying the model online in real-time requires multiple fine-tunings of the model parameters over time in case of new log sequences and failure patterns appear. HPC system operators frequently elevate system components (software/hardware) and services to add new components (i.e., hardware or software) to improve high-performance computing demands. The increased number of high-resources-hungry jobs (e.g., applications) scheduled day-to-day on HPC systems also induces the logging management systems to generate new log patterns that have not been learned.

On the other hand, Time Machine is particularly suitable for real-time failure prediction because of the high-speed prediction of forthcoming log sequence of events (chain lengths). It is noted that the growth of the number of events in the log chain does not come up with higher prediction times using *Time Machine*, whereas the baselines consume a long time to predict the same log sequence, and the speedup turns more and more obvious with increasing log chain lengths. So, our model is suitable for the real-time use-case with vast amount of logs generated in a short time (seconds), especially when the HPC components face erroneous behaviors that may lead to component crashes. Consequently, the high-speed prediction achieved by our model can boost the selection of the most suitable failure recovery action.

7.6.3 Overall Learning & Log Events Prediction Performance

We also evaluate the overall accuracy of our model in predicting the forthcoming log events (e.g., normal, errors, or failures) before the actual events occur, with respect to predict the entire system health state.



(a) Bleu (b) Rouge

Figure 7.4: Entire Health State Prediction Accuracy

Figure 7.4 (a) and (b) show the Bleu measure and Rouge measure of the entire health state prediction. Bleu and Rouge measure the degree of similarity (overlapping) between the candidate solution (predicted log events by our model or the baselines (LSTM, Bi-LSTM and GRU)) and the reference (log events generated by the supercomputer system in realtime). As shown in the figure 7.4 (a), our transformer decoder-based approach achieves a Bleu score of 0.70∼0.99 in predicting forthcoming log events based on the four system logs. In contrast, the other three state-of-the-arts have much lower Bleu scores (in the range of only 0.4∼0.5). On average, 79% of log events predicted by our model appeared in the events generated by the HPC system (the reference), compared to just 47% by Bi-LSTM (the best score from among the three baselines).

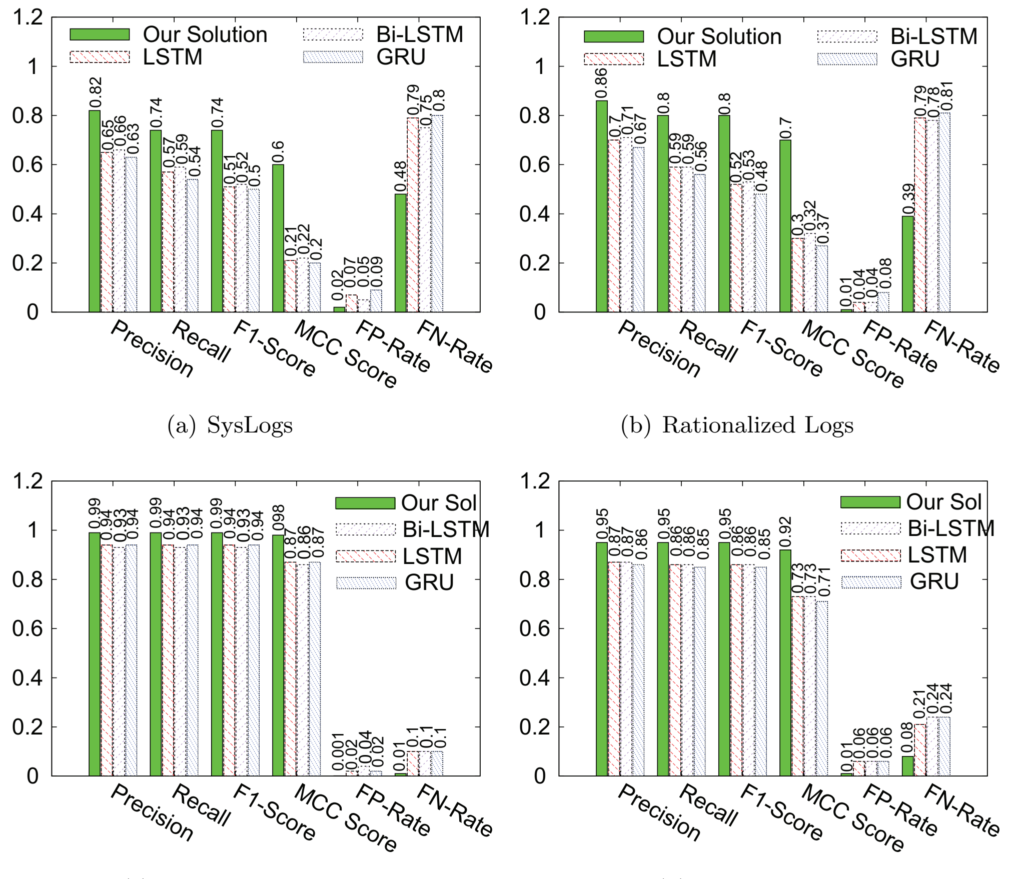
Also, our solution has a significantly higher Rouge score than the other three methods as depicted in Figure 7.4 (b). *Time Machine* obtains Rouge scores of 0.60, 0.67, 0.99 and 0.83 on the four system logs (SysLogs, Rationalized logs, Cray XC30 logs, and Cray XC40 logs), respectively. The Bi-LSTM baseline, however, obtains the Bleu scores of only 0.33, 0.39, 0.37 and 0.28, respectively. Similarly, both LSTM (Desh) and GRU based prediction solutions also have fairly low Rouge scores, which are in the range of $0.22 \sim 0.39$. On average, $\approx 77\%$ of events generated by the supercomputer systems in real-time (the reference) appear in the log events predicted by *Time Machine* (candidate), compared to just $\approx 34\%$ by the best state-of-the-art prediction solution (i.e., Bi-LSTM).

We note that all the baseline solutions can hardly capture long-range dependencies/correlations between the events of long sequences, because they are all RNN-based models, which suffer from the memory loss issue for earlier events because of the vanishing gradients. By comparison, our solution is able to predict the upcoming log sequence as long as it correlates to the preceding events, as manifested by a high match between the forthcoming log events under our prediction model, and the events generated on the real system. In particular, the self-attention mechanism can efficiently identify the log entries of important events while moving the focus away from irrelevant ones and capturing long-range dependencies and correlations between events in long sequences.

7.6.4 Node Failure Prediction Performance Evaluation

Figure 7.5 shows the prediction accuracy of failure events under our model and baselines. We apply six measurements (Recall, Precision, MCC Score, F1-score FP-Rate, and FN-Rate) to evaluate our candidate solution and reference, based on logs with removed non-failure events.

As presented in Figure 7.5, *Time Machine* predicts upcoming node failures with high average precision score (0.91) on the four HPC system logs. In



(a) SysLogs

(b) Rationalized Logs

(c) Cray XC30 Logs

(d) Cray XC40

Figure 7.5: Failure Prediction Performance

comparison, the best baselines are LSTM and Bi-LSTM, whose average scores (0.79) are lower than our model. For example, 82% of Ranger SysLogs node failures predicted by *Time Machine* indeed appear in the actual events generated by Ranger HPC system, compared to only 66% by Bi-LSTM (the best score from among the three baselines). Also, the results show that our technique obtains a better recall accuracy with an average score of 0.87 on the four HPC system logs. In comparison, the best baseline (both LSTM and Bi-LSTM) obtains the average score of only 0.74. *Time Machine* achieves a recall score of 0.74 on Ranger SysLogs; Bi-LSTM (best-baseline score) obtains a recall score only 0.59. This means on average, 74% of actual node failures generated by Ranger appear in SysLogs can be predicted by *Time Machine*, compared to only 59% by Bi-LSTM.

According to Figure 7.5, our solution has much higher MCC scores and F1-scores than all other baselines. Specifically, our model achieves better prediction on the four system logs with MCC scores of 0.6∼0.92, and the f-scores reach 0.74∼0.99, which are both much higher than that of other baselines. For example, for SysLogs and Rationalized Logs, the MCC scores of our model can reach 0.6 and 0.7, respectively, which are much higher than the baselines' (0.2∼0.22 and 0.3∼0.37).

Furthermore, the significant improvement of our model over baselines is also manifested by false positive rate (FP-rate) and false negative rate (FN-rate) (see Figure 7.5). For example, our model incurs only 1% FP-rate and only 24% FN-rate on average for all the four system logs, indicating that incorrect recovery actions are seldom triggered. However, the three baselines face higher FP-rate (5%) and FN-rate (47%). FN-rate is high in Syslogs and Rationalized Logs because the four models are not exposed to enough failure patterns for the prediction. Also, the interleaved logs are not removed before predicting failures. Thus, FN-rate could be significantly reduced in real production where our model trains on the massive amount of log data where most of the failure patterns can be encountered and learned and when a log parser is used to remove unimportant interleaved logs. Moreover, experts agree that the prediction model's usefulness remains highly valuable despite its limited accuracy. Even if only half of the failures are accurately predicted, half of the expensive global coordination needed for recovery management actions (e.g., checkpoints) can be avoided [91].

We explain the significant performance gain of our model over baseline models: Different log sequences are observed between the failure events and their associated errors, (e.g., kernel OS process, memory errors and network errors etc). Both failure relevant and irrelevant logs will interleave in such sequences, resulting in lengthy log sequences. As such, it may take hours for failure to ensue following an error message. In contrast, our transformer-decoder-based model leverages multi-head masked attention layers and the positional encoding technique, which can completely avoid recursion, processing log sentences as a whole and understands associations between log events, leading to higher prediction capability.

In recent years, the time between failures has shrunk from hours to minutes [132]. This is because of significant HPC systems components' expansion in scale and design. Also, it is observed that most node failures are attributed to applications (jobs) job issues [79], meaning that the applications' bugs lead to most of the node failures, especially when the job's requirements exceed a node's resource capacity [93]. For instance, some applications cause out-of-memory (OOM) problems, eventually leading to node crashes. Moreover, multiple errors are related to the jobs' configuration problems, which may indirectly lead to other file-system and hardware errors. Furthermore, most of the failed nodes at similar times usually share the same job ID.

7.6.5 Lead Time Prediction Performance Evaluation

This section details the performance of the lead-time prediction to failure events. As mentioned previously, the main goal of this chapter is to predict not only node failures but their lead times. This is important as appropriate recovery techniques can be triggered in timely manner, based on the failure lead-time.

Three key points need to be clarified first as follows:

- The failure prediction techniques (Bi-LSTM based method [131] and GRU-based approach [171]) do not support lead-time prediction originally. In our research work, we implement a lead time component for each of them based on Bi-LSTM and GRU neural networks, respectively, so that they are enabled to predict lead-time classes.
- For fairness, we reduce the time-lead prediction problem into a self-annotated multi-class classification problem for the baselines (LSTM [91], Bi-LSTM [131], and GRU [171]).
- We define 6 lead-time classes; however our model is flexible in increasing/decreasing lead time classes or increasing/decreasing the time range for each class based on the HPC system recovery mechanisms requirements. Accordingly, HPC system administrators can decide the number of classes and their time ranges. We also recommend the number of classes, and their time ranges are selected based on the number of existing recovery techniques and the times they take from the triggering until complete. Furthermore, the number of these classes and their criteria can be modified and updated for any reason, such as if new recovery

Table 7.4: Lead Time To Failure Prediction Performance Evaluation on 4 HPC Systems Data Logs

		Time Machine	**LSTM**	**Bi-LSTM**	**GRU**
SysLogs	Precision	0.92	0.91	0.87	0.91
	Recall	0.93	0.93	0.91	0.93
	F1-Score	0.92	0.91	0.89	0.91
	MCC Score	0.76	0.74	0.68	0.74
Rationalized Logs	Precision	0.92	0.89	0.89	0.89
	Recall	0.92	0.90	0.91	0.91
	F1-Score	0.92	0.89	0.89	0.89
	MCC Score	0.83	0.77	0.78	0.77
Cray XC30 Logs	Precision	0.99	0.99	0.99	0.99
	Recall	0.99	0.99	0.99	0.99
	F1-Score	0.99	0.99	0.99	0.99
	MCC Score	0.95	0.91	0.92	0.92
Cray XC40 Logs	Precision	0.97	0.95	0.95	0.95
	Recall	0.97	0.95	0.95	0.95
	F1-Score	0.97	0.95	0.95	0.95
	MCC Score	0.94	0.91	0.92	0.91

techniques and features have emerged with launch times that cannot fit into pre-selected classes. This flexibility feature allows our model to work for any system.

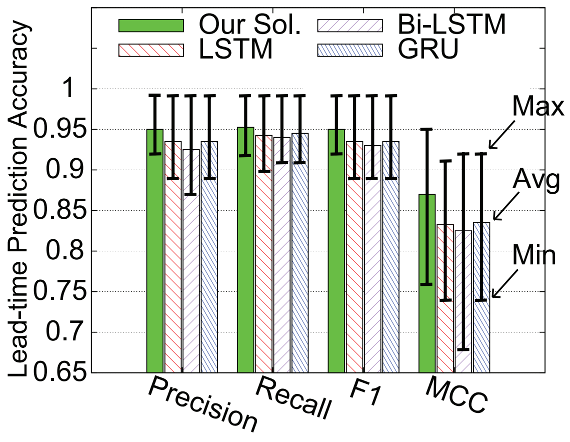


Figure 7.6: Lead Time Prediction Performance

Table 7.4 and Figure 7.6 show the failure lead-time prediction results with two critical observations. On the one hand, our model always has the highest accuracy from among all the four solutions. In absolute terms, the MCC and F1-score under our model can reach up to 0.87 and 0.95, respectively, which are higher than the scores resulted from the three state-of-the-arts (0.87 and 0.94, respectively). In particular, for SysLogs and Rationalized Logs, the MCC scores of Time Machine can get up to 0.76 and 0.83, respectively, while the baselines'

MCC scores are 0.74 and 0.78. On the other hand, all the four prediction methods have relatively high and similar accuracy in predicting failure lead-times on the four real-world system logs. Such a high and similar accuracy is primarily attributed to two reasons (i) modelling the lead-time prediction as a classification task rather than a regression task, (ii) our novel oversampling technique constructs massive, sufficient, and self-supervised training datasets. That is, the four prediction techniques are trained efficiently to learn each individual log event weight to estimate the class for the failure lead time. The reason we can model the lead-time prediction as a classification task is that there are only a few correction/recovery actions in total, and each action requires approximate lead times rather than exact lead times. The proactive recovery and error correction approaches may have largely different triggering/recovery costs. The typical proactive fault tolerance methods include job migration, checkpointing, process cloning, node quarantine, error correction code (ECC), and so on. The generic live process migration technology, for example, may require a prior notice of less than 10 seconds according to the experiments conducted by [308], while similar OS virtualization technologies may call for much longer lead times (a warning of 13-24 seconds in general). To assist redundant execution during failures, Rezaei et al. [276] showed that node cloning requires less than 200 seconds. Gupta et al. [145] demonstrated 5-9% of future failures may be prevented when quarantining the blades/cabinets by stopping scheduling jobs on the nodes for a few hours after a failure is manifested.

There are three key points about predicting the lead-time analysis, which are noteworthy being mentioned as follows:

- Our model and baselines can all accurately forecast a variety of the lead times to different types of their associated failures. For example, the lead times (60 seconds, 80 seconds, 120 seconds, 160 seconds) of Cray supercomputer failures due to different errors of OS kernel panic, job scheduler Slurm, a hardware non-maskable interrupt (NMI), and Machine Check Exceptions, respectively. It is essential to predict lead times to failures accurately in HPC systems. As the lead time accuracy increases, recovery solutions can be scheduled at the optimal time (not sooner or later), minimizing downtime and reducing costs. For instance, if the lead-time predicted is too early, unnecessary recovery actions may be executed, resulting in unnecessary costs. On the other hand, if the lead-time prediction is too late, the system component (e.g., nodes) may fail before proactive measures are finished, resulting in job failure, unplanned downtime and lost productivity.

- Our model predicts some failures of Ranger and Cray HPC systems that occurred with a very short lead time (only 5 seconds) after the occurrence of their associated errors. Some types of segmentation faults and memory corruption failures are examples of this class. Thus, In these cases, the HPC system management control should first quarantine the corresponding failure-prone nodes for a couple of hours to avoid waste of the compute resources. Second, the system should use a recovery action that takes less time (e.g., the generic live process migration) or avoid triggering any correction/recovery actions as most failures have already occurred in this case.
- Our model predicts accurately many failures with relatively long lead times. For instance, the lead times to different node soft lockups failures in the TACC Ranger supercomputer are $\approx$ 100 minutes, $\approx$ 125 minutes, $\approx$ 300 minutes of their associated errors (general protection, page fault, loss of service connection by via Network Interface Device (NID)). In this case, our technique (Time Machine) can help the HPC system's administrator choose the best suitable lightweight error correction approach instead of an expensive solution. Also, it is practical to postpone triggering the recovery mechanism technique until a certain short period before the actual failure occurrence because most of these kinds of failures can be corrected themselves automatically.

7.6.6 Impact of Time Machine on Checkpoint-based Execution

We show how Time Machine can significantly reduce the execution overhead for the checkpointing-facilitated jobs, by leveraging a discrete event driven simulator [23], which can simulate the failure occurrence, job execution, check-pointing, and recovery from failures. Since the mean time between failures

(MTBF) could be hourly or even less than a hour according to [221], we set the MTBF in our experiment to be 1 hour. The failure events are generated following a Poisson process with an average failure interval of 1 hour. The checkpointing and recovery time overhead[2] are set to 60 seconds and 30 seconds, respectively. We evaluate two cases with different job workload lengths (6 hours and 12 hours). We compare four solutions as listed below:

- Ckpt (Young): This is a baseline solution, which adopts optimal checkpointing intervals calculated by Young's formula [332], as shown in Formula 7.6.

$$T_{interval} = \sqrt{2 \cdot \mu \cdot C} \tag{7.6}$$

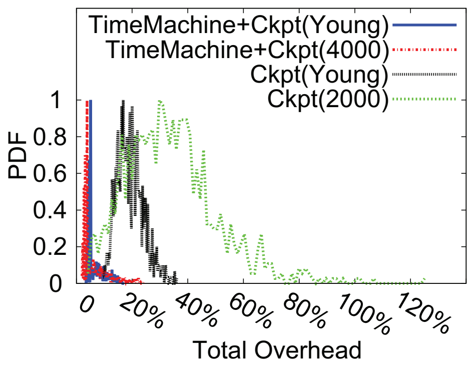
  where μ and C represent the MTBF and checkpointing time overhead, respectively. Whenever a failure occurs, the job would roll back to the latest checkpoint to restart the execution. According to our experimental setting, the optimal checkpointing interval is 657.3 seconds.

- Ckpt (2000): We evaluate the execution with a sub-optimal checkpoint interval (2k seconds), considering the practical situation with potentially inaccurate MTBF.

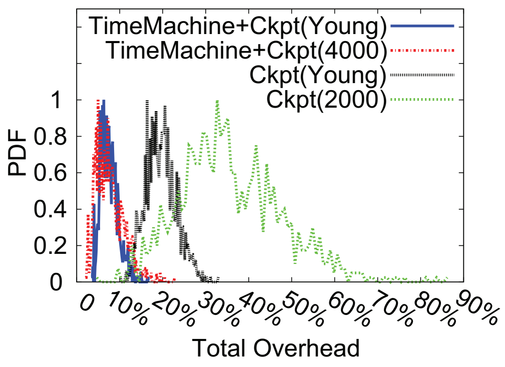
- TimeMachine + Ckpt(Young): We combine our failure lead time predictor (Time Machine) and Young's checkpoint intervals [332]. The lead times of the failures are predicted with a precision of 92% according to the worst case shown in Figure 7.6. That is, there is only one failed/missed prediction every ten failure events. For any failure predicted by Time Machine, we set the checkpoint ahead of the failure occurrence proactively. Since 92% failure events can be successfully predicted in time, the essential MTBF to the job execution is 10×MTBF, which leads to a different optimal checkpointing interval (2078 seconds).

- TimeMachine + Ckpt(4000): We evaluate TimeMachine+Ckpt in the case with inaccurate MTBF, which has an inferior setting of checkpoint interval (4k secs).

[2]Checkpointing time overhead means the time cost when setting/saving one checkpoint during the execution; recover time overhead means the time cost when restarting the application from a checkpoint.



(a) workload=21600



(b) workload=43200

Figure 7.7: Distribution of Total Execution Overhead

Table 7.5: Performance Breakdown of Simulation (job workload = 6 hours) based on 1000 tests (in seconds)

Solution	**stat**	**Load**	**Ckpt**	**Recv**	**RB loss**	**overhead**
TimeMachine + Ckpt(Young)	**min**	21600	660	30	2	3.2%
	avg	21600	992	154	432	7.3%
	max	21600	1642	162	2092	16.9%
Ckpt(Young)	**min**	21600	1920	30	48	9.3%
	avg	21600	1930	192	2007	19.1%
	max	21600	2120	450	5434	36.1%

Figure 7.7 shows the probability density function (PDF) of the job execution overhead ($=\frac{wallclock-job\_workload}{job\_workload}\times 100\%$), based on 1000 tests with different series of random failure events for each solution. We observe that combining TimeMachine with checkpointing can significantly reduce the execution overhead from ~20% down to ~7% in the execution environment. It is also worth noting that the TimeMachine+checkpointing is immune to inaccurate MTBF estimation (see red curve vs. blue curve), while the traditional checkpointing suffers from substantially higher overhead when the MTBF is inaccurate (see black curve vs. green curve). The key reason for the above result is that the checkpoints can be set more efficiently with Time Machine predictor, leading to significantly lower roll-back loss (abbreviated as *RB Loss*) during the execution, as demonstrated in Table 7.5.

Summary

In this chapter, we employ the transformer-decoder neural networks to build a novel real-time model called *Time Machine* to predict log events, failures, and their lead-times in HPC system's components (e.g, nodes). Time machine and three state-of-the-art techniques are evaluated on four HPC log data. Time Machine achieves considerably higher accuracy in predicting upcoming log events, failure location, and failure lead-time. As for log prediction, Time Machine obtains a Bleu and Rouge score of up to 0.79 and 0.77 respectively, much higher than the best of the three techniques (only 0.47 and 0.34 respectively). As for the failure location prediction, our Time Machine obtains the MCC and F1-score up to 0.80 and 0.87, respectively, which are much higher than that of the best one from the three state-of-the-arts (only 0.53 and 0.71 respectively). As for the lead-time prediction, Time Machine is also the best in class: specifically, its MCC and F1-score can reach up to 0.87 and 0.95, respectively. Time Machine is significantly faster than other approaches in both training (5.4$\sim$9.4$\times$ speed-up on average) and chain prediction (over 15$\times$ faster than the related works). We also demonstrate the significant reduction in execution overhead for the checkpointing-facilitated jobs through the utilisation of Time Machine, by leveraging a discrete event-driven simulator. Our model can trigger recovery solutions at the right places and right time with substantially reduced cost. Note that the Time Machine framework introduces a novel method to automatically (no need for labels) construct and augment a self-time annotated training dataset on sequential time-based (timestamps) raw data via automatic accumulative and iterative process. Also, Time Machine is the first model to convert the time prediction (a regression problem) into a self-annotated multi-class classification problem by predicting the class for the failure lead time.

Chapter 8

Conclusions and Future Work

Preface

This thesis is motivated by the necessity of creating dependable HPC distributed systems with minimised costly failure occurrences. As such, we explored several new AI-based models for error detection and failure prediction utilising the log data generated by the system components which capture their health states. The log messages are stored in a textual format, which motivates us to develop an NLP and AI based approaches to predict the failures' location (spatial) and the occurrence time (temporal), so that the failure management mechanism can be activated at the correct location and time. This chapter summarizes the key contributions made by each chapter in this thesis, points out the existing limitations, and discusses the possible future research directions.

8.1 Conclusion

In this thesis, we propose three novel machine/deep learning based methods, which target the error detection, failure location prediction and failure lead-time prediction, respectively.

Sentiment Analysis based Error Detection. Today's resource-hungry applications require to be executed on HPC environments such as supercomputers and datacentres. These HPC systems comprise hundreds of thousands of complicated hardware/software components, which are in charge of various data and execution services. These resident components together produce a large volume of log messages on a centralised log server. These log messages typically capture the health status of the various components (e.g., computer nodes, network). However, most large-scale systems logs are generated with non-labels, so manually classifying the massive number of unlabeled logs (i.e., millions) into faulty and healthy is infeasible. Hence, it is important to develop error detection approaches to identify faulty events from non-faulty ones for unlabelled system log data. As such, we addressed error detection in large-scale HPC systems generating unlabelled logs by proposing a novel sentiment analysis-based approach and leveraging log events' sentiment polarity and intensity to represent the system state. To accomplish this, we employed a ML model to build a sentiment lexicon. The sentiment scores of the lexicon elements allow log messages and the system components to be associated with sentiment polarity and intensity scores for error detection and fault localisation.

Failure location/lead-time prediction. Serious failures (e.g., node shutdowns) may occur in a large-scale system, because of the ever-increasing complexity of HPC systems and high-demanding applications. The consequence of the failures is tremendous: extreme computational overhead could be introduced, thus severely affecting application execution. Particularly, with breaking the barrier of exascale computing nowadays, such failures will occur at least as frequently as in the petascale era, thereby severely impacting the dependability of such systems to provide services. To mitigate the impact, failure prediction is becoming a pressing problem, especially when combined with a proactive recovery management process to mitigate the impact of such failures. The existing works on failure prediction, however, are still insufficient because relatively low accuracy and efficiency. For example, the state-of-the-art RNN-based failure prediction methods (e.g., LSTM [91], Bi-LSTM [131], and GRU [171]) suffer from non-trivial weaknesses: (i) long training time because of the absence of parallelization in recurrence learning, and (ii) the vanishing gradient problem with loss of earlier "memory", which result in limited accuracy. Thus, it is essential to develop online failure prediction techniques that can accurately and rapidly predict forthcoming failures with minimal computational overhead. As such, we introduced two novel solutions (Clairvoyant and Time Machine) for predicting failures and the lead time to these failures in HPC systems. Both rely on a self-supervised deep learning approach utilising a transformer-decoder, the self-attention mechanism, and parallel processing. Our solutions facilitate determining the appropriate proactive management mechanism with low cost, avoiding failure propagation, reducing the system maintenance cost, and preventing breakdowns/outages in large systems that provide critical services such as healthcare, emergency, education services, social media, etc.

The summary of this thesis is organised into two categories. First, we highlight the contents of the four introductory chapters, which include the introduction (Chapter 1), the background (Chapter 2), a survey of previous related work (Chapter 3) and the description of the cluster system used, the data and fault models (Chapter 4). Then, the main methodologies are presented in (Chapters 5 - 7), which discuss the key contributions, the limitations of our solutions, and future work.

8.1.1 Introductory Chapters

Chapters 1 to 4 of this thesis include the introduction, the background, the literature review of the related work, and the general description of the system used, the log data, and fault models, respectively. The following is a recap of the main points presented in the introductory chapters:

- **Chapter 1:** This chapter summarises the entire thesis. It defines the overarching objectives, motivation, research problem, as well as the proposed approach and contributions made in this thesis.
- **Chapter 2:** This chapter covers the pertinent background information regarding the machine and deep learning models we employed to build our error detection and failure prediction models in chapters 5, 6, and 7. Additionally, we highlight the state-of-the-art methods we used to compare our proposed solutions.
- **Chapter 3** This chapter introduces a taxonomy survey of the related works of the proposed automated log analysis research. The previous research is divided into four domains aiming to address the previous four log-based research perspectives. The first domain focus is log parsing which involves methods that aim to convert raw logs of HPC systems into

structured data. The second domain summarises the related techniques that utilised different approaches (e,g., machine and deep learning) for error detection, which helps to determine abnormal behaviour patterns and normal activity in log messages. The third domain discusses failure diagnosis techniques that aim to identify the underlying causes (i.e., errors and faults) of component failures in large-scale systems; finally, we survey the previous failure prediction studies that aim for early warnings of failures occurrence in HPC systems.

Chapter 4: This chapter describes the generic system model of cluster systems for which the approaches can be applied. We summarise five supercomputer clusters and their six log datasets used in our experiments. Furthermore, this chapter presents a description of the fault model in HPC systems. It also clarifies fundamental terms related to log messages and fault tolerance.

8.1.2 The Chapters Discussing Summary of Contributions, Limitations, and Future Work

Chapters (5 - 7) of this thesis present in detail our novel approaches proposed for error detection and failure prediction in large-scale distributed systems. The following is a summary of the main points presented in these chapters, their limitation, and future work.

- **Chapter 5 (GitHub [5]):** This chapter presents our sentiment analysis techniques on log events of complex HPC systems, which is an efficient solution to detect errors and identify failed components. In particular, we propose a novel approach for the automated generation of a reusable sentiment lexicon to support log analysis of different large-scale systems, since these large systems often share similar components. Our contributions are four-fold in this chapter (i) In terms of the system logs, we develop a machine learning-based approach by using stochastic gradient descent logistic regression to automatically build a sentiment lexicon. (ii) We develop an algorithm based on the sentiment features of the lexicon to detect whether a message corresponds to an error/failure or just informational, (iii) We develop an algorithm to determine whether a component is faulty in a given time window based on sentiment polarity scores. (iv) Using logs from three HPC systems of different vendors, we compare our solution to a broad range of state-of-the-art ML/deep techniques. Our results show that: (1) The learnt sentiment lexicon does consist of system developers' sentiments accurately, (2) Our technique successfully extracted developers' sentiment features with their weights from the source system with labelled log messages(i.e., IBM Blue Gene) to automatically construct lexicon items which can be used to detect the errors in the target systems with unlabelled log messages (i.e., Ranger and Lonestar4), with an average MCC score and f-score of 91% and 96% respectively, while the state of the art machine and deep learning model (LSTM) obtains only 67% and 84%.

Limitations and Future Work:

– The approach proposed in this chapter demonstrates the use of sentiment scores for transfer learning between different HPC datasets, where data is annotated with a varying amount of information (i.e. severity). However, our method is limited to classifying all events based on their severity attributes into two categories - faulty and non-faulty, which could be a future work. Specifically, we plan to classify the log messages into multiple classes based on multi-level severity messages (e.g., failure (fatal), error, warning, debug, information, etc.) in the real HPC systems logs. As a potential future work, our existing approach can be modified to extract developers' sentiment features with their weights from the source system log messages with multiple classes labels (multiple severity levels) to automatically construct sentiment lexicon items, which can classify log messages into multiple severity levels (classes) in the target systems with unlabelled logs.

– The approach proposed in this chapter is applied to logs generated from supercomputer clusters. As a potential future work, the approach can be extended to be applied to different systems such as datacenters logs or other components (e.g., switches, GPUs).

  - As a potential future work, the approach proposed in this chapter can be replaced by any modern NLP approach, especially transformer-based language models such as BERT, GPT-4, RoBERTa, etc. This can be achieved by fine-tuning the pre-trained models on large log data to learn the relationships between log messages with their assigned labels from source systems to classify log messages of the target systems with unlabelled logs.

- **Chapter 6 (GitHub [4]):** This chapter developed, Clairvoyant, a novel solution for predicting compute node failures in HPC systems. It relies on a self-supervised deep learning approach utilising a transformer-decoder and a self-attention mechanism. It predicts node failures by predicting a sequence of log events and then identifying if a failure is a part of that predicted sequence. The proposed approach has been evaluated with metrics such as Blue, Rouge, F1-score, and MCC. Clairvoyant compared with an HPC node-failure method called Desh [91] using datasets from two datasets of a supercomputer system. The results show that Clairvoyant is significantly more accurate in predicting node failures and faster than Desh due to observing long-range dependencies in the log events and a model that can be parallelised.

  Limitations:

  - The model presented in this chapter focuses on only one failure, soft lockup, that can be caused by several high-impact errors. Second and most importantly, the Clairvoyant framework aims to predict the existence of an upcoming failure only without predicting the lead time to the failure. Predicting the lead time of failures is very challenging because of many possible time moments the failure may occur in a very long/short upcoming sequence of log events. All these limitations are resolved by proposing the Time Machine framework to predict any possible types of failures and the failure lead-time accurately.

- **Chapter 7 (GitHub [23]):** This chapter presents Time Machine, a novel generative self-supervised model that predicts the failure status of the HPC systems. Two transformer-decoder based sequential stacks have been combined. One is to predict the next state of the system that can be used to generate a possible future log event sequence. If the sequence contains a log indicating a failure and its possible location, the second will be used to predict the lead time to failure. With an analogy to text prediction in NLP tasks, Time Machine uses log events as input, where each event is considered a word and a sequence of events is considered a sentence. It uses the self-attention-based language model as an estimator for the posterior probabilities to predict the probabilities of failure in HPC systems. Time Machine is evaluated on four real-world HPC clusters logs and compared against three state-of-the-art approaches. Time Machine achieves better results than the top-performing model option them: (i) Log events prediction: With scores of up to 0.79 and 0.77 in Bleu and Rouge, respectively, Time Machine significantly outperforms the best of the three techniques, whose scores are only 0.47 and 0.34. (ii) Failure Location: Time Machine achieves a MCC and F1-scores of up to 0.80 and 0.87, respectively, while the best scores of the three baselines are only 0.53 and 0.71, respectively. (iii) Failure lead time: Time Machine is also the best in class, with MCC and F1-scores of up to 0.87 and 0.95, respectively. (iv) Speed-up of training and prediction: Time Machine is significantly faster than other techniques in both training ($5.4 \sim 9.4\times$ speed-up on average) and chain prediction (over $15\times$ faster than the related works), thus, Time Machine is well-suited for real-time online failure prediction in HPC distributed systems.

Moreover, Time Machine framework presented a novel synthetic minority oversampling technique for online time-based tasks to construct the training instances in real-time from failure sequences. This method has never been discovered before in AI or other domains and can be generalised to other domains for time-based tasks (e.g., business, healthcare, booking business). Furthermore, the Time Machine is the first model to reduce/convert the time prediction problem (a regression problem) into a self-annotated multi-class classification problem by predicting the class for the failure lead time. This method can also be generalised to other domains for similar time-based tasks (e.g., business, healthcare, booking business).

Limitation and Future Work:

– Predicting failures in large cloud IT systems is a crucial area of research as supercomputer clusters, especially some large IT HPC machines can generate millions of logs (depending on the verbosity) per minute. The approach proposed in this chapter is limited to the logs generated from supercomputer clusters. As a potential future work, further research and improvements may be needed to enhance the model's generality beyond HPC logs, such as logs of datacenters and other cloud services. We expect our model will successfully predict those large systems also for two reasons. First, large IT systems (e.g., cloud systems) and supercomputer systems share several similarities, such as components (hardware (compute nodes/GPUs) and software), high-performance power to handle large amounts of data, high-speed networking, scalability, parallel processing, etc. Accordingly, while there may be some discrepancies in the logs of large IT systems and supercomputer clusters, their logs (whether log messages or metric data) share many similarities, which include logs format/structure, the information they contain, normal and failure patterns, type of metric data, and the challenges of preprocessing them. Therefore, we expect our model can be applied to predict large IT systems failures similar to large cluster supercomputers with high prediction accuracy. For example, some of the supercomputers which were involved in our study can be highly verbose and contain millions of messages per minute: this is similar to some large cloud systems. Our model overcomes these logs overhead and predicts their failures with high accuracy. Second, Time Machine outperforms the baseline models in our study, and some of the baselines have demonstrated their good performance in predicting large IT systems failures based on metric data. Hence, we expect our model will also exhibit its superiority in predicting large IT systems' failures.

– Motivated by the promising results of our solution, in future, we will explore other generative models such as Text-to-Text Transfer Transformer (T5), Bidirectional and Auto-Regressive Transformers (BART) for the prediction of HPC failures and lead-times. Moreover, we will explore different ratio for training and testing (e.g., 70/30) and different parameters such as self-attention layer number, head number, embedding size, etc., which may improve the prediction accuracy or at least maintains similar performance with a smaller number of parameters. For instance, Time Machine may achieve similar failure prediction accuracy with fewer self-attention layers or attention heads.

Bibliography

[1] Aurora supercomputer. https://www.intel.co.uk/content/www/uk/en/high-performance-computing/supercomputing/exascale-computing.html. (Accessed on 04/25/2023).

[2] Facebook lost about $65 million during hours-long outage. https://www.forbes.com/sites/abrambrown/2021/10/05/facebook-outage-lost-revenue/?sh=2076d190231a. (Accessed on 03/13/2023).

[3] Frontier. https://www.olcf.ornl.gov/frontier/. (Accessed on 04/25/2023).

[4] Github - khalid8alharthi/clairvoyant: Clairvoyant: A log-based transformer-decoder for failure prediction in large-scale systems. https://github.com/khalid8alharthi/Clairvoyant, . (Accessed on 05/29/2023).

[5] Github - khalid8alharthi/sentiment_analysis_model_for_errors_detection_in_large_scale
Sentiment analysis based error detection for large-scale systems. https://github.com/khalid8alharthi/Sentiment_Analysis_Model_For_Errors_Detection_In_Large_Scale_Systems, . (Accessed on 05/29/2023).

[6] Hyperion: Hpc market is stabilizing and headed to $50b by 2026. https://www.hpcwire.com/2022/05/30/hyperion-hpc-market-is-stabilizing-and-headed-to-50b-by-2026/. (Accessed on 04/25/2023).

[7] Keras: Deep learning for humans. https://keras.io/. (Accessed on 05/26/2023).

[8] Legacy computing program. https://www.tacc.utexas.edu/partnerships/legacy-computing-program/. (Accessed on 05/29/2023).

[9] Log management & analysis software made easy | logentries. https://logentries.com/, . (Accessed on 06/17/2022).

[10] Logstash: Collect, parse, transform logs | elastic. https://www.elastic.co/logstash/, . (Accessed on 06/22/2022).

[11] Mira | argonne leadership computing facility. https://www.alcf.anl.gov/alcf-resources/mira. (Accessed on 04/06/2023).

[12] Privacy error. https://sequencer.io/. (Accessed on 06/20/2022).

[13] Pytorch. https://pytorch.org/. (Accessed on 05/26/2023).

[14] Splunk | the data platform for the hybrid world. https://www.splunk.com/. (Accessed on 06/17/2022).

[15] Tacc lonestar 4 user guide - tacc user portal. https://portal.tacc.utexas.edu/archives/lonestar4. (Accessed on 04/06/2023).

[16] Technical specifications. https://www.lanl.gov/projects/trinity/specifications.php. (Accessed on 05/29/2023).

[17] Tensorflow. https://www.tensorflow.org/. (Accessed on 05/26/2023).

[18] Texas advanced computing center. https://www.tacc.utexas.edu/. (Accessed on 05/29/2023).

[19] Transformers in action: Attention is all you need | by soran ghaderi | towards data science. https://towardsdatascience.com/transformers-in-action-attention-is-all-you-need-ac10338a023a. (Accessed on 06/01/2023).

[20] Welcome to colaboratory - colaboratory. https://colab.research.google.com/. (Accessed on 05/26/2023).

[21] about-logs.051715.pdf. `https://portal.nersc.gov/project/m888/resilience/datasets/mutrino/about-logs.051715.pdf`. (Accessed on 04/07/2023).

[22] el capitan supercomputer at lawrence livermore national lab | hpe united kingdom. `https://www.hpe.com/uk/en/compute/hpc/cray/doe-el-capitan-press-release.html`. (Accessed on 04/25/2023).

[23] khalid8alharthi/time-machine: Code and datasets for a research paper accepted at dsn'23. `https://github.com/khalid8alharthi/Time-Machine`. (Accessed on 04/10/2023).

[24] pandas - python data analysis library. `https://pandas.pydata.org/`. (Accessed on 05/26/2023).

[25] Ieee standard for information technology–portable operating system interface (posix(tm)) base specifications, issue 7. *IEEE Std 1003.1-2017 (Revision of IEEE Std 1003.1-2008)*, pages 1–3951, 2018. doi: 10.1109/IEEESTD.2018.8277153.

[26] Muhammad Abbas, K Ali Memon, A Aleem Jamali, Saleemullah Memon, and Anees Ahmed. Multinomial naive bayes classification model for sentiment analysis. *IJCSNS Int. J. Comput. Sci. Netw. Secur*, 19(3):62, 2019.

[27] Dilmi Abdelbaqi. *Failure prediction in cloud environment using deep learning*. PhD thesis, University of M'sila, 2020.

[28] Sheikh Abujar, Abu Kaisar Mohammad Masum, SM Mazharul Hoque Chowdhury, Mahmudul Hasan, and Syed Akhter Hossain. Bengali text generation using bi-directional rnn. In *2019 10th International Conference on Computing, Communication and Networking Technologies (ICCCNT)*, pages 1–5. IEEE, 2019.

[29] Khansa Afifah, Intan Nurma Yulita, and Indra Sarathan. Sentiment analysis on telemedicine app reviews using xgboost classifier. In

2021 International Conference on Artificial Intelligence and Big Data Analytics, pages 22–27. IEEE, 2021.

[30] Bikash Agrawal, Tomasz Wiktorski, and Chunming Rong. Analyzing and predicting failure in hadoop clusters using distributed hidden markov model. In Weizhong Qiang, Xianghan Zheng, and Ching-Hsien Hsu, editors, *Cloud Computing and Big Data*, pages 232–246, Cham, 2015. Springer International Publishing. ISBN 978-3-319-28430-9.

[31] Fawaz S Al-Anzi. An effective hybrid stochastic gradient descent for classification of short text communication in e-learning environments. In *2022 8th International Conference on Control, Decision and Information Technologies (CoDIT)*, volume 1, pages 1096–1101. IEEE, 2022.

[32] Jay Alammar. The illustrated transformer. *The Illustrated Transformer–Jay Alammar–Visualizing Machine Learning One Concept at a Time*, 27, 2018.

[33] Jay Alammar. The illustrated gpt-2 (visualizing transformer language models). 2019.

[34] Khalid Ayedh Alharthi, Arshad Jhumka, Sheng Di, Franck Cappello, and Edward Chuah. Sentiment analysis based error detection for large-scale systems. In *2021 51st Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)*, pages 237–249. IEEE, 2021.

[35] Khalid Ayedh Alharthi, Arshad Jhumka, Sheng Di, and Franck Cappello. Clairvoyant: a log-based transformer-decoder for failure prediction in large-scale systems. In *Proceedings of the 36th ACM International Conference on Supercomputing*, pages 1–14, 2022.

[36] Khalid Ayedh Alharthi, Arshad Jhumka, Sheng Di, Lin Gui, Franck Cappello, and Simon McIntosh-Smith. Time machine: Generative real-time model for failure (and lead time) prediction in hpc sys-

tems. In *2023 53st Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)*. IEEE, 2023.

[37] Sara Ashour Aljuhani and Norah Saleh Alghamdi. A comparison of sentiment analysis methods on amazon reviews of mobile phones. *International Journal of Advanced Computer Science and Applications*, 10(6), 2019.

[38] Corville O Allen, Kevin B Haverlock, and Michael D Whitley. Sentiment analysis of data logs, January 3 2017. US Patent 9,536,200.

[39] Corville O Allen, Andrew R Freed, Scott N Gerard, and Dorian B Miller. Applying consistent log levels to application log messages, June 11 2019. US Patent 10,318,405.

[40] Sujit S Amin and Lata Ragha. Text generation and enhanced evaluation of metric for machine translation. In *Data Intelligence and Cognitive Informatics: Proceedings of ICDICI 2020*, pages 1–17. Springer, 2021.

[41] Peter J Angeline, Gregory M Saunders, and Jordan B Pollack. An evolutionary algorithm that constructs recurrent neural networks. *IEEE transactions on Neural Networks*, 5(1):54–65, 1994.

[42] Mihael Ankerst, Markus M Breunig, Hans-Peter Kriegel, and Jörg Sander. Optics: Ordering points to identify the clustering structure. *ACM Sigmod record*, 28(2):49–60, 1999.

[43] Ameen Abdullah Qaid Aqlan, B Manjula, and R Lakshman Naik. A study of sentiment analysis: concepts, techniques, and challenges. In *Proceedings of International Conference on Computational Intelligence and Data Engineering: Proceedings of ICCIDE 2018*, pages 147–162. Springer, 2019.

[44] Algirdas Avizienis, Jean-Claude Laprie, and Brian Randell. Fundamental concepts of dependability. *Department of Computing Science Technical Report Series*, 2001.

[45] Algirdas Avizienis, J-C Laprie, Brian Randell, and Carl Landwehr. Basic concepts and taxonomy of dependable and secure computing. *IEEE transactions on dependable and secure computing*, 1(1):11–33, 2004.

[46] Roberto Baldoni, Luca Montanari, and Marco Rizzuto. On-line failure prediction in safety-critical systems. *Future Generation Computer Systems*, 45:123–132, 2015.

[47] Chetan Bansal, Sundararajan Renganathan, Ashima Asudani, Olivier Midy, and Mathru Janakiraman. Decaf: Diagnosing and triaging performance issues in large-scale cloud services. In *Proceedings of the ACM/IEEE 42nd International Conference on Software Engineering: Software Engineering in Practice*, pages 201–210, 2020.

[48] Michael Barborak, Anton Dahbura, and Miroslaw Malek. The consensus problem in fault-tolerant computing. *ACM Computing Surveys (CSur)*, 25(2):171–220, 1993.

[49] Anne Benoit, Aurélien Cavelan, Valentin Le Fèvre, Yves Robert, and Hongyang Sun. Towards optimal multi-level checkpointing. *IEEE Transactions on Computers*, 66(7):1212–1226, 2016.

[50] Eduardo Berrocal, Li Yu, Sean Wallace, Michael E Papka, and Zhiling Lan. Exploring void search for fault detection on extreme scale systems. In *2014 IEEE International Conference on Cluster Computing (CLUSTER)*, pages 1–9. IEEE, 2014.

[51] Jasmine Bhaskar, K Sruthi, and Prema Nedungadi. Hybrid approach for emotion classification of audio conversation based on text and speech mining. *Procedia Computer Science*, 46:635–643, 2015.

[52] Marouane Birjali, Mohammed Kasri, and Abderrahim Beni-Hssane. A comprehensive survey on sentiment analysis: Approaches, challenges and trends. *Knowledge-Based Systems*, 226:107134, 2021.

[53] David Blei, Andrew Ng, and Michael Jordan. Latent dirichlet allocation. *Advances in neural information processing systems*, 14, 2001.

[54] Andrea Borghesi, Antonio Libri, Luca Benini, and Andrea Bartolini. Online anomaly detection in HPC systems. In *2019 IEEE International Conference on Artificial Intelligence Circuits and Systems (AICAS)*, pages 229–233. IEEE, 2019.

[55] Jim Brandt, Ann Gentile, Jackson Mayo, Philippe Pébay, Diana Roe, David Thompson, and Matthew Wong. Methodologies for advance warning of compute cluster problems via statistical analysis: A case study. In *Proceedings of the 2009 Workshop on Resiliency in High Performance*, Resilience '09, page 7–14, New York, NY, USA, 2009. Association for Computing Machinery. ISBN 9781605585932.

[56] Jim Brandt, Ann Gentile, Cindy Martin, Jason Repik, and Narate Taerat. New systems, new behaviors, new patterns: Monitoring insights from system standup. In *2015 IEEE International Conference on Cluster Computing*, pages 658–665. IEEE, 2015.

[57] Leo Breiman. Random forests. *Machine learning*, 45:5–32, 2001.

[58] Harsha Vardhana Krishna Sai Buddana, Surampudi Sai Kaushik, PVS Manogna, and Shijin Kumar PS. Word level lstm and recurrent neural network for automatic text generation. In *2021 International Conference on Computer Communication and Informatics (ICCCI)*, pages 1–4. IEEE, 2021.

[59] Sathya Bursic, Vittorio Cuculo, and Alessandro D'Amelio. Anomaly detection from log files using unsupervised deep learning. In *International Symposium on Formal Methods*, pages 200–207. Springer, 2019.

[60] Carlos A. C. Rincon, Jehan-François Pâris, Ricardo Vilalta, Albert M. K. Cheng, and Darrell D. E. Long. Disk failure prediction in heterogeneous environments. In *2017 International Symposium*

on Performance Evaluation of Computer and Telecommunication Systems (SPECTS), pages 1–7, 2017. doi: 10.23919/SPECTS.2017.8046776.

[61] Ling Cai, Krzysztof Janowicz, Gengchen Mai, Bo Yan, and Rui Zhu. Traffic transformer: Capturing the continuity and periodicity of time series for traffic forecasting. *Transactions in GIS*, 24(3): 736–755, 2020.

[62] Thanyalak Chalermarrewong, Tiranee Achalakul, and Simon Chong Wee See. Failure prediction of data centers using time series and fault tree analysis. In *2012 IEEE 18th International Conference on Parallel and Distributed Systems*, pages 794–799, 2012. doi: 10.1109/ICPADS.2012.129.

[63] Thanyalak Chalermarrewong, Tiranee Achalakul, and Simon Chong Wee See. Failure prediction of data centers using time series and fault tree analysis. In *2012 IEEE 18th International Conference on Parallel and Distributed Systems*, pages 794–799. IEEE, 2012.

[64] Purimpat Cheansunan and Phond Phunchongharn. Detecting anomalous events on distributed systems using convolutional neural networks. In *2019 IEEE 10th International Conference on Awareness Science and Technology (iCAST)*, pages 1–5. IEEE, 2019.

[65] An Ran Chen. An empirical study on leveraging logs for debugging production failures. In *2019 IEEE/ACM 41st International Conference on Software Engineering: Companion Proceedings (ICSE-Companion)*, pages 126–128. IEEE, 2019.

[66] Mike Chen, Alice X Zheng, Jim Lloyd, Michael I Jordan, and Eric Brewer. Failure diagnosis using decision trees. In *International Conference on Autonomic Computing, 2004. Proceedings.*, pages 36–43. IEEE, 2004.

[67] Tianqi Chen and Carlos Guestrin. Xgboost: A scalable tree boosting system. In *Proceedings of the 22nd acm sigkdd international*

conference on knowledge discovery and data mining, pages 785–794, 2016.

[68] Xin Chen, Charng-Da Lu, and Karthik Pattabiraman. Failure analysis of jobs in compute clouds: A google cluster case study. In *2014 IEEE 25th International Symposium on Software Reliability Engineering*, pages 167–177. IEEE, 2014.

[69] Yiwei Cheng, Haiping Zhu, Jun Wu, and Xinyu Shao. Machine health monitoring using adaptive kernel spectral clustering and deep long short-term memory recurrent neural networks. *IEEE Transactions on Industrial Informatics*, 15(2):987–997, 2018.

[70] Ilia Chetviorkin and Natalia Loukachevitch. Extraction of russian sentiment lexicon for product meta-domain. In *Proceedings of COLING 2012*, pages 593–610, 2012.

[71] Davide Chicco and Giuseppe Jurman. The advantages of the Matthews correlation coefficient (MCC) over f1 score and accuracy in binary classification evaluation. *BMC Genomics*, 21, 2020.

[72] Kyunghyun Cho, Bart Van Merriënboer, Caglar Gulcehre, Dzmitry Bahdanau, Fethi Bougares, Holger Schwenk, and Yoshua Bengio. Learning phrase representations using rnn encoder-decoder for statistical machine translation. *arXiv preprint arXiv:1406.1078*, 2014.

[73] Edward Chuah. *Features correlation-based workflows for high-performance computing systems diagnosis.* PhD thesis, University of Warwick, 2020.

[74] Edward Chuah, Shyh-hao Kuo, Paul Hiew, William-Chandra Tjhi, Gary Lee, John Hammond, Marek T Michalewicz, Terence Hung, and James C Browne. Diagnosing the root-causes of failures from cluster log files. In *2010 International Conference on High Performance Computing*, pages 1–10. IEEE, 2010.

[75] Edward Chuah, Arshad Jhumka, Sai Narasimhamurthy, John Hammond, James C Browne, and Bill Barth. Linking resource usage anomalies with system failures from cluster log data. In *2013 IEEE 32nd International Symposium on Reliable Distributed Systems*, pages 111–120. IEEE, 2013.

[76] Edward Chuah, Arshad Jhumka, James C Browne, Bill Barth, and Sai Narasimhamurthy. Insights into the diagnosis of system failures from cluster message logs. In *2015 11th European Dependable Computing Conference (EDCC)*, pages 225–232. IEEE, 2015.

[77] Edward Chuah, Arshad Jhumka, Samantha Alt, Daniel Balouek-Thomert, James C Browne, and Manish Parashar. Towards comprehensive dependability-driven resource use and message log-analysis for HPC systems diagnosis. *Journal of Parallel and Distributed Computing*, 132:95–112, 2019.

[78] Edward Chuah, Arshad Jhumka, Samantha Alt, Juan J Villalobos, Joshua Fryman, William Barth, and Manish Parashar. Using resource use data and system logs for HPC system error propagation and recovery diagnosis. In *2019 IEEE Intl Conf on Parallel & Distributed Processing with Applications, Big Data & Cloud Computing, Sustainable Computing & Communications, Social Computing & Networking (ISPA/BDCloud/SocialCom/SustainCom)*, pages 458–467. IEEE, 2019.

[79] Edward ChuahM, Arshad Jhumka, Samantha Alt, R Todd Evans, and Neeraj Suri. Failure diagnosis for cluster systems using partial correlations. In *2021 IEEE Intl Conf on Parallel & Distributed Processing with Applications, Big Data & Cloud Computing, Sustainable Computing & Communications, Social Computing & Networking (ISPA/BDCloud/SocialCom/SustainCom)*, pages 1091–1101. IEEE, 2021.

[80] Junyoung Chung, Caglar Gulcehre, KyungHyun Cho, and Yoshua

Bengio. Empirical evaluation of gated recurrent neural networks on sequence modeling. *arXiv preprint arXiv:1412.3555*, 2014.

[81] Marcello Cinque, Raffaele Della Corte, Vincenzo Moscato, and Giancarlo Sperlí. A graph-based approach to detect unexplained sequences in a log. *Expert Systems with Applications*, 171:114556, 2021.

[82] Carlos H.A. Costa, Yoonho Park, Bryan S. Rosenburg, Chen-Yong Cher, and Kyung Dong Ryu. A system software approach to proactive memory-error avoidance. In *SC '14: Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, pages 707–718, 2014. doi: 10.1109/SC.2014.63.

[83] Novelty Octaviani Faomasi Daeli and Adiwijaya Adiwijaya. Sentiment analysis on movie reviews using information gain and k-nearest neighbor. *Journal of Data Science and Its Applications*, 3(1):1–7, 2020.

[84] Hetong Dai, Heng Li, Che Shao Chen, Weiyi Shang, and Tse-Hsun Chen. Logram: Efficient log parsing using n-gram dictionaries. *IEEE Transactions on Software Engineering*, 2020.

[85] Dinh Dai Vu, Xuan Tuong Vu, and Younghan Kim. Deep learning-based fault prediction in cloud system. In *2021 International Conference on Information and Communication Technology Convergence (ICTC)*, pages 1826–1829. IEEE, 2021.

[86] R Damarta, A Hidayat, and AS Abdullah. The application of k-nearest neighbors classifier for sentiment analysis of pt pln (persero) twitter account service quality. In *Journal of Physics: Conference Series*, volume 1722, page 012002. IOP Publishing, 2021.

[87] 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)*, pages 1092–1101, 2020.

[88] Anwesha Das and Frank Mueller. Holistic root cause analysis of node failures in production HPC. *SC Poster Session*, 2018.

[89] Anwesha Das, Frank Mueller, Paul Hargrove, Eric Roman, and Scott Baden. Doomsday: predicting which node will fail when on supercomputers. In *SC18: International Conference for High Performance Computing, Networking, Storage and Analysis*, pages 108–121. IEEE, 2018.

[90] Anwesha Das, Frank Mueller, Paul Hargrove, Eric Roman, and Scott Baden. Doomsday: Predicting which node will fail when on supercomputers. In *SC18: International Conference for High Performance Computing, Networking, Storage and Analysis*, pages 108–121, 2018. doi: 10.1109/SC.2018.00012.

[91] Anwesha Das, Frank Mueller, Charles Siegel, and Abhinav 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*, pages 40–51, 2018.

[92] Anwesha Das, Frank Mueller, and Barry Rountree. Aarohi: Making real-time node failure prediction feasible. In *2020 IEEE International Parallel and Distributed Processing Symposium (IPDPS)*, pages 1092–1101, 2020. doi: 10.1109/IPDPS47924.2020.00115.

[93] Anwesha Das, Frank Mueller, and Barry Rountree. Systemic assessment of node failures in hpc production platforms. In *2021 IEEE International Parallel and Distributed Processing Symposium (IPDPS)*, pages 267–276. IEEE, 2021.

[94] Kia Dashtipour, Mandar Gogate, Jingpeng Li, Fengling Jiang, Bin Kong, and Amir Hussain. A hybrid persian sentiment analysis framework: Integrating dependency grammar based rules and deep neural networks. *Neurocomputing*, 380:1–10, 2020.

[95] Kalyanmoy Deb, Amrit Pratap, Sameer Agarwal, and TAMT Meyarivan. A fast and elitist multiobjective genetic algorithm: Nsga-

ii. *IEEE transactions on evolutionary computation*, 6(2):182–197, 2002.

[96] Mitali Desai and Mayuri A Mehta. Techniques for sentiment analysis of twitter data: A comprehensive survey. In *2016 International Conference on Computing, Communication and Automation (ICCCA)*, pages 149–154. IEEE, 2016.

[97] S. Di, H. Guo, R. Gupta, E. R. Pershey, M. Snir, and F. Cappello. Exploring properties and correlations of fatal events in a large-scale hpc system. *IEEE Transactions on Parallel and Distributed Systems*, 30(2):361–374, 2019. doi: 10.1109/TPDS.2018.2864184.

[98] S. Di, H. Guo, E. Pershey, M. Snir, and F. Cappello. Characterizing and understanding HPC job failures over the 2k-day life of IBM bluegene/q system. In *2019 49th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)*, pages 473–484, 2019.

[99] Sheng Di, Rinku Gupta, Marc Snir, Eric Pershey, and Franck Cappello. Logaider: A tool for mining potential correlations of HPC log events. In *2017 17th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGRID)*, pages 442–451. IEEE, 2017.

[100] Sheng Di, Hanqi Guo, Eric Pershey, Marc Snir, and Franck Cappello. Characterizing and understanding hpc job failures over the 2k-day life of ibm bluegene/q system. In *2019 49th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)*, pages 473–484, 2019. doi: 10.1109/DSN.2019.00055.

[101] Karim Djemame and Hamish Carr. Exascale computing deployment challenges. In *Economics of Grids, Clouds, Systems, and Services: 17th International Conference, GECON 2020, Izola, Slovenia, September 15–17, 2020, Revised Selected Papers 17*, pages 211–216. Springer, 2020.

[102] John Daly DOD, Bill Harrod, Thuc Hoang, Lucy Nowell, Bob Adolf, Shekhar Borkar, Nathan DeBardeleben, Mike Heroux, David Rogers, Vivek Sarkar, et al. Inter-agency workshop on hpc resilience at extreme scale. `http://www.rdadolf.com/papers/daly2012report.pdf`, 2012.

[103] Chenhe Dong, Yinghui Li, Haifan Gong, Miaoxin Chen, Junxin Li, Ying Shen, and Min Yang. A survey of natural language generation. *ACM Computing Surveys*, 55(8):1–38, 2022.

[104] Jack Dongarra, Thomas Herault, and Yves Robert. *Fault Tolerance Techniques for High-Performance Computing.* 05 2015. ISBN 978-3-319-20942-5. doi: 10.1007/978-3-319-20943-2_1.

[105] Fernando Dione dos Santos Lima, Gabriel Maia Rocha Amaral, Lucas Goncalves de Moura Leite, João Paulo Pordeus Gomes, and Javam de Castro Machado. Predicting failures in hard drives with lstm networks. In *2017 Brazilian Conference on Intelligent Systems (BRACIS)*, pages 222–227. IEEE, 2017.

[106] Min Du and Feifei Li. Spell: Streaming parsing of system event logs. In *2016 IEEE 16th International Conference on Data Mining (ICDM)*, pages 859–864. IEEE, 2016.

[107] Min Du, Feifei Li, Guineng Zheng, and Vivek 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*, pages 1285–1298, 2017.

[108] Qingfeng Du, Liang Zhao, Jincheng Xu, Yongqi Han, and Shuangli Zhang. Log-based anomaly detection with multi-head scaled dot-product attention mechanism. In *International Conference on Database and Expert Systems Applications*, pages 335–347. Springer, 2021.

[109] Rehab M Duwairi and Islam Qarqaz. Arabic sentiment analysis using supervised classification. In *2014 International Conference on Future Internet of Things and Cloud*, pages 579–583. IEEE, 2014.

[110] Abhishek Dwaraki, Shachi Kumary, and Tilman Wolf. Automated event identification from system logs using natural language processing. In *2020 International Conference on Computing, Networking and Communications (ICNC)*, pages 209–215. IEEE, 2020.

[111] Nosayba El-Sayed, Hongyu Zhu, and Bianca 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)*, pages 1333–1344, 2017. doi: 10.1109/ICDCS.2017.317.

[112] Oracle/Sun Grid Engine. `https://www.oracle.com/it-infrastructure/`. Online.

[113] R Todd Evans, James C Browne, and William L Barth. Understanding application and system performance through system-wide monitoring. In *2016 IEEE International Parallel and Distributed Processing Symposium Workshops (IPDPSW)*, pages 1702–1710. IEEE, 2016.

[114] Arif Abdurrahman Farisi, Yuliant Sibaroni, and Said Al Faraby. Sentiment analysis on hotel reviews using multinomial naïve bayes classifier. In *Journal of Physics: Conference Series*, volume 1192, page 012024. IOP Publishing, 2019.

[115] Mostafa Farshchi, Jean-Guy Schneider, Ingo Weber, and John Grundy. Experience report: Anomaly detection of cloud application operations using log and cloud metric correlation analysis. In *2015 IEEE 26th international symposium on software reliability engineering (ISSRE)*, pages 24–34. IEEE, 2015.

[116] M Ali Fauzi. Random forest approach fo sentiment analysis in indonesian. *Indones. J. Electr. Eng. Comput. Sci*, 12:46–50, 2018.

[117] Ilenia Fronza, Alberto Sillitti, Giancarlo Succi, Mikko Terho, and Jelena Vlasenko. Failure prediction based on log files using random

indexing and support vector machines. *Journal of Systems and Software*, 86(1):2–11, 2013.

[118] Qiang Fu, Jian-Guang Lou, Yi Wang, and Jiang Li. Execution anomaly detection in distributed systems through unstructured log analysis. In *2009 ninth IEEE international conference on data mining*, pages 149–158. IEEE, 2009.

[119] Xiaoyu Fu, Rui Ren, Jianfeng Zhan, Wei Zhou, Zhen Jia, and Gang Lu. Logmaster: Mining event correlations in logs of large-scale cluster systems. In *2012 IEEE 31st Symposium on Reliable Distributed Systems*, pages 71–80. IEEE, 2012.

[120] Xiaoyu Fu, Rui Ren, Sally A. McKee, Jianfeng Zhan, and Ninghui Sun. Digging deeper into cluster system logs for failure prediction and root cause diagnosis. In *2014 IEEE International Conference on Cluster Computing (CLUSTER)*, pages 103–112, 2014. doi: 10.1109/CLUSTER.2014.6968768.

[121] Xiaoyu Fu, Rui Ren, Sally A McKee, Jianfeng Zhan, and Ninghui Sun. Digging deeper into cluster system logs for failure prediction and root cause diagnosis. In *2014 IEEE International Conference on Cluster Computing (CLUSTER)*, pages 103–112. IEEE, 2014.

[122] Zhe Fu, Shijie Zhou, and Jun Li. bitfa: A novel data structure for fast and update-friendly regular expression matching. In *Proceedings of the SIGCOMM Posters and Demos*, pages 130–132. 2017.

[123] Errin W Fulp, Glenn A Fink, and Jereme N Haack. Predicting computer system failures using support vector machines. *WASL*, 8: 5–5, 2008.

[124] Philip Gage. A new algorithm for data compression. *C Users Journal*, 12(2):23–38, 1994.

[125] Ana Gainaru, Franck Cappello, and William Kramer. Taming of the shrew: Modeling the normal and faulty behaviour of large-

scale HPC systems. In *2012 IEEE 26th International Parallel and Distributed Processing Symposium*, pages 1168–1179. IEEE, 2012.

[126] Ana Gainaru, Franck Cappello, Marc Snir, and William 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*, pages 1–11, 2012. doi: 10.1109/SC.2012.57.

[127] Ana Gainaru, Franck Cappello, Marc Snir, and William 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*, pages 1–11. IEEE, 2012.

[128] Ana Gainaru, Mohamed-Slim Bouguerra, Franck Cappello, Marc Snir, and William T. C. Kramer. Navigating the blue waters : Online failure prediction in the petascale era. 2013.

[129] Ana Gainaru, Franck Cappello, Marc Snir, and William Kramer. Failure prediction for hpc systems and applications: Current situation and open issues. *The International journal of high performance computing applications*, 27(3):273–282, 2013.

[130] Sandipan Ganguly, Ashish Consul, Ali Khan, Brian Bussone, Jacqueline Richards, and Alejandro 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)*, pages 105–116, 2016. doi: 10.1109/BigDataService.2016.10.

[131] Jiechao Gao, Haoyu Wang, and Haiying Shen. Task failure prediction in cloud data centers using deep learning. *IEEE Transactions on Services Computing*, 2020.

[132] Peter Garraghan, Ismael Solis Moreno, Paul Townend, and Jie Xu. An analysis of failure-related energy waste in a large-scale cloud

environment. *IEEE Transactions on Emerging topics in Computing*, 2(2):166–180, 2014.

[133] Babacar Gaye, Dezheng Zhang, and Aziguli Wulamu. Sentiment classification for employees reviews using regression vector-stochastic gradient descent classifier (rv-sgdc). *PeerJ Computer Science*, 7:e712, 2021.

[134] Felix A Gers, Jürgen Schmidhuber, and Fred Cummins. Learning to forget: Continual prediction with lstm. *Neural computation*, 12 (10):2451–2471, 2000.

[135] Markus Goldstein and Andreas Dengel. Histogram-based outlier score (hbos): A fast unsupervised anomaly detection algorithm. *KI-2012: poster and demo track*, 1:59–63, 2012.

[136] Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial networks. *Communications of the ACM*, 63 (11):139–144, 2014.

[137] Arepalli Peda Gopi, R Naga Sravana Jyothi, V Lakshman Narayana, and K Satya Sandeep. Classification of tweets data based on polarity using improved rbf kernel of svm. *International Journal of Information Technology*, pages 1–16, 2020.

[138] Jim Gray and Andreas Reuter. *Transaction processing: concepts and techniques*. Elsevier, 1992.

[139] Jiexing Gu, Ziming Zheng, Zhiling Lan, John White, Eva Hocks, and Byung-Hoon Park. Dynamic meta-learning for failure prediction in large-scale systems: A case study. In *2008 37th International Conference on Parallel Processing*, pages 157–164, 2008. doi: 10.1109/ICPP.2008.17.

[140] Qiang Guan, Ziming Zhang, and Song Fu. Proactive failure management by integrated unsupervised and semi-supervised learning for dependable cloud systems. In *2011 Sixth International Conference*

on Availability, Reliability and Security, pages 83–90, 2011. doi: 10.1109/ARES.2011.20.

[141] Qiang Guan, Ziming Zhang, and Song Fu. Ensemble of bayesian predictors and decision trees for proactive failure management in cloud computing systems. *J. Commun.*, 7(1):52–61, 2012.

[142] Lin Gui, Yu Zhou, Ruifeng Xu, Yulan He, and Qin Lu. Learning representations from heterogeneous network for sentiment classification of product reviews. *Knowledge-Based Systems*, 124:34–45, 2017.

[143] Luanzheng Guo, Dong Li, Ignacio Laguna, and Martin Schulz. Fliptracker: Understanding natural error resilience in hpc applications. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage, and Analysis*, SC '18. IEEE Press, 2018.

[144] Yicheng Guo, Yujin Wen, Congwei Jiang, Yixin Lian, and Yi Wan. Detecting log anomalies with multi-head attention (lama). *arXiv preprint arXiv:2101.02392*, 2021.

[145] Saurabh Gupta, Devesh Tiwari, Christopher Jantzi, James Rogers, and Don Maxwell. Understanding and exploiting spatial properties of system failures on extreme-scale hpc systems. In *2015 45th Annual IEEE/IFIP International Conference on Dependable Systems and Networks*, pages 37–44. IEEE, 2015.

[146] Nentawe Gurumdimma and Arshad Jhumka. Detection of recovery patterns in cluster systems using resource usage data. In *2017 IEEE 22nd Pacific Rim International Symposium on Dependable Computing (PRDC)*, pages 58–67. IEEE, 2017.

[147] Nentawe Gurumdimma, Arshad Jhumka, Maria Liakata, Edward Chuah, and James Browne. Towards detecting patterns in failure logs of large-scale distributed systems. In *2015 IEEE International Parallel and Distributed Processing Symposium Workshop*, pages 1052–1061. IEEE, 2015.

[148] Nentawe Gurumdimma, Arshad Jhumka, Maria Liakata, Edward Chuah, and James Browne. Crude: Combining resource usage data and error logs for accurate error detection in large-scale distributed systems. In *2016 IEEE 35th Symposium on Reliable Distributed Systems (SRDS)*, pages 51–60. IEEE, 2016.

[149] Nentawe Gurumdimma, Gideon Dadik Bibu, Desmond Bala Bisandu, and Mammuan Titus Alams. Identifying recovery patterns from resource usage data of cluster systems. *Science World Journal*, 13(4):87–94, 2018.

[150] Nentawe Y Gurumdimma. *Towards efficient error detection in large-scale HPC systems.* PhD thesis, University of Warwick, 2016.

[151] Salman Habib, Adrian Pope, Hal Finkel, Nicholas Frontiere, Katrin Heitmann, David Daniel, Patricia Fasel, Vitali Morozov, George Zagaris, Tom Peterka, et al. Hacc: Simulating sky surveys on state-of-the-art supercomputing architectures. *New Astronomy*, 42: 49–65, 2016.

[152] John L Hammond, Tommy Minyard, and Jim Browne. End-to-end framework for fault management for open source clusters: Ranger. In *Proceedings of the 2010 TeraGrid Conference*, pages 1–6, 2010.

[153] Hossein Hamooni, Biplob Debnath, Jianwu Xu, Hui Zhang, Guofei Jiang, and Abdullah Mueen. Logmine: Fast pattern recognition for log analytics. In *Proceedings of the 25th ACM International on Conference on Information and Knowledge Management*, pages 1573–1582, 2016.

[154] Hongyu Han, Jianpei Zhang, Jing Yang, Yiran Shen, and Yongshi Zhang. Generate domain-specific sentiment lexicon for review sentiment analysis. *Multimedia Tools and Applications*, 77:21265–21280, 2018.

[155] Jiawei Han, Jian Pei, and Yiwen Yin. Mining frequent patterns without candidate generation. *ACM sigmod record*, 29(2):1–12, 2000.

[156] Tanvi Hardeniya and Dilipkumar A Borikar. Dictionary based approach to sentiment analysis-a review. *International Journal of Advanced Engineering, Management and Science*, 2(5):239438, 2016.

[157] Louise Harding, Fabien Wernli, and Frédéric Suter. Sequence-rtg: Efficient and production-ready pattern mining in system log messages. In *2021 IEEE International Conference on Cluster Computing (CLUSTER)*, pages 623–631. IEEE, 2021.

[158] Shayan Hashemi and Mika Mäntylä. Onelog: Towards end-to-end training in software log anomaly detection. *arXiv preprint arXiv:2104.07324*, 2021.

[159] Shayan Hashemi and Mika Mäntylä. Sialog: detecting anomalies in software execution logs using the siamese network. *Automated Software Engineering*, 29(2):1–28, 2022.

[160] Pinjia He, Jieming Zhu, Zibin Zheng, and Michael R Lyu. Drain: An online log parsing approach with fixed depth tree. In *2017 IEEE International Conference on Web Services (ICWS)*, pages 33–40. IEEE, 2017.

[161] Shilin He, Jieming Zhu, Pinjia He, and Michael R Lyu. Experience report: System log analysis for anomaly detection. In *2016 IEEE 27th International Symposium on Software Reliability Engineering (ISSRE)*, pages 207–218. IEEE, 2016.

[162] Shilin He, Qingwei Lin, Jian-Guang Lou, Hongyu Zhang, Michael R Lyu, and Dongmei Zhang. Identifying impactful service system problems via log analysis. In *Proceedings of the 2018 26th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, pages 60–70, 2018.

[163] Hani D Hejazi, Ahmed A Khamees, Muhammad Alshurideh, and Said A Salloum. Arabic text generation: Deep learning for poetry

synthesis. In *Advanced Machine Learning Technologies and Applications: Proceedings of AMLTA 2021*, pages 104–116. Springer, 2021.

[164] ChukFong Ho, Masrah Azrifah Azmi Murad, Shyamala Doraisamy, and Rabiah Abdul Kadir. Extracting lexical and phrasal paraphrases: a review of the literature. *Artificial Intelligence Review*, 42:851–894, 2014.

[165] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural computation*, 9(8):1735–1780, 1997.

[166] Soudamini Hota and Sudhir Pathak. Knn classifier based approach for multi-class sentiment analysis of twitter data. *Int. J. Eng. Technol*, 7(3):1372–1375, 2018.

[167] Lihan Hu, Lixin Han, Zhenyuan Xu, Tianming Jiang, and Huijun Qi. A disk failure prediction method based on lstm network due to its individual specificity. *Procedia Computer Science*, 176:791–799, 2020.

[168] Yonghua Huo, Jing Dong, Zhongdi Ge, Ping Xie, Na An, and Yang Yang. Iwapriori: An association rule mining and self-updating method based on weighted increment. In *2020 21st Asia-Pacific Network Operations and Management Symposium (APNOMS)*, pages 167–172. IEEE, 2020.

[169] Clayton J Hutto and Eric Gilbert. Vader: A parsimonious rule-based model for sentiment analysis of social media text. In *Eighth international AAAI conference on weblogs and social media*, 2014.

[170] Touseef Iqbal and Shaima Qureshi. The survey: Text generation models in deep learning. *Journal of King Saud University-Computer and Information Sciences*, 34(6):2515–2528, 2022.

[171] Mohammad Saiful Islam and Andriy Miranskyy. Anomaly detection in cloud components. In *2020 IEEE 13th International Conference on Cloud Computing (CLOUD)*, pages 1–3. IEEE, 2020.

[172] Tariqul Islam and Dakshnamoorthy Manivannan. Predicting application failure in cloud: A machine learning approach. In *2017 IEEE International Conference on Cognitive Computing (ICCC)*, pages 24–31, 2017. doi: 10.1109/IEEE.ICCC.2017.11.

[173] Tariqul Islam and Dakshnamoorthy Manivannan. Predicting application failure in cloud: A machine learning approach. In *2017 IEEE International Conference on Cognitive Computing (ICCC)*, pages 24–31. IEEE, 2017.

[174] David Jauk, Dai Yang, and Martin 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*, pages 1–13, 2019.

[175] Arshad Jhumka. *Automated design of efficient fail-safe fault tolerance*. PhD thesis, Technische Universität, 2004.

[176] Arshad Jhumka. Dependability in service-oriented computing. *Agent-Based Service-Oriented Computing*, pages 141–160, 2010.

[177] Arshad Jhumka and Matt Leeke. Issues on the design of efficient fail-safe fault tolerance. In *2009 20th International Symposium on Software Reliability Engineering*, pages 155–164. IEEE, 2009.

[178] Arshad Jhumka and Neeraj Suri. Designing efficient fail-safe multitolerant systems. In *Formal Techniques for Networked and Distributed Systems-FORTE 2005: 25th IFIP WG 6.1 International Conference, Taipei, Taiwan, October 2-5, 2005. Proceedings 25*, pages 428–442. Springer, 2005.

[179] Arshad Jhumka, Martin Hiller, and Neeraj Suri. Assessing intermodular error propagation in distributed software. In *Proceedings 20th IEEE Symposium on Reliable Distributed Systems*, pages 152–161. IEEE, 2001.

[180] Arshad Jhumka, Felix C Gärtner, Christof Fetzer, and Neeraj Suri. On systematic design of fast and perfect detectors. Technical report, 2002.

[181] Arshad Jhumka, Martin Hiller, and Neeraj Suri. Component-based synthesis of dependable embedded software. In *Formal Techniques in Real-Time and Fault-Tolerant Systems: 7th International Symposium, FTRTFT 2002 Co-sponsored by IFIP WG 2.2 Oldenburg, Germany, September 9–12, 2002 Proceedings 7*, pages 111–128. Springer, 2002.

[182] Arshad Jhumka, Neeraj Suri, and Martin Hiller. A framework for the design and validation of efficient fail-safe fault-tolerant programs. In *Software and Compilers for Embedded Systems: 7th International Workshop, SCOPES 2003, Vienna, Austria, September 24-26, 2003. Proceedings 7*, pages 182–197. Springer, 2003.

[183] Arshad Jhumka, Martin Hiller, and Neeraj Suri. An approach for designing and assessing detectors for dependable component-based systems. In *Eighth IEEE International Symposium on High Assurance Systems Engineering, 2004. Proceedings.*, pages 69–78. IEEE Computer Society, 2004.

[184] Arshad Jhumka, Felix Freiling, Christof Fetzer, and Neeraj Suri. An approach to synthesise safe systems. *International Journal of Security and Networks*, 1(1-2):62–74, 2006.

[185] Zhen Ming Jiang, Ahmed E Hassan, Parminder Flora, and Gilbert Hamann. Abstracting execution logs to execution events for enterprise applications (short paper). In *2008 The Eighth International Conference on Quality Software*, pages 181–186. IEEE, 2008.

[186] Sian Jin, Pascal Grosset, Christopher M Biwer, Jesus Pulido, Jiannan Tian, Dingwen Tao, and James Ahrens. Understanding gpu-based lossy compression for extreme-scale cosmological simulations. In *2020 IEEE International Parallel and Distributed Processing Symposium (IPDPS)*, pages 105–115. IEEE, 2020.

[187] Armand Joulin, Edouard Grave, Piotr Bojanowski, Matthijs Douze, Hérve Jégou, and Tomas Mikolov. Fasttext. zip: Compressing text classification models. *arXiv preprint arXiv:1612.03651*, 2016.

[188] Daniel Jurafsky and James H Martin. Speech and language processing: An introduction to natural language processing, computational linguistics, and speech recognition.

[189] Daniel Jurafsky and James H Martin. Speech and language processing. *Chapter A: Hidden Markov Models (Draft of October 2, 2019). Retrieved October*, 19:2019, 2019.

[190] Mohammed Kaity and Vimala Balakrishnan. Sentiment lexicons and non-english languages: a survey. *Knowledge and Information Systems*, 62(12):4445–4480, 2020.

[191] Irfan Ali Kandhro, Muhammad Ameen Chhajro, Kamlesh Kumar, Haque Nawaz Lashari, and Usman Khan. Student feedback sentiment analysis model using various machine learning schemes: a review. *Indian Journal of Science and Technology*, 12(14):1–9, 2019.

[192] P Karthika, R Murugeswari, and R Manoranjithem. Sentiment analysis of social media network using random forest algorithm. In *2019 IEEE international conference on intelligent techniques in control, optimization and signal processing (INCOS)*, pages 1–5. IEEE, 2019.

[193] Chhinder Kaur and Anand Sharma. Sentiment analysis of tweets on social issues using machine learning approach. *International Journal*, 9(4), 2020.

[194] Fatema Khatun, SM Mazharul Hoque Chowdhury, Zerin Nasrin Tumpa, SK Fazlee Rabby, Syed Akhter Hossain, and Sheikh Abujar. Sentiment analysis of amazon book review data using lexicon based analysis. In *Computational Vision and Bio-Inspired Computing: ICCVBIC 2019*, pages 1303–1309. Springer, 2020.

[195] Subhendu Khatuya, Niloy Ganguly, Jayanta Basak, Madhumita Bharde, and Bivas Mitra. Adele: Anomaly detection from event log empiricism. In *IEEE INFOCOM 2018-IEEE Conference on Computer Communications*, pages 2114–2122. IEEE, 2018.

[196] Chloé Kiddon, Luke Zettlemoyer, and Yejin Choi. Globally coherent text generation with neural checklist models. In *Proceedings of the 2016 conference on empirical methods in natural language processing*, pages 329–339, 2016.

[197] Tatsuaki Kimura, Keisuke Ishibashi, Tatsuya Mori, Hiroshi Sawada, Tsuyoshi Toyono, Ken Nishimatsu, Akio Watanabe, Akihiro Shimoda, and Kohei Shiomoto. Spatio-temporal factorization of log data for understanding network events. In *IEEE INFOCOM 2014-IEEE Conference on Computer Communications*, pages 610–618. IEEE, 2014.

[198] Tatsuaki Kimura, Akio Watanabe, Tsuyoshi Toyono, and Keisuke Ishibashi. Proactive failure detection learning generation patterns of large-scale network logs. *IEICE Transactions on Communications*, 102(2):306–316, 2019.

[199] Kaitlin Kirasich, Trace Smith, and Bivin Sadler. Random forest vs logistic regression: binary classification for heterogeneous datasets. *SMU Data Science Review*, 1(3):9, 2018.

[200] Svetlana Kiritchenko, Xiaodan Zhu, and Saif M Mohammad. Sentiment analysis of short informal texts. *Journal of Artificial Intelligence Research*, 50:723–762, 2014.

[201] Jannis Klinkenberg, Christian Terboven, Stefan Lankes, and Matthias S Müller. Data mining-based analysis of hpc center operations. In *2017 IEEE International Conference on Cluster Computing (CLUSTER)*, pages 766–773. IEEE, 2017.

[202] Jannis Klinkenberg, Christian Terboven, Stefan Lankes, and Matthias S. Müller. Data mining-based analysis of hpc center operations.

In *2017 IEEE International Conference on Cluster Computing (CLUSTER)*, pages 766–773, 2017. doi: 10.1109/CLUSTER.2017.23.

[203] Satoru Kobayashi, Yuya Yamashiro, Kazuki Otomo, and Kensuke Fukuda. amulog: A general log analysis framework for diverse template generation methods. In *2020 16th International Conference on Network and Service Management (CNSM)*, pages 1–5. IEEE, 2020.

[204] Zhiling Lan, Ziming Zheng, and Yawei Li. Toward automated anomaly identification in large-scale systems. *IEEE Transactions on Parallel and Distributed Systems*, 21(2):174–187, 2009.

[205] Jean-Claude Laprie. Dependable computing: Concepts, limits, challenges. In *Special issue of the 25th international symposium on fault-tolerant computing*, pages 42–54. Citeseer, 1995.

[206] Van-Hoang Le and Hongyu Zhang. Log-based anomaly detection without log parsing. *arXiv preprint arXiv:2108.01955*, 2021.

[207] Yukyung Lee, Jina Kim, and Pilsung Kang. Lanobert: System log anomaly detection based on bert masked language model. *arXiv preprint arXiv:2111.09564*, 2021.

[208] Dan Li and Jiang Qian. Text sentiment analysis based on long short-term memory. In *2016 First IEEE International Conference on Computer Communication and the Internet (ICCCI)*, pages 471–475. IEEE, 2016.

[209] Longhao Li and Taieb Znati. Atfp: Attention-based failure predictor for extreme-scale computing. In *2022 13th International Conference on Reliability, Maintainability, and Safety (ICRMS)*, pages 23–27. IEEE, 2022.

[210] Yangguang Li, Zhen Ming Jiang, Heng Li, Ahmed E Hassan, Cheng He, Ruirui Huang, Zhengda Zeng, Mian Wang, and Pinan Chen. Predicting node failures in an ultra-large-scale cloud computing

platform: an aiops solution. *ACM Transactions on Software Engineering and Methodology (TOSEM)*, 29(2):1–24, 2020.

[211] Zheng Li, Yue Zhao, Nicola Botta, Cezar Ionescu, and Xiyang Hu. Copod: copula-based outlier detection. In *2020 IEEE international conference on data mining (ICDM)*, pages 1118–1123. IEEE, 2020.

[212] Yinglung Liang, Yanyong Zhang, A. Sivasubramaniam, M. Jette, and R. Sahoo. Bluegene/l failure analysis and prediction models. In *International Conference on Dependable Systems and Networks (DSN'06)*, pages 425–434, 2006. doi: 10.1109/DSN.2006.18.

[213] Yinglung Liang, Yanyong Zhang, Anand Sivasubramaniam, Morris Jette, and Ramendra Sahoo. Bluegene/l failure analysis and prediction models. In *International Conference on Dependable Systems and Networks (DSN'06)*, pages 425–434. IEEE, 2006.

[214] Yinglung Liang, Yanyong Zhang, Hui Xiong, and Ramendra Sahoo. Failure prediction in IBM bluegene/l event logs. In *Seventh IEEE International Conference on Data Mining (ICDM 2007)*, pages 583–588. IEEE, 2007.

[215] Chinghway Lim, Navjot Singh, and Shalini Yajnik. A log mining approach to failure analysis of enterprise telephony systems. In *2008 IEEE International Conference on Dependable Systems and Networks With FTCS and DCC (DSN)*, pages 398–403. IEEE, 2008.

[216] Chin-Yew Lin. Rouge: A package for automatic evaluation of summaries. In *Text summarization branches out*, pages 74–81, 2004.

[217] Qingwei Lin, Hongyu Zhang, Jian-Guang Lou, Yu Zhang, and Xuewei Chen. Log clustering based problem identification for online service systems. In *2016 IEEE/ACM 38th International Conference on Software Engineering Companion (ICSE-C)*, pages 102–111. IEEE, 2016.

[218] Qingwei Lin, Ken Hsieh, Yingnong Dang, Hongyu Zhang, Kaixin Sui, Yong Xu, Jian-Guang Lou, Chenggang Li, Youjiang Wu, Randolph Yao, et al. 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*, pages 480–490, 2018.

[219] Fei Tony Liu, Kai Ming Ting, and Zhi-Hua Zhou. Isolation-based anomaly detection. *ACM Transactions on Knowledge Discovery from Data (TKDD)*, 6(1):1–39, 2012.

[220] Quan Liu, Hui Jiang, Si Wei, Zhen-Hua Ling, and Yu Hu. Learning semantic word embeddings based on ordinal knowledge constraints. In *Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*, pages 1501–1511, 2015.

[221] Rui-Tao Liu and Zuo-Ning Chen. A large-scale study of failures on petascale supercomputers. *Journal of Computer Science and Technology*, 33(1):24–41, 2018.

[222] Tianyu Liu, Kexiang Wang, Lei Sha, Baobao Chang, and Zhifang Sui. Table-to-text generation by structure-aware seq2seq learning. In *Proceedings of the AAAI conference on artificial intelligence*, volume 32, 2018.

[223] Jian-Guang Lou, Qiang Fu, Shenqi Yang, Ye Xu, and Jiang Li. Mining invariants from console logs for system problem detection. In *2010 USENIX Annual Technical Conference (USENIX ATC 10)*, 2010.

[224] Jie Lu, Feng Li, Lian Li, and Xiaobing 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*, pages 3–14, 2018.

[225] Sidi Lu, Bing Luo, Tirthak Patel, Yongtao Yao, Devesh Tiwari, and Weisong Shi. Making disk failure predictions smarter! In *18th {USENIX} Conference on File and Storage Technologies ({FAST} 20)*, pages 151–167, 2020.

[226] Siyang Lu, BingBing Rao, Xiang Wei, Byungchul Tak, Long Wang, and Liqiang Wang. Log-based abnormal task detection and root cause analysis for spark. In *2017 IEEE International Conference on Web Services (ICWS)*, pages 389–396. IEEE, 2017.

[227] Siyang Lu, Xiang Wei, Yandong Li, and Liqiang Wang. Detecting anomaly in big data system logs using convolutional neural network. In *2018 IEEE 16th Intl Conf on Dependable, Autonomic and Secure Computing, 16th Intl Conf on Pervasive Intelligence and Computing, 4th Intl Conf on Big Data Intelligence and Computing and Cyber Science and Technology Congress (DASC/PiCom/DataCom/CyberSciTech)*, pages 151–158. IEEE, 2018.

[228] Xu LU, Hui qiang WANG, Ren jie ZHOU, and Bao yu GE. Autonomic failure prediction based on manifold learning for large-scale distributed systems. *The Journal of China Universities of Posts and Telecommunications*, 17(4):116–124, 2010. ISSN 1005-8885. doi: https://doi.org/10.1016/S1005-8885(09)60497-0. URL `https://www.sciencedirect.com/science/article/pii/S1005888509604970`.

[229] Ao Ma, Fred Douglis, Guanlin Lu, Darren Sawyer, Surendar Chandra, and Windsor Hsu. Raidshield: Characterizing, monitoring, and proactively protecting against disk failures. In *Proceedings of the 13th USENIX Conference on File and Storage Technologies*, FAST'15, page 241–256, USA, 2015. USENIX Association. ISBN 9781931971201.

[230] Ao Ma, Rachel Traylor, Fred Douglis, Mark Chamness, Guanlin Lu, Darren Sawyer, Surendar Chandra, and Windsor Hsu. Raidshield:

characterizing, monitoring, and proactively protecting against disk failures. *ACM Transactions on Storage (TOS)*, 11(4):1–28, 2015.

[231] Yukun Ma, Haiyun Peng, Tahir Khan, Erik Cambria, and Amir Hussain. Sentic lstm: a hybrid network for targeted aspect-based sentiment analysis. *Cognitive Computation*, 10:639–650, 2018.

[232] Adetokunbo AO Makanju, A Nur Zincir-Heywood, and Evangelos E Milios. Clustering event logs using iterative partitioning. In *Proceedings of the 15th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 1255–1264, 2009.

[233] Addi Malviya-Thakur, David E Bernholdt, William F Godoy, Gregory R Watson, Mathieu Doucet, Mark A Coletti, David M Rogers, Marshall McDonnell, Jay Jay Billings, and Barney Maccabe. Research software engineering at oak ridge national laboratory. *Computing in Science & Engineering*, 2023.

[234] Angelos K Marnerides, Simon Malinowski, Ricardo Morla, and Hyong S Kim. Fault diagnosis in dsl networks using support vector machines. *Computer Communications*, 62:72–84, 2015.

[235] Maha Mdini, Gwendal Simon, Alberto Blanc, and Julien Lecoeuvre. Arcd: a solution for root cause diagnosis in mobile networks. In *2018 14th International Conference on Network and Service Management (CNSM)*, pages 280–284. IEEE, 2018.

[236] Walaa Medhat, Ahmed Hassan, and Hoda Korashy. Sentiment analysis algorithms and applications: A survey. *Ain Shams engineering journal*, 5(4):1093–1113, 2014.

[237] Weibin Meng, Ying Liu, Shenglin Zhang, Dan Pei, Hui Dong, Lei Song, and Xulong Luo. Device-agnostic log anomaly classification with partial labels. In *2018 IEEE/ACM 26th International Symposium on Quality of Service (IWQoS)*, pages 1–6. IEEE, 2018.

[238] Weibin Meng, Ying Liu, Yichen Zhu, Shenglin Zhang, Dan Pei, Yuqing Liu, Yihao Chen, Ruizhi Zhang, Shimin Tao, Pei Sun, et al.

Loganomaly: Unsupervised detection of sequential and quantitative anomalies in unstructured logs. In *IJCAI*, volume 19, pages 4739–4745, 2019.

[239] Weibin Meng, Ying Liu, Yuheng Huang, Shenglin Zhang, Federico Zaiter, Bingjin Chen, and Dan Pei. A semantic-aware representation framework for online log analysis. In *2020 29th International Conference on Computer Communications and Networks (ICCCN)*, pages 1–7. IEEE, 2020.

[240] Weibin Meng, Ying Liu, Federico Zaiter, Shenglin Zhang, Yihao Chen, Yuzhe Zhang, Yichen Zhu, En Wang, Ruizhi Zhang, Shimin Tao, et al. Logparse: Making log parsing adaptive through word classification. In *2020 29th International Conference on Computer Communications and Networks (ICCCN)*, pages 1–9. IEEE, 2020.

[241] Weibin Meng, Ying Liu, Shenglin Zhang, Federico Zaiter, Yuzhe Zhang, Yuheng Huang, Zhaoyang Yu, Yuzhi Zhang, Lei Song, Ming Zhang, et al. Logclass: Anomalous log identification and classification with partial labels. *IEEE Transactions on Network and Service Management*, 2021.

[242] Salma Messaoudi, Annibale Panichella, Domenico Bianculli, Lionel Briand, and Raimondas Sasnauskas. A search-based approach for accurate identification of log message formats. In *2018 IEEE/ACM 26th International Conference on Program Comprehension (ICPC)*, pages 167–16710. IEEE, 2018.

[243] Subrata Mitra, Rajesh Panta, Moo-Ryong Ra, and Saurabh Bagchi. Partial-parallel-repair (ppr) a distributed technique for repairing erasure coded storage. In *Proceedings of the eleventh European conference on computer systems*, pages 1–16, 2016.

[244] Saif Mohammad, Cody Dunne, and Bonnie Dorr. Generating high-coverage semantic orientation lexicons from overtly marked words and a thesaurus. In *Proceedings of the 2009 conference on empirical methods in natural language processing*, pages 599–608, 2009.

[245] Bashir Mohammed, Irfan Awan, Hassan Ugail, and Muhammad Younas. Failure prediction using machine learning in a virtualised HPC system and application. *Cluster Computing*, 22(2):471–485, 2019.

[246] Marrin Molan, Andrea Borghesi, Francesco Beneventi, Massimiliano Guarrasi, and Andrea Bartolini. An explainable model for fault detection in hpc systems. In *International conference on high performance computing*, pages 378–391. Springer, 2021.

[247] Cristian Molinaro, Vincenzo Moscato, Antonio Picariello, Andrea Pugliese, Antonino Rullo, and VS Subrahmanian. Padua: Parallel architecture to detect unexplained activities. *ACM Transactions on Internet Technology (TOIT)*, 14(1):1–28, 2014.

[248] Alejandro Moreo, M Romero, JL Castro, and Jose Manuel Zurita. Lexicon-based comments-oriented news sentiment analyzer system. *Expert Systems with Applications*, 39(10):9166–9180, 2012.

[249] Gihan R Mudalige, IZ Reguly, Satya P Jammy, Christian T Jacobs, Michael B Giles, and Neil D Sandham. Large-scale performance of a dsl-based multi-block structured-mesh application for direct numerical simulation. *Journal of Parallel and Distributed Computing*, 131:130–146, 2019.

[250] Gihan R Mudalige, Istvan Z Reguly, Arun Prabhakar, Dario Amirante, Leigh Lapworth, and Stephen A Jarvis. Towards virtual certification of gas turbine engines with performance-portable simulations. In *2022 IEEE International Conference on Cluster Computing (CLUSTER)*, pages 206–217. IEEE, 2022.

[251] GSN Murthy, Shanmukha Rao Allu, Bhargavi Andhavarapu, Mounika Bagadi, and Mounika Belusonti. Text based sentiment analysis using lstm. *Int. J. Eng. Res. Tech. Res*, 9(05), 2020.

[252] Meiyappan Nagappan and Mladen A Vouk. Abstracting log lines to log event types for mining software system logs. In *2010 7th*

IEEE Working Conference on Mining Software Repositories (MSR 2010), pages 114–117. IEEE, 2010.

[253] Nithin Nakka, Ankit Agrawal, and Alok 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*, pages 1557–1566, 2011. doi: 10.1109/IPDPS.2011.310.

[254] Animesh Nandi, Atri Mandal, Shubham Atreja, Gargi B Dasgupta, and Subhrajit Bhattacharya. Anomaly detection using program control flow graph mining from execution logs. In *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 215–224, 2016.

[255] Pansy Nandwani and Rupali Verma. A review on sentiment analysis and emotion detection from text. *Social Network Analysis and Mining*, 11(1):81, 2021.

[256] Sasho Nedelkoski, Jasmin Bogatinovski, Alexander Acker, Jorge Cardoso, and Odej Kao. Self-supervised log parsing. *arXiv preprint arXiv:2003.07905*, 2020.

[257] Alessio Netti, Zeynep Kiziltan, Ozalp Babaoglu, Alina Sîrbu, Andrea Bartolini, and Andrea Borghesi. A machine learning approach to online fault classification in hpc systems. *Future Generation Computer Systems*, 110:1009–1022, 2020.

[258] 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)*, pages 22–31, 2017.

[259] Bin Nie, Ji Xue, Saurabh Gupta, Tirthak Patel, Christian Engelmann, Evgenia Smirni, and Devesh Tiwari. Machine learning

models for GPU error prediction in a large scale HPC system. In *2018 48th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)*, pages 95–106. IEEE, 2018.

[260] Xia Ning, Geoff Jiang, Haifeng Chen, and Kenji Yoshihira. 1hlaer: a system for heterogeneous log analysis. 2014.

[261] Gang Niu, Marthinus Christoffel du Plessis, Tomoya Sakai, Yao Ma, and Masashi Sugiyama. Theoretical comparisons of positive-unlabeled learning against positive-negative learning. *Advances in neural information processing systems*, 29, 2016.

[262] Ahmed Assim Nsaif and Dhafar Hamed Abd. Sentiment analysis of political post classification based on xgboost. In *Proceedings of International Conference on Computing and Communication Networks: ICCCN 2021*, pages 177–188. Springer, 2022.

[263] Kishore Papineni, Salim Roukos, Todd Ward, and Wei-Jing Zhu. Bleu: a method for automatic evaluation of machine translation. In *Proceedings of the 40th annual meeting of the Association for Computational Linguistics*, pages 311–318, 2002.

[264] Wenjie Pei, Tadas Baltrusaitis, David MJ Tax, and Louis-Philippe Morency. Temporal attention-gated model for robust sequence classification. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 6730–6739, 2017.

[265] Alejandro Pelaez, Andres Quiroz, James C. Browne, Edward Chuah, and Manish Parashar. Online failure prediction for hpc resources using decentralized clustering. In *2014 21st International Conference on High Performance Computing (HiPC)*, pages 1–9, 2014. doi: 10.1109/HiPC.2014.7116903.

[266] Teerat Pitakrat, Dušan Okanović, André van Hoorn, and Lars Grunske. Hora: Architecture-aware online failure prediction. *Journal of Systems and Software*, 137:669–685, 2018.

[267] Daniel Plaisted and Mengjun Xie. Dip: a log parser based on" disagreement index token" conditions. In *Proceedings of the 2022 ACM Southeast Conference*, pages 113–122, 2022.

[268] Marc Platini, Thomas Ropars, Benoit Pelletier, and Noel De Palma. Logflow: Simplified log analysis for large scale systems. In *International Conference on Distributed Computing and Networking 2021*, pages 116–125, 2021.

[269] Guang Qiu, Xiaofei He, Feng Zhang, Yuan Shi, Jiajun Bu, and Chun Chen. Dasa: dissatisfaction-oriented advertising based on sentiment analysis. *Expert Systems with Applications*, 37(9):6182–6191, 2010.

[270] Tongqing Qiu, Zihui Ge, Dan Pei, Jia Wang, and Jun Xu. What happened in my network: mining network events from router syslogs. In *Proceedings of the 10th ACM SIGCOMM conference on Internet measurement*, pages 472–484, 2010.

[271] Alec Radford, Karthik Narasimhan, Tim Salimans, Ilya Sutskever, et al. Improving language understanding by generative pre-training. 2018.

[272] Alec Radford, Jeffrey Wu, Rewon Child, David Luan, Dario Amodei, Ilya Sutskever, et al. Language models are unsupervised multitask learners. *OpenAI blog*, 1(8):9, 2019.

[273] Trishna Rajkumar and Johnny Öberg. Anode: A log-based self-supervised framework to detect scrubber failures in sram-fpga. In *2022 IEEE 27th Pacific Rim International Symposium on Dependable Computing (PRDC)*, pages 164–171. IEEE, 2022.

[274] Ajay Rawat, Rama Sushil, Amit Agarwal, and Afzal Sikander. A new approach for vm failure prediction using stochastic model in cloud. *IETE Journal of research*, 67(2):165–172, 2021.

[275] Rui Ren, Jinheng Li, Yan Yin, and Shuai Tian. Failure prediction for large-scale clusters logs via mining frequent patterns. In *Bench-*

Council International Federated Intelligent Computing and Block Chain Conferences, pages 147–165. Springer, 2020.

[276] Arash Rezaei, Frank Mueller, Paul Hargrove, and Eric Roman. Dino: Divergent node cloning for sustained redundancy in hpc. *Journal of Parallel and Distributed Computing*, 109:350–362, 2017.

[277] Stephen Robertson. Understanding inverse document frequency: on theoretical arguments for idf. *Journal of documentation*, 2004.

[278] Ananya B Sai, Akash Kumar Mohankumar, and Mitesh M Khapra. A survey of evaluation metrics used for nlg systems. *arXiv preprint arXiv:2008.12009*, 2020.

[279] Felix Salfner and Miroslaw Malek. Using hidden semi-markov models for effective online failure prediction. In *2007 26th IEEE International Symposium on Reliable Distributed Systems (SRDS 2007)*, pages 161–174. IEEE, 2007.

[280] Amina Samih, Abderrahim Ghadi, and Abdelhadi Fennan. Enhanced sentiment analysis based on improved word embeddings and xgboost. *International Journal of Electrical and Computer Engineering*, 13(2):1827, 2023.

[281] Mike Schuster and Kaisuke Nakajima. Japanese and korean voice search. In *2012 IEEE international conference on acoustics, speech and signal processing (ICASSP)*, pages 5149–5152. IEEE, 2012.

[282] Mike Schuster and Kuldip K Paliwal. Bidirectional recurrent neural networks. *IEEE transactions on Signal Processing*, 45(11):2673–2681, 1997.

[283] Fabrizio Sebastiani and Andrea Esuli. Sentiwordnet: A publicly available lexical resource for opinion mining. In *Proceedings of the 5th international conference on language resources and evaluation*, pages 417–422. European Language Resources Association (ELRA) Genoa, Italy, 2006.

[284] Mina Sedaghat, Eddie Wadbro, John Wilkes, Sara De Luna, Oleg Seleznjev, and Erik Elmroth. Diehard: reliable scheduling to survive correlated failures in cloud data centers. In *2016 16th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGrid)*, pages 52–59. IEEE, 2016.

[285] Febrian Setianto, Erion Tsani, Fatima Sadiq, Georgios Domalis, Dimitris Tsakalidis, and Panos Kostakos. Gpt-2c: A gpt-2 parser for cowrie honeypot logs. *arXiv preprint arXiv:2109.06595*, 2021.

[286] Alex Sherstinsky. Fundamentals of recurrent neural network (rnn) and long short-term memory (lstm) network. *Physica D: Nonlinear Phenomena*, 404:132306, 2020.

[287] Keiichi Shima. Length matters: Clustering system log messages using length of words. *arXiv preprint arXiv:1611.03213*, 2016.

[288] Kush Shrivastava and Shishir Kumar. A sentiment analysis system for the hindi language by integrating gated recurrent unit with genetic algorithm. *Int. Arab J. Inf. Technol.*, 17(6):954–964, 2020.

[289] Nikolay A. Simakov, Joseph P. White, Robert L. DeLeon, Steven M. Gallo, Matthew D. Jones, Jeffrey T. Palmer, Benjamin Plessinger, and Thomas R. Furlani. A workload analysis of nsf's innovative hpc resources using xdmod, 2018.

[290] Jyoti Prakash Singh, Seda Irani, Nripendra P Rana, Yogesh K Dwivedi, Sunil Saumya, and Pradeep Kumar Roy. Predicting the "helpfulness" of online consumer reviews. *Journal of Business Research*, 70:346–355, 2017.

[291] Alina Sîrbu and Özalp Babaoglu. Towards operator-less data centers through data-driven, predictive, proactive autonomics. *CoRR*, abs/1606.04456, 2016. URL `http://arxiv.org/abs/1606.04456`.

[292] Donghwan Song, Adrian Matias Chung Baek, and Namhun Kim. Forecasting stock market indices using padding-based fourier trans-

form denoising and time series deep learning models. *IEEE Access*, 9:83786–83796, 2021.

[293] Junseok Song, Kyung Tae Kim, Byungjun Lee, Sangyoung Kim, and Hee Yong Youn. A novel classification approach based on naïve bayes for twitter sentiment analysis. *KSII Transactions on Internet and Information Systems (TIIS)*, 11(6):2996–3011, 2017.

[294] Mbarka Soualhia, Foutse Khomh, and Sofiene 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*, pages 58–65, 2015. doi: 10.1109/HPCC-CSS-ICESS.2015.170.

[295] Hudan Studiawan, Ferdous Sohel, and Christian Payne. Anomaly detection in operating system logs with deep learning-based sentiment analysis. *IEEE Transactions on Dependable and Secure Computing*, 18(5):2136–2148, 2020.

[296] Hudan Studiawan, Ferdous Sohel, and Christian Payne. Sentiment analysis in a forensic timeline with deep learning. *IEEE Access*, 8:60664–60675, 2020.

[297] Aixin Sun. Short text classification using very few words. In *Proceedings of the 35th international ACM SIGIR conference on Research and development in information retrieval*, pages 1145–1146, 2012.

[298] Luchen Tan, Haotian Zhang, Charles Clarke, and Mark Smucker. Lexical comparison between wikipedia and twitter corpora by using word embeddings. In *Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing (Volume 2: Short Papers)*, pages 657–661, 2015.

[299] Shimin Tao, Weibin Meng, Yimeng Cheng, Yichen Zhu, Ying Liu, Chunning Du, Tao Han, Yongpeng Zhao, Xiangguang Wang, and Hao Yang. Logstamp: Automatic online log parsing based on sequence labelling. *ACM SIGMETRICS Performance Evaluation Review*, 49(4):93–98, 2022.

[300] Harsh Thakkar and Dhiren Patel. Approaches for sentiment analysis on twitter: A state-of-art study. *arXiv preprint arXiv:1512.01043*, 2015.

[301] Joshua Thompson, David W. Dreisigmeyer, Terry Jones, Michael Kirby, and Joshua Ladd. Accurate fault prediction of bluegene/p ras logs via geometric reduction. In *2010 International Conference on Dependable Systems and Networks Workshops (DSN-W)*, pages 8–14, 2010. doi: 10.1109/DSNW.2010.5542626.

[302] Peter D Turney and Michael L Littman. Measuring praise and criticism: Inference of semantic orientation from association. *acm Transactions on Information Systems (tois)*, 21(4):315–346, 2003.

[303] Risto Vaarandi and Mauno Pihelgas. Logcluster-a data clustering and pattern mining algorithm for event logs. In *2015 11th International conference on network and service management (CNSM)*, pages 1–7. IEEE, 2015.

[304] Maarten Van Steen and Andrew S Tanenbaum. *Distributed systems*. Maarten van Steen Leiden, The Netherlands, 2017.

[305] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. In *Advances in neural information processing systems*, pages 5998–6008, 2017.

[306] Binita Verma and Ramjeevan Singh Thakur. Sentiment analysis using lexicon and machine learning-based approaches: A survey. In *Proceedings of International Conference on Recent Advancement on Computer and Communication: ICRAC 2017*, pages 441–447. Springer, 2018.

[307] Arthur Vervaet, Raja Chiky, and Mar Callau-Zori. Ustep: Unfixed search tree for efficient log parsing. In *2021 IEEE International Conference on Data Mining (ICDM)*, pages 659–668. IEEE, 2021.

[308] Chao Wang, Frank Mueller, Christian Engelmann, and Stephen L Scott. Proactive process-level live migration in hpc environments. In *SC'08: Proceedings of the 2008 ACM/IEEE conference on Supercomputing*, pages 1–12. IEEE, 2008.

[309] Haoyu Wang and Haiying Shen. Proactive incast congestion control in a datacenter serving web applications. In *IEEE INFOCOM 2018-IEEE Conference on Computer Communications*, pages 19–27. IEEE, 2018.

[310] Haoyu Wang, Haiying Shen, and Zhuozhao Li. Approaches for resilience against cascading failures in cloud datacenters. In *2018 IEEE 38th International Conference on Distributed Computing Systems (ICDCS)*, pages 706–717. IEEE, 2018.

[311] Mengying Wang, Lele Xu, and Lili Guo. Anomaly detection of system logs based on natural language processing and deep learning. In *2018 4th International Conference on Frontiers of Signal Processing (ICFSP)*, pages 140–144. IEEE, 2018.

[312] Xuerui Wang and Andrew 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*, pages 424–433, 2006.

[313] Zumin Wang, Jiyu Tian, Hui Fang, Liming Chen, and Jing Qin. Lightlog: A lightweight temporal convolutional network for log anomaly detection on the edge. *Computer Networks*, 203:108616, 2022.

[314] Mayur Wankhade, Annavarapu Chandra Sekhara Rao, and Chaitanya Kulkarni. A survey on sentiment analysis methods, applications, and challenges. *Artificial Intelligence Review*, 55(7):5731–5780, 2022.

[315] Yukihiro Watanabe and Yasuhide Matsumoto. Online failure prediction in cloud datacenters. *Fujitsu scientific & technical journal*, 50(1):67–71, 2014.

[316] Yukihiro Watanabe, Hiroshi Otsuka, Masataka Sonoda, Shinji Kikuchi, and Yasuhide 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*, pages 504–511. IEEE, 2012.

[317] Ronald J Williams and David Zipser. A learning algorithm for continually running fully recurrent neural networks. *Neural computation*, 1(2):270–280, 1989.

[318] Thorsten Wittkopp, Alexander Acker, Sasho Nedelkoski, Jasmin Bogatinovski, Dominik Scheinert, Wu Fan, and Odej Kao. A2log: Attentive augmented log anomaly detection, 2021.

[319] Yonghui Wu, Mike Schuster, Zhifeng Chen, Quoc V Le, Mohammad Norouzi, Wolfgang Macherey, Maxim Krikun, Yuan Cao, Qin Gao, Klaus Macherey, et al. Google's neural machine translation system: Bridging the gap between human and machine translation. *arXiv preprint arXiv:1609.08144*, 2016.

[320] Zhiyue Wu, Hongzuo Xu, Guansong Pang, Fengyuan Yu, Yijie Wang, Songlei Jian, and Yongjun Wang. Dram failure prediction in aiops: Empirical evaluation, challenges and opportunities. *arXiv preprint arXiv:2104.15052*, 2021.

[321] Bin Xia, Yuxuan Bai, Junjie Yin, Yun Li, and Jian Xu. Loggan: A log-level generative adversarial network for anomaly detection using permutation event modeling. *Information Systems Frontiers*, 23(2):285–298, 2021.

[322] Chuming Xiao, Jiaming Huang, and Weigang Wu. Detecting anomalies in cluster system using hybrid deep learning model. In *International Symposium on Parallel Architectures, Algorithms and Programming*, pages 393–404. Springer, 2019.

[323] Yongzheng Xie, Hongyu Zhang, Bo Zhang, Muhammad Ali Babar, and Sha Lu. Logdp: Combining dependency and proximity for log-based anomaly detection. In *Service-Oriented Computing: 19th International Conference, ICSOC 2021, Virtual Event, November 22–25, 2021, Proceedings 19*, pages 708–716. Springer, 2021.

[324] Yongzheng Xie, Hongyu Zhang, and Muhammad Ali Babar. Loggd: Detecting anomalies from system logs by graph neural networks. *arXiv preprint arXiv:2209.07869*, 2022.

[325] Chang Xu, Gang Wang, Xiaoguang Liu, Dongdong Guo, and Tie-Yan Liu. Health status assessment and failure prediction for hard drives with recurrent neural networks. *IEEE Transactions on Computers*, 65(11):3502–3508, 2016.

[326] Qianwen Ariel Xu, Victor Chang, and Chrisina Jayne. A systematic review of social media-based sentiment analysis: Emerging trends and challenges. *Decision Analytics Journal*, page 100073, 2022.

[327] Wei Xu, Ling Huang, Armando Fox, David Patterson, and Michael Jordan. Online system problem detection by mining patterns of console logs. In *2009 ninth IEEE international conference on data mining*, pages 588–597. IEEE, 2009.

[328] Sunita A. Yadwad and V. Valli Kumari. Predicting service outages using tweets. In *International Journal of Recent Technology and Engineering (IJRTE)*, 2020.

[329] Kenji Yamanishi and Yuko Maruyama. Dynamic syslog mining for network failure monitoring. In *Proceedings of the eleventh ACM SIGKDD international conference on Knowledge discovery in data mining*, pages 499–508, 2005.

[330] Lin Yang, Junjie Chen, Zan Wang, Weijing Wang, Jiajun Jiang, Xuyuan Dong, and Wenbin Zhang. Semi-supervised log-based anomaly detection via probabilistic label estimation. In *2021 IEEE/ACM 43rd International Conference on Software Engineering (ICSE)*, pages 1448–1460. IEEE, 2021.

[331] Ruipeng Yang, Dan Qu, Yekui Qian, Yusheng Dai, and Shaowei Zhu. An online log template extraction method based on hierarchical clustering. *EURASIP Journal on Wireless Communications and Networking*, 2019(1):1–12, 2019.

[332] John W. Young. A first order approximation to the optimum checkpoint interval. *Commun. ACM*, 17(9):530–531, sep 1974. ISSN 0001-0782. doi: 10.1145/361147.361115. URL `https://doi.org/10.1145/361147.361115`.

[333] Li Yu, Ziming Zheng, Zhiling Lan, and Susan Coghlan. Practical online failure prediction for blue gene/p: Period-based vs event-driven. In *2011 IEEE/IFIP 41st International Conference on Dependable Systems and Networks Workshops (DSN-W)*, pages 259–264, 2011. doi: 10.1109/DSNW.2011.5958823.

[334] Qiao Yu, Zhang Wengui, Haeri Soroush, Notaro Paolo, Jorge Cardoso, and Odej Kao. Himfp: Hierarchical intelligent memory failure prediction for cloud service reliability. In *2023 53st Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)*. IEEE, 2023.

[335] Yan Yu and Haopeng Chen. An approach to failure prediction in cluster by self-updating cause-and-effect graph. In *International Conference on Cloud Computing*, pages 114–129. Springer, 2019.

[336] Ding Yuan, Haohui Mai, Weiwei Xiong, Lin Tan, Yuanyuan Zhou, and Shankar Pasupathy. Sherlog: error diagnosis by connecting clues from run-time logs. In *Proceedings of the fifteenth International Conference on Architectural support for programming languages and operating systems*, pages 143–154, 2010.

[337] Nurulhuda Zainuddin, Ali Selamat, and Roliana Ibrahim. Improving twitter aspect-based sentiment analysis using hybrid approach. In *Intelligent Information and Database Systems: 8th Asian Conference, ACIIDS 2016, Da Nang, Vietnam, March 14–16, 2016, Proceedings, Part I 8*, pages 151–160. Springer, 2016.

[338] Di Zhang, Dong Dai, Runzhou Han, and Mai Zheng. Sentilog: Anomaly detecting on parallel file systems via log-based sentiment analysis. In *Proceedings of the 13th ACM Workshop on Hot Topics in Storage and File Systems*, pages 86–93, 2021.

[339] Lin Zhang, Xueshuo Xie, Kunpeng Xie, Zhi Wang, Ye Lu, and Yujun Zhang. An efficient log parsing algorithm based on heuristic rules. In *International Symposium on Advanced Parallel Processing Technologies*, pages 123–134. Springer, 2019.

[340] Shenglin Zhang, Ying Liu, Dan Pei, Yu Chen, Xianping Qu, Shimin Tao, Zhi Zang, Xiaowei Jing, and Mei Feng. Funnel: Assessing software changes in web-based services. *IEEE Transactions on Services Computing*, 11(1):34–48, 2016.

[341] Shenglin Zhang, Weibin Meng, Jiahao Bu, Sen Yang, Ying Liu, Dan Pei, Jun Xu, Yu Chen, Hui Dong, Xianping Qu, et al. Syslog processing for switch failure diagnosis and prediction in datacenter networks. In *2017 IEEE/ACM 25th International Symposium on Quality of Service (IWQoS)*, pages 1–10. IEEE, 2017.

[342] Shenglin Zhang, Ying Liu, Weibin Meng, Zhiling Luo, Jiahao Bu, Sen Yang, Peixian Liang, Dan Pei, Jun Xu, Yuzhi Zhang, et al. Prefix: Switch failure prediction in datacenter networks. *Proceedings of the ACM on Measurement and Analysis of Computing Systems*, 2(1):1–29, 2018.

[343] Tong Zhang. Solving large scale linear prediction problems using stochastic gradient descent algorithms. In *Proceedings of the twenty-first international conference on Machine learning*, page 116, 2004.

[344] Wenhao Zhang, Hua Xu, and Wei Wan. Weakness finder: Find product weakness from chinese reviews by using aspects based sentiment analysis. *Expert Systems with Applications*, 39(11):10283–10291, 2012.

[345] Xiaobo Zhang and Qingsong Yu. Hotel reviews sentiment analysis based on word vector clustering. In *2017 2nd IEEE International*

Conference on Computational Intelligence and Applications (ICCIA), pages 260–264. IEEE, 2017.

[346] Xu Zhang, Yong Xu, Qingwei Lin, Bo Qiao, Hongyu Zhang, Yingnong Dang, Chunyu Xie, Xinsheng Yang, Qian Cheng, Ze Li, et al. 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*, pages 807–817, 2019.

[347] Ying Zhao, Xiang Liu, Siqing Gan, and Weimin Zheng. Predicting disk failures with hmm-and hsmm-based approaches. In *Industrial Conference on Data Mining*, pages 390–404. Springer, 2010.

[348] Zhenfei Zhao, Weina Niu, Xiaosong Zhang, Runzi Zhang, Zhenqi Yu, and Cheng Huang. Trine: Syslog anomaly detection with three transformer encoders in one generative adversarial network. *Applied Intelligence*, pages 1–10, 2021.

[349] Ziming Zheng, Zhiling Lan, Rinku Gupta, Susan Coghlan, and Peter 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)*, pages 15–22, 2010. doi: 10.1109/DSNW.2010.5542627.

[350] Ziming Zheng, Zhiling Lan, Rinku Gupta, Susan Coghlan, and Peter 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)*, pages 15–22. IEEE, 2010.

[351] Ziming Zheng, Li Yu, Zhiling Lan, and Terry Jones. 3-dimensional root cause diagnosis via co-analysis. In *Proceedings of the 9th international conference on Autonomic computing*, pages 181–190, 2012.

[352] Haoyi Zhou, Shanghang Zhang, Jieqi Peng, Shuai Zhang, Jianxin Li, Hui Xiong, and Wancai Zhang. Informer: Beyond efficient

transformer for long sequence time-series forecasting. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 35, pages 11106–11115, 2021.

[353] Bingpeng Zhu, Gang Wang, Xiaoguang Liu, Dianming Hu, Sheng Lin, and Jingwei Ma. Proactive drive failure prediction for large scale storage systems. In *2013 IEEE 29th Symposium on Mass Storage Systems and Technologies (MSST)*, pages 1–5, 2013. doi: 10.1109/MSST.2013.6558427.