A Framework for Proactive Fault Tolerance in Cloud-IoT Applications

by

Mohammad S. Jassas

A thesis submitted to the School of Graduate and Postdoctoral Studies
in partial fulfillment of the requirement for the degree of

Doctor of Philosophy in Electrical and Computer Engineering

Department of Electrical, Computer and Software Engineering
Faculty of Engineering and Applied Science
University of Ontario Institute of Technology (Ontario Tech University)
Oshawa, Ontario, Canada
April 2022

THESIS EXAMINATION INFORMATION

Submitted by: **Mohammad S. Jassas**

Doctor of Philosophy in Electrical and Computer Engineering

Thesis Title: A Framework for Proactive Fault Tolerance in Cloud-IoT Applications

An oral defence of this thesis took place on April 12, 2022 in front of the following examining committee:

Examining Committee:

Chair of Examining Committee:	Dr. Khalid Elgazzar
Research Supervisor:	Dr. Qusay H. Mahmoud
Examining Committee Member:	Dr. Masoud Makrehchi
Examining Committee Member:	Dr. Akramul Azim
University Examiner:	Dr. Richard W. Pazzi
External Examiner:	Dr. Cherie Ding, Ryerson University

The above committee determined that the thesis is acceptable in form and content and that a satisfactory knowledge of the field covered by the thesis was demonstrated by the candidate during an oral examination. A signed copy of the Certificate of Approval is available from the School of Graduate and Postdoctoral Studies.

Abstract

Integrating Internet of Things (IoT) devices with the cloud has several benefits, including expanding local IoT resources and improving cloud-IoT application performance. Cloud computing can benefit from IoT devices and applications by extending its scope to include real-world surroundings. On the other hand, IoT can use the cloud's unlimited computing and storage power. Modern cloud-based applications, including smart cities, home automation, and eHealth, require a highly scalable and available framework that enables computing, storage, and data analysis. Cloud computing cannot respond to the growing number of IoT devices due to its remote location, and cloud providers are struggling to meet the quality of service (QoS), such as low latency. Cloud applications have a high probability of failure as they operate in a large-scale environment, including physical and virtual machines. The Coronavirus pandemic (COVID-19) has tested cloud providers in many ways, none of which could have been predicted. Although the public cloud has proven remarkably resilient in overcoming an unprecedented stress test, there are remarkable exceptions to cloud failure problems that occurred in the first half of 2020.

In this thesis, the main objective is to design and implement a cloud-IoT framework that has been developed utilizing proactive fault tolerance techniques to provide high reliability and availability for IoT applications. The framework aims to decrease the number of task failures and minimize the time and cost of using the cloud. This thesis also analyzes and characterize the behaviour of failed and finished tasks using publicly accessible traces.

A design of highly reliable and available IoT applications has been proposed based on the development of Edge-Cloud architecture to support modern IoT applications. The evaluation results show a significant correlation between unsuccessful tasks and the resources requested. The results indicate that the proposed framework performance has improved, as well as the throughput efficiency increases by 55% after integrating the local resources with the cloud. The machine and deep learning-based failure prediction model can reduce the number of failed tasks for cloud-IoT applications. Moreover, the failure prediction model can predict failed tasks with a high rate of precision, recall, and F1-score.

Keywords: Fault Tolerance; Failure Analysis; Failure Prediction; Internet of Things; Cloud computing; Edge Computing; Machine Learning; Deep Learning

Author's Declaration

I hereby declare that this thesis consists of original work of which I have authored. This is a true copy of the thesis, including any required final revisions, as accepted by my examiners.

I authorize the University of Ontario Institute of Technology (Ontario Tech University) to lend this thesis to other institutions or individuals for the purpose of scholarly research. I further authorize the University of Ontario Institute of Technology (Ontario Tech University) to reproduce this thesis by photocopying or by other means, in total or in part, at the request of other institutions or individuals for the purpose of scholarly research. I understand that my thesis may be made electronically available to the public.

_____Mohammad S. Jassas

Acknowledgements

First and foremost, I would like to praise Allah the Almighty, the Most Gracious, and the Most Merciful for His blessing given to me during my study and in completing this thesis.

I am deeply grateful to my supervisor Dr. Qusay H. Mahmoud for his valuable guidance, continuous support, and patience during my PhD study. He provided me with the tools that I needed to choose the right direction and complete writing my thesis.

I would also like to express my gratitude to the other members of my thesis committee, Dr. Masoud Makrehchi, Dr. Akramul Azim, Dr. Richard W. Pazzi, Dr. Cherie Ding, and Dr. Khalid Elgazzar, for their support, feedback and insightful comments.

I would like to thank all the members in the Devices, Networks and Applications (DNA) Lab. It is their kind help and support that have made my study and life in Canada a wonderful time.

I would also like to give special thanks to my wife, daughters, and son for their patience and understanding when undertaking my research and writing my thesis.

Finally, I would like to express my gratitude to Umm Al-Qura University in Saudi Arabia for providing financial support.

Dedication

This thesis is dedicated to my wonderful parents who have raised me to be the person I am today.

Statement of Contributions

Results from this thesis research have been disseminated in the following publications:

1. M. S. Jassas, and Q. H. Mahmoud, " Analysis of Job Failure and Prediction Model for Cloud Computing using Machine Learning," Sensors 22, no. 5 (2022): 2035, DOI: 10.3390/s22052035.

2. M. S. Jassas and Q. H. Mahmoud, "Evaluation of Failure Analysis of IoT Applications Using Edge-Cloud Architecture," in 2022 IEEE International Systems Conference (SysCon), IEEE, 2022 (accepted, 8 pages).

3. M. S. Jassas and Q. H. Mahmoud, "A Failure Prediction Model for Large Scale Cloud Applications using Deep Learning," in 2021 IEEE International Systems Conference (SysCon), pp. 1-8. IEEE, 2021, DOI: 10.1109/SysCon48628.2021.9447141.

4. M. S. Jassas and Q. H. Mahmoud, "Evaluation of a failure prediction model for large scale cloud applications," in Canadian Conference on Artificial Intelligence. Springer, pp. 321–327, 2020, DOI: 10.1007/978-3-030-47358-7_32.

5. M. S. Jassas and Q. H. Mahmoud, "Failure characterization and prediction of scheduling jobs in google cluster traces," in 2019 IEEE 10th GCC Conference & Exhibition (GCC), pp. 1-7. IEEE, 2019, DOI: 10.1109/GCC45510.2019.1570516010.

6. M. S. Jassas and Q. H. Mahmoud, "Failure analysis and characterization of scheduling jobs in google cluster trace," in IECON 2018-44th Annual Conference of the IEEE

Industrial Electronics Society, pp. 3102-3107. IEEE, 2018,
DOI: 10.1109/IECON.2018.8592822.

7. M. S. Jassas, J. Mathew, A. Azim, and Q. H. Mahmoud, "A framework for extending resources of embedded systems using the Cloud," in In 2017 IEEE 30th Canadian Conference on Electrical and Computer Engineering (CCECE), pp. 1-5. IEEE, 2017, DOI: 10.1109/CCECE.2017.7946662.

Table of Contents

List of Tables

List of Figures

List of Abbreviations

API Application Programming Interface

DL deep learning

DTs Decision Trees

FNR False Negative Rate

FPR False Positive Rate

HPC High-Performance Computing

IaaS Infrastructure as a Service

IoT Internet of Things

KNN K-Nearest Neighbor

ML machine learning

NB Naive Bayes

PaaS Platform as a Service

QDA Quadratic Discriminant Analysis

QoS quality of services

RF Random Forest

SaaS Software as a Service

SLA Service Level Agreement

Chapter 1

Introduction

This chapter first provides an overview of cloud-IoT integration and fault tolerance techniques, followed by motivation and problem statement, research objectives, research contributions and thesis structure.

1.1 Overview

Cloud computing and the Internet of Things (IoT) are two of the most powerful technologies in today's modern world. Enterprises are increasingly moving workloads to public clouds and adopting multi-cloud strategies to save cost, enhance agility, and increase flexibility. The popularity of cloud computing over traditional distribution systems has resulted in cloud providers building massive data centres worldwide. The data centres operate in

different geographical regions to efficiently meet the enormous demand of their users. The value of cloud computing is to store, manage, and analyze user data in the data centers via the Internet instead of using local machines. Cloud providers are responsible for managing and maintaining the operation of these data centres to provide cloud consumers with high quality of services (QoS). Cloud users have full access to cloud services such as virtual machines or cloud storage using Application Programming Interface (API), enabling applications and data sources to communicate with each other. Another feature of cloud computing is scalability, which allows cloud users and organizations to scale in or out based on their business requirements.

Innovative IoT applications are being used in various fields, including health care and transportation, thanks to the advancement of IoT devices. According to [16], the number of Internet-connected devices and machines will reach 38.6 billion by 2025, with an economic impact between $3.9 trillion and $11.1 trillion. Even though IoT systems have economic advantages in transportation, energy, construction, and healthcare, deploying an efficient IoT platform becomes a bottleneck. This research study focuses on the context of Software as a Service (SaaS), and performance parameters include scalability, availability, throughput, and response time. The study [122] found that response time has a significant impact on cloud computing performance and availability. Increasing numbers of IoT applications connected to the cloud to take advantage of the flexibility, scalability, and minimal initial costs that cloud computing provides. The combination of local IoT resources with modern computing resources, such as edge and cloud computing, opens up

new research opportunities. As an illustrative example of the power of combining cloud computing and IoT, smart cities are considered a significant driver of healthcare and other industry development [90] because they rely on integrating multiple city systems such as transportation, healthcare, and operations research to provide a high quality of life for their residents. With advancements in data connection techniques, an increasing number of devices are getting closer to forming a system that significantly enhances the capabilities of devices through the combination of IoT and cloud computing. As a result, a framework that can use both cloud and IoT technologies is becoming increasingly important. This framework could be applied to IoT applications that require scalability, high availability and high performance.

The utilization of edge computing technologies, such as intelligent terminals, enables data storage and processing in real time. Compared to cloud computing, edge computing reduces the challenge of high energy consumption, saves costs, and reduces network bandwidth congestion. Many industries benefit from edge computing's capabilities, including manufacturing, energy, smart homes, and transportation [20]. Edge computing models are therefore important for the development of cloud-IoT applications.

Fault tolerance is the ability of cloud computing to continue providing services, even if one or more cloud components fail for any reason. Due to heterogeneity and its large-scale nature, cloud architectures have become more complex than traditional distributed systems. Many organizations plan to use the public cloud, but they are concerned about the reliability of the current cloud computing environment. Therefore, the need to design

and implement a reliable cloud computing environment has been increased because several real-time applications use the cloud and require high availability computing. Research focuses on developing a framework that can achieve the required level of reliability while meeting the business requirements of cloud consumers.

As demand for the cloud increases, the uptime of 24x7 services becomes one of the biggest challenges facing cloud providers, as it is challenging to escape service outages. Furthermore, cloud applications have a high probability of failure [107, 32, 103] because they run in a large-scale environment, including data centres and virtual servers. The reliability and availability of cloud computing remain the main concern of cloud consumers. For example, Amazon Web Services (AWS) has experienced a failure in one of its services, Elastic Block Storage (EBS). This failure brings down thousands of hosted websites and applications for 24 hours [64]. After certificates securing client data expired in February 2013, Microsoft's Azure cloud service faced a global outage [52]. In February 2017, AWS' Simple Storage Service (S3), which hosts entire websites and apps, broke down for four hours in the US-EAST-1 region due to debugging progress more slowly than expected [117]. Host applications and websites require fault tolerance to overcome the effects of failure and to perform their tasks correctly when failures occur. As a result, cloud providers focus on increasing the availability of services to ensure that cloud systems continue to function correctly in the event of faults.

This thesis investigates the impact of integrating local IoT resources with cloud and edge computing and presents approaches for increasing the reliability and availability of cloud-

IoT applications based on task failure analysis and prediction. This thesis also discusses two approaches to fault tolerance: task failure analysis and failure prediction. Failure analysis techniques help detect misconduct in operating cloud applications and determine the cause of software failure. However, the main aim of failure prediction is to detect task or job failures early before they occur to increase the performance of cloud applications and reduce the number of failed tasks. This thesis also provides an architecture for a highly reliable IoT application based on the Edge-Cloud architecture development and can support modern IoT applications [98].

1.2 Motivation and Problem Statement

The integration of IoT devices with the cloud offers many advantages, including extending the IoT resources and increasing the performance of cloud-IoT applications. Cloud computing can benefit from IoT devices and applications by extending its scope to include real-world surroundings. On the other hand, the IoT can benefit from the cloud's unlimited computing and storage resources. In many scenarios, cloud-IoT integration can provide an intermediate layer between IoT devices world and cloud services. However, many challenges face the integration of cloud-IoT, including reliability, availability, and performance. Thus, many modern applications, such as smart cities, smart homes, and eHealth, require a new architecture of cloud-IoT that can increase the reliability and availability and reduce the network latency of IoT applications. Also, while the number of cloud services has

increased, the cloud architecture has become more complicated due to its large scale and heterogeneity nature. As a result, cloud consumers are concerned about the availability and reliability of cloud services, as several cloud services, including software and infrastructure services, have recently experienced failures. In fact, the reliability and availability concerns have become one of the most significant challenges facing both the traditional High-Performance Computing (HPC) and the cloud environments so that the complexity of cloud architectures can increase the probability rate of failure [124, 34, 104].

In recent years, cloud service providers have encountered reliability-related issues similar to those we faced in the past. These challenges are power outages, unexpected hardware failures, failed deployments, software bugs, and human errors. The reliability and availability of cloud computing remain the primary concern among cloud consumers. The Coronavirus pandemic (COVID-19) has tested cloud providers in many ways, none of which could have been predicted. Although the public cloud has proven remarkably resilient in overcoming an unprecedented stress test, there are remarkable exceptions to cloud failure problems that occurred in the first half of 2020. For example, on March 3, 2020, almost all Azure services in Microsoft's East US region had a more than six-hour outage across most of its services, starting at 9:30 a.m., according to the Azure status history website [126].

Edge-Cloud architecture is urgently needed for IoT application development since cloud computing cannot meet the growing number of IoT devices and the data they generate and meet its QoS standards, including low latency. As the number of smart devices connected to the Internet is rising rapidly, leading to large-scale data; this causes several problems:

slow response times, poor security and privacy, high bandwidth load in traditional cloud computing models [27]. Advanced edge computing solutions have arisen because traditional cloud computing is no longer sufficient to meet the diversified data processing requirements of today's intelligent society. Edge computing, as compared to cloud computing, is closer to the user and the data source. Although many research studies focus on cloud-IoT integration in general, there is limited research investigating the reliability and availability of cloud-IoT integration. Our motivation is to provide a new framework for integrating the local IoT resources with the cloud. This thesis focuses on designing and implementing a framework that improves fault tolerance mechanisms in cloud computing and IoT platforms to enhance the reliability, availability, and performance of cloud-IoT applications.

The main challenge that needs to be addressed at an application level is handling tasks between local IoT resources and the cloud. Our goal is to distribute resources efficiently in order to develop a scalable and available cloud-IoT framework. To achieve this goal, we focus on two aspects: providing a computational mechanism that reduces the response time of connected devices while increasing throughput.

Failed tasks consume considerable computational resources, including memory and CPU. Cloud resources are therefore wasted as the number of failed tasks increases. Many previous studies [48, 104] have analyzed and characterized the workload features of Google traces [106]. The most recent studies have focused mainly on analyzing and studying failure behaviour, while there are limited research findings in the development of job failure prediction models [46, 32, 112, 107]. Thus, the design and implementation of failure prediction

models using machine learning (ML) and deep learning (DL) algorithms is an additional challenge that should be taken into account.

1.3 Research Objectives

The main objective of this thesis research is to design and implement a cloud-IoT framework that delivers high reliability and availability for IoT applications through the use of proactive fault tolerance techniques. The framework aims to decrease the number of task failures and minimize the time and cost of using the cloud.

The objectives of this thesis are:

1. Investigate job and task failure in the cloud to study the behaviour of failed tasks and examine the correlation between failed jobs and requested resources.

2. Design a framework for extending local IoT resources using cloud and edge computing in order to provide scalability and high availability for cloud-IoT applications using an offloading technique that can minimize the cloud's execution time and computation cost.

3. Design a failure prediction model for cloud-IoT applications that answers the following question: how can we accurately classify and predict incoming task event types (fail, success) before the management system schedules them on the cloud?

1.4 Research Contributions

This thesis provides new techniques which improve the reliability and availability of cloud-IoT applications based on the design and implementation of failure prediction models. Also, the proposed framework provides an offloading technique based on a scheduling algorithm for distributing incoming tasks between local IoT resources and the cloud. In summary, the main contributions of this thesis are:

- Design and implementation of failure prediction models based on machine learning and deep learning approaches.
- Study and analyze the characterization of large-scale workloads based on understanding the failure behavior and the correlation between failed tasks and cloud attributes.
- Design and implementation of a framework for extending local IoT resources using the cloud. This framework can be used in IoT applications that require high performance and scalability.
- Design and implement an offloading technique that transfers incoming tasks based on the required computation and priority levels.
- A highly reliable and available design of IoT applications based on the development of Edge-Cloud architecture to support modern IoT applications. The Edge-Cloud architecture has been evaluated in terms of response time, bandwidth, VM CPU utilization and task failure rate.

1.5 Thesis Outline

The remainder of this thesis is organized as follows. Chapter 2 presents the background and related work to provide an overview of previous research. Chapter 3 illustrates the proposed framework for cloud-IoT integration, failure analysis, and failure prediction. Chapter 4 reports and discusses the results and evaluation. Finally, Chapter 5 concludes the thesis and presents some future research directions.

Chapter 2

Background and Related Work

This chapter presents an overview of the leading research areas related to this thesis, namely, cloud computing, edge computing, the Internet of Things (IoT), and fault tolerance. Furthermore, this chapter reviews previous studies on failure analysis and failure prediction in the cloud and other computing research areas such as HPC, and virtualization.

2.1 Modern Compute Resources

The IoT, cloud computing, edge computing, fog computing, and mist computing [83] have received much interest in academia and industry in recent years. However, a clear and direct definition of these computing paradigms and their relationships is difficult to find in

the literature. Thus, this section will discuss each paradigm in detail, outlining its essential characteristics and their relationship to the others.

2.1.1 Cloud Computing

Cloud computing enables consumers to access, configure and manage cloud resources, including software and hardware services via the Internet. Visualization is a key component of cloud computing because it allows users to access resources more rapidly and reduce costs. Cloud computing offers scalability, management and availability, as well as cost-effectiveness, on-demand service, convenience, multi-tenancy, and elasticity.

The National Institute of Standards and Technology (NIST) has published the popular definition of cloud computing, which is: "*Cloud computing is a model for enabling convenient, on-demand network access to a shared pool of configurable computing resources (e.g., networks, servers, storage, applications, and services) that can be rapidly provisioned and released with minimal management effort or service provider interaction. This cloud model promotes the availability and is composed of five essential characteristics, three service models, and four deployment models [84]*". As shown in Figure 2.1, there are three different service types and four different development models for a cloud computing environment. Cloud computing primarily provides three service models: Infrastructure as a Service (IaaS), Platform as a Service (PaaS) and Software as a Service (SaaS). NIST defines four cloud development models: private, public, community and hybrid [123].

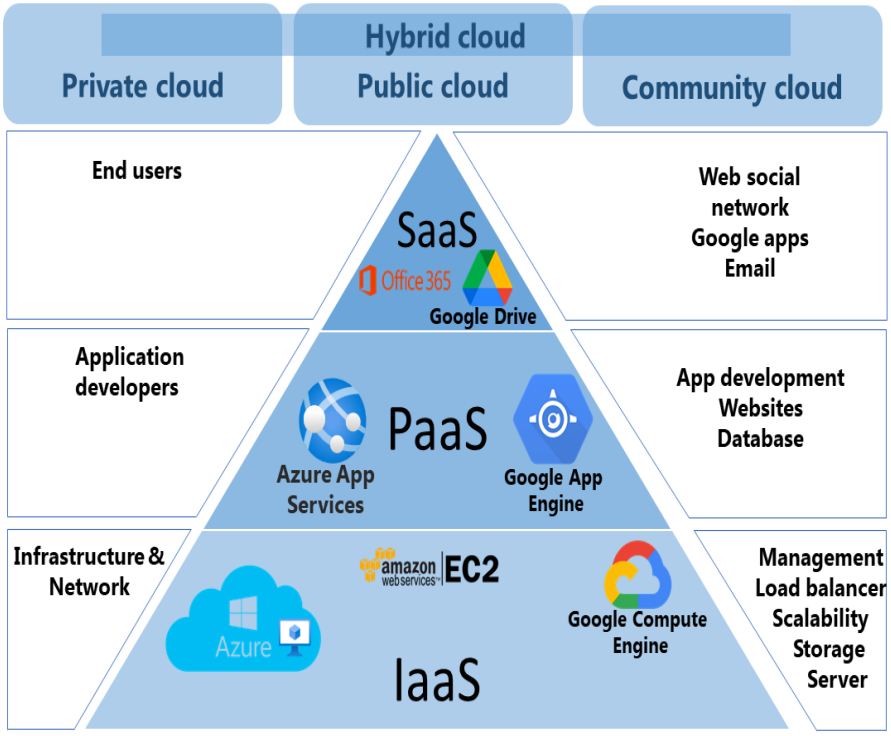


Figure 2.1: Types of cloud services and deployment models

The public cloud becomes one of the best solutions for increasing scalability and availability as well as decreasing the cost-effectiveness of business applications. Recently, many organizations have migrated to the public cloud because they can eliminate or decrease the size of their infrastructure. The reduction of physical servers and software costs can significantly reduce IT costs [47].

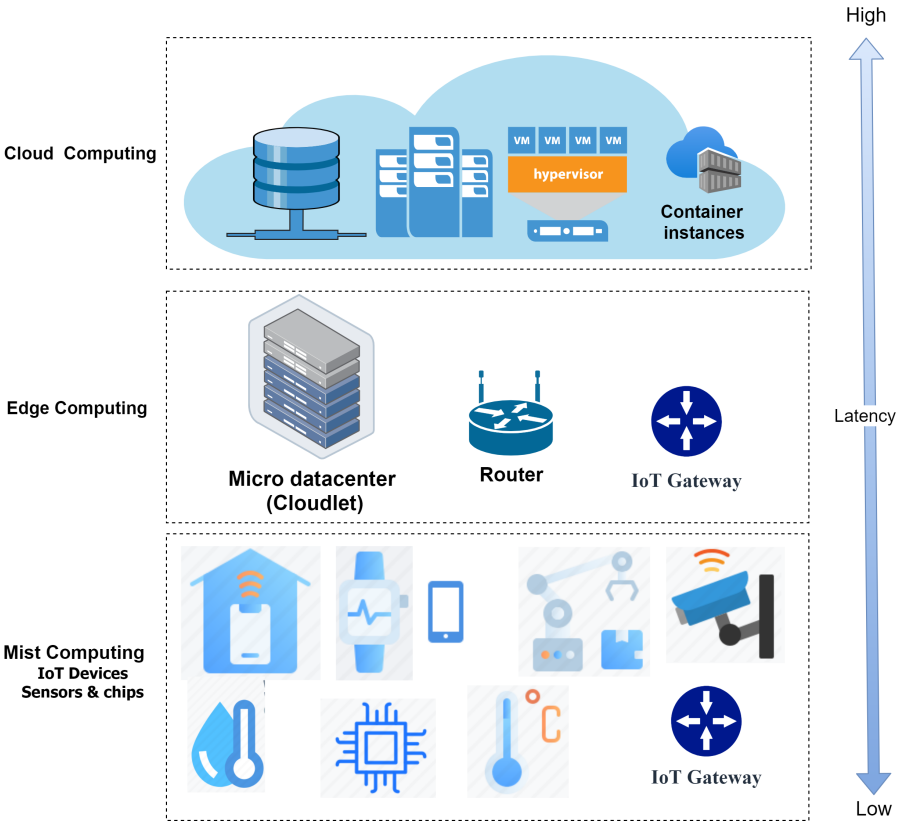


Figure 2.2: The architecture of cloud, edge, mist computing

2.1.2 Edge, Fog, and Mist Computing

This section explains the differences and comparisons between edge, fog and mist. The use of these computing terms has been increasing in recent years, but they are still not well understood by many people. The edge, fog and mist computing components are shown in Figure 2.2.

Edge Computing

Data can currently be processed using edge computing on devices connected to sensors or gateway devices near sensors. As a result, edge computing enables devices to process data without the assistance of the cloud or fog. Edge computing allows devices to process data in near real-time by bringing data processing closer to the edge; this reduces centralized cloud overheads [116]. In a connected smart home, edge computing can be used to perform near-real-time operations, such as turning on lights. Additionally, edge computing may aid in predictive maintenance by providing timely alerts when a device is going to fail. Edge computing is suited for addressing privacy, latency, and connection issues. However, the response time in edge computing might be higher than in the fog or cloud if the local processing unit is not powerful enough [134]. As a result, edge computing can cut service costs and latency while simultaneously reducing power consumption by being closer to IoT devices [6]. As shown in Figure 2.2, an IoT gateway serves as a network router, allowing data to flow between IoT devices and the cloud. A new model of IoT gateway devices is designed to handle both inbound and outbound traffic. Some IoT gateways are designed to pre-process data locally at the edge rather than transmitting it to the cloud.

Computing at the edge could improve internal communication by combining physical assets with IoT devices to collect and evaluate vital data. Data can be saved locally and transferred to the cloud for further analysis after being processed by IoT devices, so it is important to note that sensitive data can be safely handled at the point of origin.

Edge computing devices can also perform near-real-time analytics that can help to optimize performance and uptime. The primary downside of edge computing is that it is less scalable than cloud computing.

Task offloading can be accomplished via user devices, edge devices, and edge nodes. Offloading is used to deal with a variety of problems that impact on optimization. Data management, application computing restrictions, latency control and energy management are some of the challenges that need to be addressed. If fog nodes are not performing well on their primary resource, offloading is utilized to move the task to another resource. Edge-Cloud applications could also be improved by using offloading strategies. When a low-speed CPU cannot accomplish a task, it may be offloaded to a high-speed processor [119, 1].

Fog Computing

Fog computing is a network architecture that extends from the point of data creation to the point of data storage in the cloud or the organization's data centres. Gateways, routers, and cloud services are among the components of the fog computing architecture. A computational layer that functions between the cloud and the edge is also known as fog. Fog computing extends the cloud to the network edge and enables decentralized computing by processing data on the fog node. For this purpose, any device with storage, computation and network connectivity can be used as a fog node. Fog computing could be useful in smart cities, where many devices rely on real-time data to perform various functions.

For example, autonomous vehicles use fog computing because real-time data processing is essential. The downside of fog computing is that it relies on many connections to send data from the physical asset chain to the digital layer, which leads to network failure.

Mist Computing

Mist computing can be used to process data at the extreme edge of a network, where micro-controllers and sensors are present. Mist computing can harvest resources by operating at the extreme edge of the sensor and using the processing and communication capabilities of the sensors. In the field of mist computing, data is transferred to fog computing nodes and then to the cloud using micro-controllers and microcomputers. Many edge nodes, including mobile phones, connected vehicles, and intelligent home appliances, have become mist computing components that process data at the extreme edge.

One example of the value of mist computing is the deployment of a network of real-time cameras using a machine learning model. A private mesh computer can use a mist node to transmit live streaming videos to other camera mist nodes. The mist nodes with static video frames may help other mist nodes with moving video frames in computing machine learning algorithms that check for human counts, behaviours, guns, etc. For example, when a gun is detected, the mist can begin to transmit frames to cloud-based mist nodes with face detection databases to identify criminals in the live video stream. The distinctions among cloud, fog, edge, and mist are summarized in Table 2.1.

Table 2.1: The differences among cloud, fog, edge, and mist

Cloud computing	- Central processing based model. - Highly scalable and unlimited storage space. - High latency and response time. - High power consumption. - Downtime
Fog computing	- Extending the cloud to the edge of the network. - Decentralized computing. (millions of small nodes) - Fog is a computational layer that functions between the cloud and the edge. - Fog computing relies on several links to transport data from the physical resource to the cloud, which might fail. - Fog does not have the same scalability as the cloud. - Fog nodes can be public, private, or hybrid.
Edge computing	- Edge computing, unlike fog computing may operate without the use of a cloud or fog. - Close to the source of information. - Low latency. - Less scalable than fog computing.
Mist computing	- Utilization of microcontrollers and microchips to provide lightweight computing within the network. - Local Decision-making data.

2.2 Internet of Things

IoT refers to the use of intelligent devices that are connected to the Internet. The essential features of the cloud-IoT application architecture are connectivity, sensing, scalability, intelligence and integration. IoT devices can be connected via various types of network communication, such as Bluetooth, Wi-Fi, and radio waves. IoT devices should be designed and implemented to be scaled down or scaled up based on business and application requirements. According to [16], the study estimates that by 2025, the number of Internet-connected devices and machines will reach 38.6 billion. It is expected to have a total economic impact of between $3.9 trillion and $11.1 trillion. IoT systems have economic benefits in transport, energy, construction and health.

On the other hand, cloud computing has faced numerous obstacles due to its rapid growth and remote geographical location. Cloud computing cannot support the growing number of IoT devices and their generated data while maintaining QoS standards such as low latency. As more intelligent devices connect to the Internet, traditional cloud computing models suffer from poor response times, a lack of security and privacy, and high bandwidth usage [27]. Advanced edge computing solutions have emerged as standard cloud computing no longer meets the variety of data processing demands of today's intelligent society. Compared to cloud computing, edge computing is closer to the user and the data source.

With the development of IoT applications, edge computing models are urgently needed. The impact of IoT and edge computing integration on failure analysis and prediction is also addressed in this thesis. In addition, we present an architecture for a highly reliable and available IoT application based on the development of the Edge-Cloud architecture, which can support a new model of IoT applications [98].

2.3 Fault Tolerance

A fault, error, and failure are related abnormal conditions in a system. The formal definitions of fault, error, and failure are [19]:

- Fault: A fault or bug is defined as a defect or abnormal state. This fault can affect one or several parts of a system, and it may be consequences in preventing the system

from performing a specific function.

- Error: Incorrect behaviour caused by a fault (a manifestation of a fault).
- Failure: A failure occurs when the system cannot deliver its intended function.

The methods of fault tolerance are divided into two categories:

- ***Proactive fault tolerance*** prevents fault, error, and failure based on applying prediction methods such as *software rejuvenation, self-healing, and preemptive migration* [118]. The main objective of proactive fault tolerance is to discover the failure before it occurs.
- ***Reactive fault tolerance*** is attempting to reduce the impact of failure after the failure effectively occurs. Several techniques can be applied as reactive fault tolerance, such as *check-pointing, replication, job migration, and Retry* [12, 10].

2.3.1 Fault Tolerance Elements

Dependability includes reliability, safety, resiliency, and redundancy. The term reliability refers to the capability of cloud components to perform without failure consistently. The following sections will define these elements and explain how they can affect the system's dependability [23, 69]. Figure 2.3 presents the correlation between fault tolerance elements and system dependability.

Figure 2.3: Fault tolerance elements

- **Reliability**: Reliability refers to the capability of a specific system to be able to operate correctly over a period of time. For a system, reliability can be calculated by determining the system's probability of functioning correctly for a given time interval t [23].

- **Safety**: Safety can be defined as a feature of the system that does not cause harm to humans or damage the environment.

- **Resilience**: The main objective of resiliency is to increase the ability of the cloud management system to recover different types of failures and continue operating without bringing down the entire system. Applying resilience techniques helps to respond to the failures and to avoid data loss and downtime. High Availability (HA) and Disaster Recovery (DR) are the most significant aspects of resiliency [23, 69].

  - **High Availability**: Many cloud consumers are concerned about the availability of cloud computing. HA refers to the ability of the cloud component to continue functioning in an active state without unacceptable downtime. We can also express the availability in terms of average downtime. When we design applications to be reliable and resilient, we should consider the availability

requirements and the acceptable amount of downtime. Business owners can answer this question when they investigate the cost of downtime. Based on the business requirements, owners can decide how much they should invest in order to ensure that the application has high availability. The following formula is commonly used to calculate the availability of a system or application:

$$Availability = \frac{Uptime}{Uptime - Downtime} \tag{2.1}$$

The formula returns a percentage value, for example, 99.99 % ("four nines") or 99.999 % ("five nines").

The agreed-upon service time is the system's estimated operational time every month. The system's scheduled downtime is specifically excluded from the agreed-upon service time.

– **Disaster Recovery (DR)**: Cloud DR is a cloud-based solution that enables the backup and recovery of remote devices. In other words, DR is the system's capability to recover main incidents such as wide-scale and non-transient failures such as service disruption that can affect an entire region. Disasters include practically any circumstance that causes systems in a data center to fail, such as fires, equipment failures, and weather conditions. Many approaches are used in disaster recovery to reduce the impact of failure, including archiving and data

backup.

Recovery Time Objective (RTO) and Recovery Point Objective (RPO) are two important metrics related to business requirements. RTO is calculated based on the maximal acceptable time that a cloud component can be unobtainable after an incident, and RPO is the maximal time of data loss that can be acceptable during a disaster. For any reason, if the Mean Time to Restore (MTTR) surpasses the RTO, this failure will lead to unacceptable business disruption [23, 69].

- **Redundancy**: Redundancy refers to the duplication of components. The main objective of resiliency is to bring back the cloud application to a complete health state after an incident occurs. The more redundancy there is in the components of a system, the more resilient it will be.

In cloud computing, a failure occurs for any reason, such as hardware failures or transient network failures. Additionally, a disruption may affect an entire region in some instances. Consequently, we must develop methods to avoid or mitigate the impacts of these failures. Recent research focuses on minimizing the Mean Time to Restore (MTTR) instead of preventing failures and optimizing Mean Time Between Failures (MTBF). Focusing on MTTR helps to reduce the impact of the failure. Many advantages of designing and implementing fault tolerance techniques include failure recovery, improved performance, and lower cost.

2.3.2 Fault Tolerance in the Cloud

There have been several studies that present surveys of fault-tolerance in the cloud computing environment. The authors in [55, 56, 92] have discussed various fault tolerance approaches and different fault types in cloud computing. All resources are offered as cloud services in cloud computing and include SaaS, PaaS, and IaaS [17, 84]. The architecture of cloud layers should be designed and implemented with advanced fault tolerance techniques in order to reduce the probability of failure that can occur in one or more cloud components. Figure 2.4 presents the fault tolerance cloud architecture that includes all cloud service layers with the types of faults that can occur at each layer. A failure may occur suddenly, and impacting the cloud service layers offered to consumers by cloud providers. For instance, a fault in a PaaS layer might generate errors at the application level or SaaS, which is running on top of PaaS. Cloud providers offer cloud services that run in different data centers, and these data centers operate in various regions. A single data center includes different cloud layers that could produce different fault types. Thus, the new architecture of cloud computing requires varying levels of fault-tolerant methods to ensure that cloud services are reliable.

Fault Tolerance for SaaS

SaaS provides remote access to software applications and their functions as a web-based service. Also, in order to build SaaS, it is important to take into consideration three

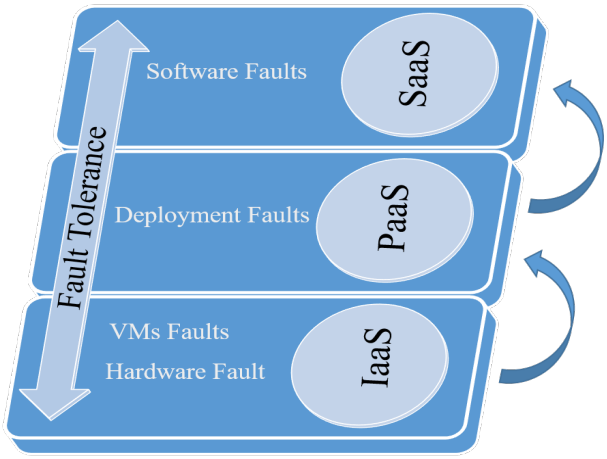


Figure 2.4: Fault tolerance cloud architecture

aspects: SaaS providers, cloud providers, and SaaS users [17]. The SaaS fault types can be classified into two fault types: a software service fault and a third-party software fault. If SaaS has any types of failure, it causes immediate effect for users. In order to have a high quality of service, the most important aspect is to ensure that all functions continue to work correctly even though a fault suddenly occurs. Correctness is necessary when users expect specific results from the software.

Reliable software can be defined as systems that execute their functions correctly during a particular period and meet the users' expectations. One of the main parts of software engineering is software reliability, and the primary roles of software reliability are measuring, analyzing, and improving the reliability of software systems.

Tsai et al. [129] proposed an adaptive test configuration to identify different types of faults in the SaaS layer. In [53], the authors presented a method that can be used

to detect silent failures. These types of failures may produce high impacts such as silent data corruption and silent data loss. Their proposed solution has been designed using the combination of invariant violation checks and FSM analysis in order to detect silent failures. Samir [114] presents the common important attributes for SaaS; the author also extracts the critical SaaS quality attributes based on business values and cloud providers. Reliability is one of the essential characteristics of SaaS quality. Reliability is one of the most important characteristics of SaaS quality. Alannsary and Tian [9] developed a technique for predicting the reliability of SaaS by taking advantage of analyzing web server logs. The Cloud-ODC model has been designed based on three phases: the first phase is the data source analysis, the second phase is the classification, and the final phase is the result analysis. The authors of [43] examined Service Level Agreement (SLA) violations on a production SaaS platform, identified the underlying causes, identified numerous important failure modes, and then proposed various solution options to increase the perceived availability of the platform for end-users. According to Pham and Roy [109], a forecasting technique was proposed to examine and improve the reliability of web applications. Kallepalli et al. [67] developed web testing and reliability techniques in order to ensure the quality of web services. Cotroneo et al. [38] proposed an approach that uses application logs to detect the reasons behind the failure. However, mining interleaved logs for underlying distributed system infrastructure should be considered throughout the analysis phase. Abderrahim et al. [2] propose an architecture that aims to ensure that the cloud providers have successfully achieved the requirement of dependability properties for cloud services.

Fault Tolerance for PaaS

PaaS clouds allow consumers to develop, run, and manage their applications without concerns about the infrastructure's design, maintenance, and management. PaaS services provide a ready platform to developers, including servers, networks, operating systems, storage, programming languages run time, web server and other services that help developers host their applications. Thus, developers only need to deploy their applications and apply the configuration settings. Even though PaaS services provide an easier and faster application deployment and built-in function for horizontal scalability, the current deployment mechanisms have become insufficient with critical applications. Critical applications require high availability, disaster recovery, and high performance. Li and Kapitza [72] present BFT-DEP a framework for automatically deploying Byzantine Fault-Tolerant (BFT) services in a PaaS cloud. The main objective of their framework is to ensure that all deployment requirements have been achieved. They designed the BFT protocol to be integrated with the cloud platform as a built-in service. Addo et al.[3] present a proposed solution for implementing automatic fail-over between PaaS cloud provider platforms.

Fault Tolerance for IaaS

IaaS offers a pool of cloud computing resources, including hardware, servers, networking components, and vast storage spaces. The essential IaaS components are physical servers, including virtual servers, CPU, memory, and storage. Different types of faults may occur,

including physical server faults, VM faults, and container faults. One of the major challenges in cloud computing is maintaining the availability of cloud services. Nabi et al. [94] described the issues and challenges related to the upgrade of cloud computing in terms of availability. Wang et al. [132] proposed a mechanism that provides high availability for software services running on the VMs. Muthumanikandan et al. [93] developed a Switch Failure Detection (SFD) technique in order to detect switch failures in Software Defined Networking (SDN). When a link fails, SFD determines whether it failed as a single link or as a result of a subsequent switch failure. The proposed method reduces the amount of time required for other links connected to the failed switch to be detected, and their proposed technique also boosts the network's throughput.

Many commercial virtual servers support a high availability technique such as VMware [24]. VMware uses heartbeats in order to detect VM failures. Wang et al. [132] use three different approaches to detect VM availability based on physical servers: a *FRU check*, a *watchdog-timer check*, and a *sensor check*. The FRU is used to check if a server exists and investigate the server status in terms of the stage of booting, serving, or turning off. The sensor check verifies whether the temperature and the voltage of the servers are in a normal state. The watchdog-timer is responsible for diagnosing the health status of the servers.

2.4 Dependability of Cloud Computing

The capability of a system to avoid service failures that are more frequent and more severe than is tolerable is referred to as dependability. Dependability measures a system's reliability, availability, durability, and, in some situations, maintainability, safety, and security. Dependability in real-time computing refers to delivering services that can be trusted across time [69].

Jhawar et al. [65] introduce an approach for creating and managing fault tolerance in cloud computing. The main feature of this approach is to enable cloud consumers to select and apply the desired level of fault tolerance based on system requirements without requiring knowledge of fault tolerance techniques and implementation.

Zhao et al. [138] present middleware called Low Latency Fault Tolerance (LLFT). The primary goal of their approach is to provide fault tolerance for distributed applications deployed within a data center running in the cloud. Their approach has been implemented using the replication technique. Other researchers also provide fault tolerance techniques for web services [111, 136, 18]. Meshram et al. [86] propose a model called Fault Tolerance Model for Cloud Computing (FTMC). The primary function of the FTMC model is to tolerate the failures of each computing node. If a computing node does not perform well for applications, it will be deleted.

Cloud computing architecture focuses on three important aspects: computing, storage, and network services. Most public cloud providers offer computing resources as Infrastruc-

ture as a Services. The main objective of providing these resources to cloud consumers is to deliver the required infrastructure and assist in deploying applications faster without concern in maintaining the infrastructure. AWS [22], Microsoft Azure [25], Google, and IBM SoftLayer provide persistent storage for storing virtual machine data. Furthermore, the cloud providers offer multiple availability sets and regions across the world to increase dependability and disaster recovery capability. AWS and Google cloud providers offer a monthly uptime to cloud consumers of at least 99.95% using two or more virtual machines. However, guarantees are not provided for using only one virtual machine. Microsoft Azure also provides a high SLA rate with 99.95% uptime for at least one of all virtual machines running on different availability sets. Microsoft Azure also offers 99.9% of uptime for any single virtual machine that uses premium storage for all data disks and operating system disks. IBM SoftLayer does not provide a specific SLA rate for the virtual server in terms of storage, but it offers a service level agreement of 100% for a public network. In terms of public cloud storage, we have studied three different storage types: file storage, block storage, and object storage. AWS, Microsoft Azure, and Google clouds offer file storage services. These cloud providers allow cloud consumers to access their file storage via API. However, the protocols used for connections differ from one cloud provider to another. For example, AWS provides Elastic File System (EFS) using NFSv4 while Microsoft Azure provides a file storage service called Azure File Storage (AFS) based on the standard Server Message Block (SMB) protocol. Other features such as snapshot, replication, high availability, and flexible customization and configuration are similar among all public cloud

Table 2.2: SLA for public cloud storage

	AWS	Azure		Google	IBM
File Storage	no guarantees of uptime	Read Availability 99.99% for RA-GRS 99.9% for LRS, ZRS, GRS	Write Availability 99.9% for RA-GRS, LRS, ZRS, GRS	Availability 99.95% for Multi-Regional 99.9% for Regional Storage 99.0% for Nearline, Coldline	no guarantees of uptime
Block Storage	99.95% SLA				
Object Storage	Durability 99.999999999% Availability 99.9% for Standard 99.0% for Standard IA				
Backup and Archive	Durability 99.999999999%	99.9% SLA		99% SLA	no guarantees of uptime

providers.

The SLA describes availability for uptime and connectivity. If the SLA for a cloud service is 99.9%, this service must be available 99.9% of the time. In order to achieve four 9's, the cloud provider should take into consideration to design the cloud components to be self-diagnosing and self-healing instead of using manual intervention to recover from failures. Cloud providers recommend especially for critical applications to have several redundant components deployed across several data centers to increase the application availability by failing over if one of the data centers failed. However, deploying components or hosted applications in multiple regions is very expensive. Thus, if high availability is not one of the business requirements, the application should be running in a single data center. Table 2.2 presents the reliability and SLA for public cloud storage.

2.5 Extending Local Resources

In their study, Lee et al. [71] have addressed the issues of meeting dynamic computational needs in a distributed environment. A key shortcoming in the current sensor network

approach is a limitation of elasticity in sensing and local resources. The study team has demonstrated that Amazon Elastic Compute Cloud (EC2) can meet the system requirements. The evaluation of their paper proves that unpredictable computational demands that are generated by real-world environmental sensing and monitoring applications can be dealt with sufficiently elastic cloud computing. Cloud-IoT integration has been extensively studied in terms of its integration applications, challenges and issues [79, 102].

Marshall et al. [81] have proposed a model of an elastic site. They extend the local site cluster with cloud services and resources to solve the local resources' limitations and adapt to the dynamic demands of new model applications. The core component of this model is an elastic site manager, which is responsible for resource provisioning. In order to implement and evaluate the elastic site model, the authors applied different types of technologies and tools such as the Nimbus toolkit and Amazon Web Services (AWS).

One of the most significant features of using the cloud is providing high availability and scalability. Thus, applying the load balancing techniques in the cloud environment plays an essential role in increasing the availability and performance of cloud applications. The load balancing technique has been explained in a white paper by Adler [4]. The paper presents the techniques and tools that have been commonly used for applying load balancing in the cloud environment. However, the author states that the load balancing technique still has some challenges in adapting to the many changes in the cloud. Zenon et al. [28] illustrate the important role of load balancing to improve availability and performance. Randles et al. [101] investigate three different distribution methods that could be used for load

balancing. These solutions are Biased Random Sampling, Honeybee Foraging Behavior, and Active Clustering. In [87], various performance parameters for load balancing are explained in detail. DCBT is a hybrid scheduling algorithm proposed by Shridhar and Ram [45] that combines the Divide-and-Conquer and Throttled algorithm methodologies. Their algorithm efficiently distributes the incoming load to maximize resource utilization in a cloud environment.

2.6 Failure Analysis and Prediction

This section discusses current literature studies that are relevant to this thesis research. Based on the study areas in task failure analysis and prediction, we divide the related work into the following three subsections:

2.6.1 Statistical Analysis

A forecasting technique based on Generalized Autoregressive Conditional Heteroscedastic (GARCH) and Autoregressive Integrated Moving Average (ARIMA) models were presented by Amin et al. [11] to estimate the time between failures and response time in web services. Khan et al. [68] found the repetitive workload patterns of virtual machines. They then developed a technique based on Hidden Markov Modeling to describe and forecast the workload patterns of virtual machines. Zhao et al. [140] approach the topic of disk failure prediction from a completely different perspective than the previous researchers. They use

different characteristics measured at successive time intervals for a disk drive as a time series, and they utilize Hidden Markov Model (HMM) and Hidden Semi-Markov Model (HSMM) to model such time series in order to identify "failed" disks from "good" disks. Morais et al. [88] developed a framework for the development of auto-scaling services based on a variety of CPU usage prediction algorithms, including Linear Regression (LR), Auto Correlation (AC) and Autoregressive Integrated Moving Average (ARIMA). Moreover, a pattern matching and state-driven technique were used to estimate workloads by Gong et al. [54] in order to build the workload prediction system called Predictive Elastic reSource Scaling (PRESS). It begins by employing signal processing techniques to determine whether or not the CPU used in a virtual machine displays recurrent activity patterns. If the answer is yes, the repeated patterns are utilized to estimate future workloads; if the answer is no, PRESS utilizes a statistical state-driven technique. A discrete-time Markov chain is used to predict demand for the upcoming few days or weeks.

2.6.2 Failure Analysis

Failure analysis and characterization have been extensively researched in cloud computing, grid computing, and supercomputers [32]. Some studies focus more on characterizing the reliability of cloud computing hardware. For example, Vishwanath et al. [130] presented a detailed analysis of failure predictors and failure characteristics. The main objective of their study is to understand the hardware dependability in the cloud computing environment. They found that hard disks are one of the less reliable components, and approximately 8%

of all servers can expect a minimum of one hardware failure per year.

Pan et al. [96] propose an approach called Ganesha, a black-box diagnosis technique that utilizes the operating system metrics in order to identify and diagnose faults in MapReduce systems. Zhang [137] studied the workload characterizations that are generated by applying realistic performance benchmarks in order to evaluate the performance impact of the cloud system changes. Fadishei et al. [48] examine workload characteristics such as CPU speed and memory usage, task execution time, and other monitoring data. They discover a relationship between unsuccessful jobs and workload characteristics.

The Google cluster traces [106, 105] are used in different research studies, including workload trace characterization [32] and applying statistical methods in order to compare Google datacentres – a cloud environment – to Grid or HPC systems [42].

Reiss et al. [104] analyze the Google dataset to highlight the big-data trace's heterogeneous and highly dynamic behaviour. However, this study does not consider job failure analysis and prediction, so their work is limited to the general analysis of the Google cluster trace. Liu et al. [76] studied and analyzed the Google traces, including the distribution of machines. The authors have summarised statistical data relevant to tasks, jobs, and machine events. Garraghan et al. [50] have used the same set of Google traces to analyze the server characteristics and resource utilization on the Google platform. They also study the impact of tasks that are terminated before completed in terms of wasted resource utilization per server. Mesbahi et al. [85] offer a dependability study and a Markov model based on Google cluster usage traces. They also have investigated the reliability of Google cluster

traces using various physical machine failure rates and characteristics, such as steady-state availability, Mean time to repair (MTTR), and Mean time to failure (MTTF). Ruan et al. [110] have introduced a comprehensive multi-view approach to comparing two cloud workloads and conducted a case study on Alibaba and Google cluster traces. Ahmed et al. [5] have utilized the Google cluster workload to discover the distribution function for the time to repair and the time to failure for the cloud servers.

Some studies have used unsupervised learning in machine learning in order to characterize cloud applications based on jobs and tasks events [41, 8]. Di et al. [41] have identified cloud apps based on task events and resource utilization using the K-means clustering technique with an optimal number of sets. The number of applications in the K-means clustering sets is distributed in a comparable way to the Pareto distribution. Alam et al. [8] conducted a statistical analysis of resource usage and workload traces. Although there has been much earlier workload analysis of Google clusters, the key contributions of this study are Google workload patterns clustering and job categorization based on K-means clustering.

Amvrosiadis et al. [15, 13] have introduced four novel traces, two from private clusters and two from HPCs. According to their findings, workloads in private clusters, consisting of data analysis tasks that are likely to be more closely linked to Google's workload, are more similar to those in HPC clusters. This observation shows that other cloud traces should be considered when evaluating the generality of new findings. Their new traces include two from Two Sigma's private cloud and two from Los Alamos National Laboratory's

high-performance computing clusters (LANL) [14].

In [59], we examined workload characteristics such as memory utilization, CPU speed, and storage space. We observe a direct correlation between unsuccessful jobs and workload characteristics. Additionally, there is strong evidence that killed and unsuccessful tasks utilized a sizable part of cloud resources. A small percentage of failed tasks were resubmitted several times in an attempt to complete them successfully. However, because these unsuccessful tasks consumed many resources, they were classified as killed jobs. Moreover, all jobs with a scheduling class of (3) failed. This issue demonstrates a direct relationship between the scheduling class and failure.

2.6.3 Failure Prediction using ML and DL

Earlier research on job failure has focused mainly on the study and characterization of failures. However, some studies have been published on predicting job or task failures [73, 107, 46, 32, 120, 125, 7].

Samak et al. [113] have applied the Naive Bayes classification algorithm to the execution logs of scientific processes to predict task failure. Then, they have shown in some cases that when an incoming task that is predicted as failed, it can be successfully scheduled to a different available resource. Liang et al. [73] utilize log files from IBM's BlueGene machine to predict failure based on investigating the characteristics of fatal failure events and finding the correlation between non-fatal and fatal events. Bala and Chana [21] proposed

architecture for task failure prediction using data analysis and machine learning algorithms to classify their approach under proactive fault tolerance techniques. Thus, their approach is applied during the execution time of the applications before the failure occurs. El-Sayed et al. [46] have designed a job failure prediction model based on the Random Forest (RF) machine learning algorithm. The results show that they can successfully recall up to 94% of all failed jobs with at least 95% precision. Rosa et al. [107] have created an approach for predicting the outcome of task events. Their methodology is based on the integration of three distinct algorithms: Quadratic Discriminant Analysis (QDA), Linear Discriminant Analysis (LDA), and Linear Regression (LR). Their proposed model has been designed to be updated every 24 hours. Shetty et al. [120] have proposed a model based on the XG-boost classifier that uses three distinct resampling strategies. They obtained an accuracy of 92% and a recall of 94.8%. Machine learning classifiers were tested to predict job failures by Hongyan et al. [57]. They evaluated the performance of four different algorithms: RF, KNN, KDT, and LR. In order to test the classifier's accuracy, the OpenCloud dataset is used. Additionally, Sun et al. [127] used a deep learning model to predict a software failure. They used a mechanism for creating new samples to generate failure data. Different machine learning classifiers were used by Padmakumari and Umamakeswari [95] to predict task failure in scientific applications. Classifiers were trained and tested on a simulated dataset. The results reveal that the NB classifier has a high degree of accuracy (up to 94.9%).

Deep learning has been used by Gao et al. [49] to predict job and task failures. They

used the Bidirectional Long Short Term Memory (Bi-LSTM) model with many layers. In order to model training and testing data sets, the authors extract static attributes and generate dynamic ones. According to the results, Bi-LSTM predicted task failure with an accuracy of up to 93% and job failure with an accuracy of up to 87%. Moreover, Islam and Manivannan [58] found important attributes related to cloud application failure. In order to predict the application's termination state, they used the LSTM model. The results indicate that LSTM achieves up to 87% accuracy.

The primary shortcoming of earlier work is that the majority of previous research assessed their models against a single classification method without comparing them to other classifiers to guarantee that their results were accurate. As a result, we used a variety of classification techniques, including Decision Trees (DTs), RF, K-Nearest Neighbor (KNN), XGBoost, Naive Bayes (NB), Gradient Boosting and Quadratic Discriminant Analysis (QDA). Then, we determined the optimal model based on applying various assessment criteria and feature selection strategies. In previous work, we have [62] proposed a failure prediction model based on an RF classifier and enhanced the model's accuracy using several feature selection algorithms. Our failure prediction model outperformed prior work in [46, 125]. Table 2.3 and Table 2.4 summarize the current state-of-the-art in cloud computing for failure analysis and prediction. In comparison with the previous studies, we have achieved the highest precision, recall, F1-score, and we have applied different evaluated methods to ensure that our failure prediction model is accurate.

Table 2.3: Summary of the state of the art in the field of cloud computing for failure analysis and prediction

Related work	Trace	Focus	Model	Results
Chen Xin et al.[34]	Google trace	Predicting job failures	Deep Learning (RNNs)	Accuracy(82%) Recall(86%)
Garraghan et al.[51]	Google trace	Study server characteristics and server resource utilization	Statistical Analysis	X
Pan et al.[96]	Google's MapReduce[39]	Identify and diagnose faults in MapReduce systems	Ganesha's diagnosis algorithm	X
Chen Xin et al.[32]	Google trace	Study characteristics of failed and killed jobs	Statistical Analysis	X
Di Sheng et al.[41]	Google trace	Understanding of the characteristics and behaviors of cloud applications, and classifying cloud applications	K-means	X
Lu et al.[78]	Alibaba	Understand machine characteristics and workload behaviors	Statistical Analysis	X
Liang et al.[73]	Log files from the BlueGene machine of IBM	Predict failure based on investigating the characteristics of fatal failure events	Bursty nature of failure occurrence, spatial skewness, and preceding non-fatal events	X
Zhang [137]	Google's Compute Clusters	Study the problem of deriving characterization models for task usage shapes in Google's compute cloud.	Statistical Analysis	X
Di Martino et al. [44]	Cloud data	Analyzing the causes of SLA violations for SaaS platform	Statistical Analysis	X
Chen Weiwei et al.[31]	Scientific workflow applications	Transient failures and the evaluation of fault-tolerant task clustering techniques on homogeneous environments.	Statistical and Probabilistic Analysis	X

Table 2.4: Summary of the state of the art in the field of cloud computing for failure analysis and prediction

Related work	Trace	Focus	Model	Results
Samak et al.[113]	Scientific workflow applications	Predicting the failure probability of jobs	Machine Learning (Naive Bayes)	X
Bala and Chana[21]	Scientific workflow applications	Task failure prediction	Machine Learning (NB RF,LR,ANN)	NB is the highest accuracy(93%)
Rosa et al.[107]	Google trace	Failure Analysis and Prediction for failed tasks	Machine Learning (LDA, QDA, LR, SVM, ANN)	For ANN accuracy (76.8%)
Liu et al.[74]	Google trace	Predicting job termination status	Machine Learning (ELM, SVM)	Accuracy(93.07%) Precision(93.32%)
Reiss and et al.[104]	Google trace	Highlighting the heterogeneous and highly dynamic behavior of the big-data trace	Statistical Analysis	X
Jassas and Mahmoud[59]	Google trace	Study the workload features such as memory usage, CPU speed, disk space.	Statistical Analysis	X
Mao et al.[80]	IMS and CRWU bearing data	Prediction method for imbalanced fault diagnosis problem	Machine Learning (ELM)	X
Wang et al.[133]	Network management logs Equipment failure data	focus on risk-aware models in optical networks, and investigate how to predict the risk of an equipment failure.	Machine Learning (SVM)	Accuracy (95%)
El-Sayed et al.[46]	Google trace CMU OpenCloud LANL HPC Cluster	Design a job failure prediction model	Machine Learning (RF)	Precision(95%) Recall(94%)
Shetty et al.[120]	Google trace	Predicting job termination status using three different resampling techniques	Machine Learning (XGboost classifier)	Precision(92%) Recall(94.8%)
Jassas and Mahmoud[62, 60]	Google trace Mustang Trinity	Design a failure prediction model	Machine Learning (RF, DTs, NB, QDA)	For Google trace Accuracy(99%) Precision(99%)

2.7 Edge-Cloud Architecture

Devices at the edge of the network offload tasks to the cloud for processing in traditional cloud computing. Due to insufficient processing capabilities in some devices, devices with capacity-limited batteries are forced to offload their tasks in order to extend battery life. The offloading mechanism is similar in edge and mist computing. Edge nodes can collaborate to enhance system throughput by offloading tasks. Deng et al. [40] developed an approximation method for transferring load on fog devices and looked into the trade-off between transmission latency and power consumption. Yousef-pour et al. [135] developed a basic paradigm for task distribution in IoT applications. Their framework aims to minimize response time and apply job distribution decisions based on how simple or complex the jobs are to be processed.

In [139], an approach to offloading energy-sensitive tasks has been developed. The architecture enables devices to select whether to transfer their tasks to the cloud or the Fog based on their delay tolerance and power consumption. According to the findings, this technique improves cloud-only and fog-only solutions. Mtibaa et al. [89] have developed a technique to distribute the computing load across the nodes of a mobile device cloud (MDC) in a way that minimizes power consumption.

Mukherjee et al. [91] proposed a framework for performing data analysis in the IoT by utilizing connected devices at the network edge. Capacity-based partitioning, which divides data into chunks based on the device's capabilities, was designed to address this issue.

Performance has been reduced due to using these devices, but cloud demand has decreased, perhaps resolving the cloud scalability. Based on resource availability, the authors [115] provide a framework that schedules tasks across IoT gateways, fog servers, and the cloud. In [29], Chamola et al. focused on reducing latency in Mobile Edge Computing by looking for the optimal fog node to perform tasks. Compared to the standard approach, which simply uses the closest fog node, this algorithm has reduced latency. Cao et al. [26] also presented the multi-access edge computing (MAEC) concept and its key uses. They also looked at the most important research using various machine learning methods. In state-of-the-art research, the authors highlighted the need for improved intelligence in MAEC and investigated fundamental concepts of common ML-based techniques.

We aim to design and implement highly available IoT applications that integrate cloud and edge computing to minimize the failure rate and reduce the time and cost of using the Cloud. As a result, one of the approaches that can be applied is the offloading and scheduling techniques. We have also studied and proposed various failure prediction models [62, 63]. We are interested in evaluating the performance of IoT applications in the integration of edge and cloud environments.

2.8 Summary

This chapter discussed recent advances in cloud-IoT integration, cloud fault-tolerance, and failure analysis and prediction utilizing machine learning and deep learning. We conducted

a comprehensive study of the literature on failure analysis and prediction, utilizing Google cluster traces. First, we studied the big picture of fault tolerance by focusing on the reliability and availability of cloud computing to comprehend the dependability challenges related to cloud-IoT applications. We then addressed related works on task failure prediction from three relevant perspectives: statistical analysis, machine learning, and deep learning. The next chapter describes the proposed solutions for designing highly available and reliable cloud-IoT applications.

Chapter 3

Proposed Framework

This chapter presents a framework for providing high reliability and availability of cloud-IoT integration. The main objective is to design and apply proactive fault-tolerance techniques to increase the reliability and availability of cloud-IoT applications. We also aim to improve the computing performance of cloud-IoT applications and reduce the time and cost of using public cloud.

We have developed a scheduling algorithm for transferring incoming tasks based on the required computation. Moreover, we have studied and characterized different cloud workload traces focusing on failure analysis. Then, we have applied several failure prediction models based on machine learning and deep learning approaches to select the best performance among all classifiers. The proposed model can be used in large data centers in order to detect failed jobs before the cloud management system schedules them. Finally,

we have designed a highly reliable and available Edge-Cloud architecture that can serve the new model of IoT applications.

3.1 Assumptions

The scope of this research is limited to the cloud and edge computing for IoT applications that require fault tolerance and high scalability. The proposed framework aims to increase the reliability and availability of cloud-IoT applications. Additionally, it demonstrates fault-tolerance techniques for cloud-IoT architectures through the use of load balancers and availability zones. The following assumptions apply to the framework described in this thesis:

- Incoming tasks have already been classified in terms of *priority levels*, *scheduling classes*, and *required computation*.
- If incoming tasks are not classified, the load balancer (LB) server executes the task for a fixed period in order to find out the required type of computation. The specified of execution could be changed according to system requirements.
- The IoT devices presented in the proposed framework are assumed to have computation power.
- The workload traces used in this research were collected from the cloud, and high-performance computing environments, such as Google cluster traces. Monitoring

data indicate cloud characteristics, such as the required resources (CPU, memory, disk storage) to perform a task. As a result, the proposed failure prediction model has been designed and implemented for cloud deployment. However, the task failure prediction model could be adapted to various forms of computing, such as edge, fog or local servers, because the observed data is similar.

- The failure prediction models were designed and implemented to predict task failures, but the proposed model might also be used to predict job failures. Based on our assumption, the failure prediction model is trained and tested offline, as it requires a long training time and a high computation power.

3.2 Framework Architecture

As a preliminary step, a cloud-IoT framework has been developed by utilizing cloud computing to enhance local IoT resources, which are significantly limited in processing, memory, and storage. The proposed framework considers the local load balancer because it is essential for allocating incoming workloads to available resources. As shown in Figure 3.1, IoT sensors are connected to the local resources, which act as a gateway, to collect data from different sensors. The scheduling algorithm transfers incoming tasks to appropriate resources based on the required computation and classification of receiving tasks. The cloud computing environment executes incoming tasks if these tasks require high computation. Otherwise, local resources can handle these tasks with low latency. Thus, instead

of sending all incoming tasks to the cloud, the local scheduling algorithm assigns received tasks to appropriate resources. The external cloud load balancer plays an important role in applying the failure prediction model before scheduling incoming tasks to the available resources in the cloud environment. Moreover, the framework provides different availability levels based on business and application requirements. As a result, the proposed framework reduces the cost of using public clouds and increases cloud-IoT availability and performance.

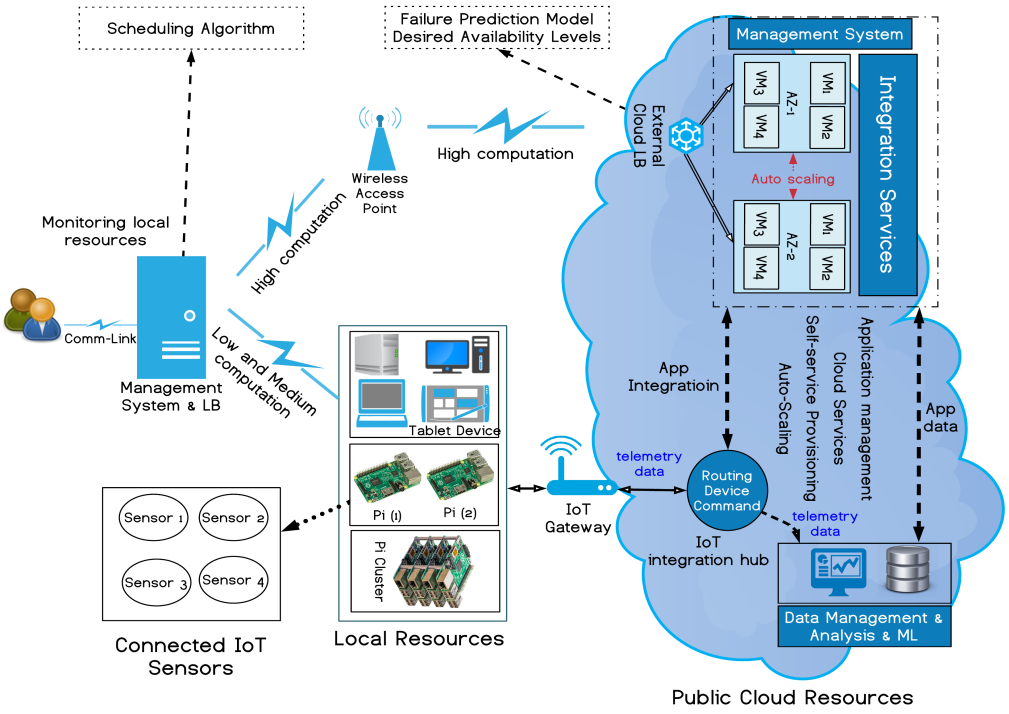


Figure 3.1: The proposed framework

Modern cloud-based applications, including smart homes and cities, require high levels

of reliability and availability. All cloud services, including hardware and software, experience failures because of their large scale and heterogeneous nature. We have stated the problem as thus: using the cloud workload attributes to answer the essential question: Can we build a failure prediction model that can decrease the number of failed tasks and save cloud resources?

Some features of the framework, components, concepts, and techniques that are designed to build this framework are explained in the following sections:

3.2.1 Local Resources

Local resources include local servers, IoT devices, and sensors. IoT sensors are connected to local resources such as Raspberry Pi, which act as a getaway. Recently, Raspberry Pi devices have become more powerful; they can be used to transfer collected sensor data to the cloud in order to be processed and analyzed. Therefore, instead of processing this data in the cloud, part of the most current data has been stored locally for low-latency processing. For example, if users want to monitor their house temperature for the last three days, the data will be saved in local resources, resulting in high-performance outcomes.

Local Servers

The local servers are responsible for executing the medium computational tasks. Instead of fully executing all operations in the cloud, we can reduce the cost of using the cloud and

avoid the latency challenge by performing some operations on local servers.

IoT Devices and Sensors

In general, IoT devices communicate with IoT gateways, local servers, or edge devices, from which sensor data can be processed locally and sent to the cloud for analysis. The data processing capabilities are incorporated in specific devices, which reduces the quantity of data that must be transmitted to the cloud or data centre. Machine learning is often used to process data from IoT devices, and it is growing in popularity as more IoT devices generate data.

IoT Gateway and IoT Integration Hub

An IoT gateway connects IoT devices and sensors to the cloud IoT platform and provides a bridge between different communication technologies, which often have different connectivity, protocols, or interfaces. An IoT gateway may connect to hundreds of sensors and communicate with them using a variety of protocols such as Message Queuing Telemetry Transport (MQTT), ZigBee, and Bluetooth. Another important capability of IoT gateway is to filter out unnecessary data to reduce the amount of data that is delivered to the cloud or edge computing [99].

The cloud-based IoT Integration Hub manages communication between IoT applications and connected devices. Millions of devices and backend systems can be consistently

and safely connected. In addition, the sensor data is called telemetry; it generally enables robust data collection and transfer to centralized systems for effective use. Telemetry is data acquired by sensors and is read-only [99].

3.2.2 Management System and Local Load Balancer

The management system and local load balancer are significant components because the management system plays an important role in monitoring the local resources in terms of availability. The load balancer is responsible for transferring incoming tasks to the healthy nodes based on the required computation [70].

Local Load Balancer

The task scheduling will be applied based on the ***classification of the tasks*** and the ***computation power required***. Local servers are suitable for high-priority tasks that require low or medium computing power. In a home automation system, for example, temperature monitoring and control should be performed on local resources, not in the public cloud. High-priority operations that require real-time execution are performed near IoT devices to meet the expected response time.

3.2.3 Cloud Resources and External Cloud Load Balancer

We integrate local resources with the cloud to increase the rate of availability to ensure that the framework provides high availability. Some machine learning tasks require high computation, so the cloud is responsible for handling this type of task. In order to increase the availability of the proposed framework, we have also developed a new approach for the external cloud load balancer. We have designed our new cloud load balancer to predict the failed tasks before they are transferred to the available VMs. This approach will help to use cloud resources efficiently.

3.3 Extending IoT Local Resources

Modern IoT devices generate a large volume of data that requires a highly scalable and available framework that enables computing, storage, and data analysis. Many cloud providers offer unlimited storage as well as a flexible processing infrastructure, allowing for executing a large number of tasks with high performance. Even though digital information technology is widespread, IoT devices have limited storage, computation, and memory capacity. Sensors, actuators, or smartphones generate vast amounts of data within a short time. These data need to be stored, monitored, managed and analyzed in order to extract useful information. Thus, integrating local IoT resources and cloud computing can be one of the best solutions to increase the capability of IoT devices. In an application level, the main challenge that needs to be addressed is handling tasks between local resources

and the cloud. Our goal is to distribute resources efficiently to design an available and scalable framework for IoT applications. In order to accomplish this goal, we will focus our attention on three issues. To begin, developing a computing strategy that minimizes the response time of connected devices. The second step is to devise a method for increasing throughput. Finally, minimizing the amount of time spent in the cloud reduces the expense of using public cloud services.

This thesis presents a framework for extending the local resources of IoT devices, which are limited in terms of computing, memory, and storage, using cloud computing. This framework provides scalability and high availability for cloud-IoT applications. Moreover, a scheduling algorithm is implemented to minimize the execution time and computation cost of using the cloud, according to a logic that directly depends on tasks' classification and computation requirements.

The core of the framework is to design and implement an architecture for integrating IoT devices and local resources with the cloud to increase the computing performance of IoT applications and reduce the cost of using the public cloud. The aim is to extend local IoT resources with public cloud services to solve the limitations of the local resources, including local servers and IoT devices such as Raspberry Pi, Arduino, and microcontrollers. Figure 3.2 shows the framework architecture for extending local resources to the cloud. Sensors are connected to IoT devices, which act as a gateway to collect data from different types of sensors. The scheduling algorithm transfers incoming tasks from local resources to appropriate resources based on the priority levels and computation requirements. The

cloud computing environment executes incoming tasks if these tasks require high computation. Otherwise, local resources can handle these requests with low latency. Instead of sending all incoming requests to the cloud, the scheduling algorithm assigns received tasks to appropriate resources. As a result, the proposed framework reduces the cost of using the public cloud as well as the proposed framework increases the IoT application performance.

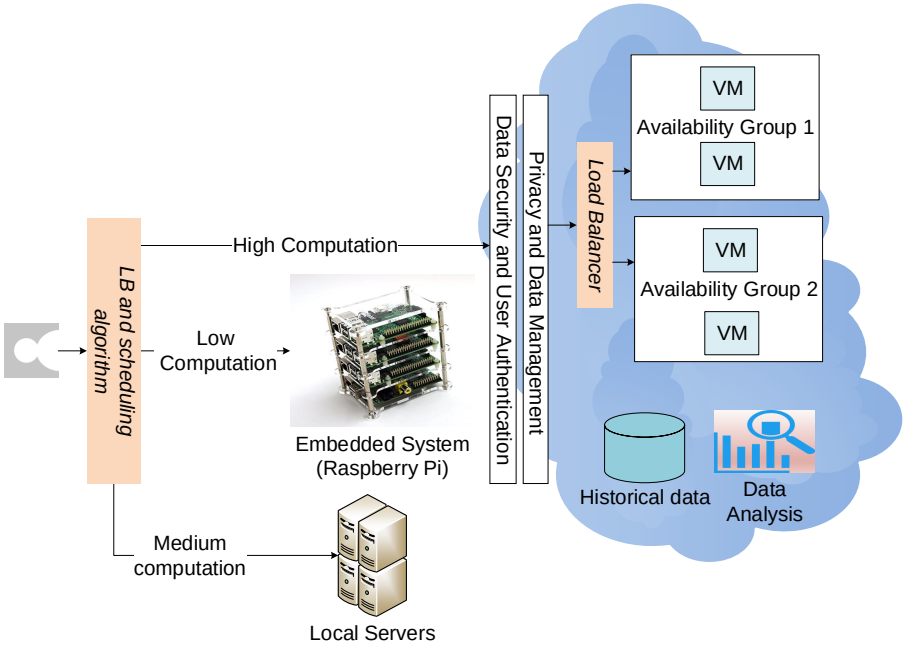


Figure 3.2: Framework architecture

3.3.1 Load Balancer and Task Offloading

Load balancing is responsible for ensuring that only healthy VMs receive requests by detecting unhealthy VMs and changing the new request path towards the remaining healthy

VMs. The following are the primary advantages of load balancing [100]:

- It serves in traffic control and tracking.
- It increases availability of resources.
- It optimizes network load distribution depending on node capabilities.
- It enhances resource utilization.

Cloud providers employ a variety of load balancer scheduling algorithms, including round-robin, weighted round-robin, least-connection, and dynamic feedback. On the other hand, modern cloud-IoT systems require a new offloading approach to balance numerous tasks between local IoT resources and the cloud. This technique should take task computation and priority levels into account. Based on the failure analysis explained in detail in Chapter 4, we have noticed that failed tasks and jobs consumed many resources because the management system has resubmitted the failed tasks hundreds of times to be completed.

The task offloading will be applied based on the ***classification of the tasks*** and the ***computation power required***. The logic for switching or scheduling between local resources and cloud resources are clearly presented in Table 3.1. Figure 3.3 shows the flowchart of the proposed scheduling algorithm.

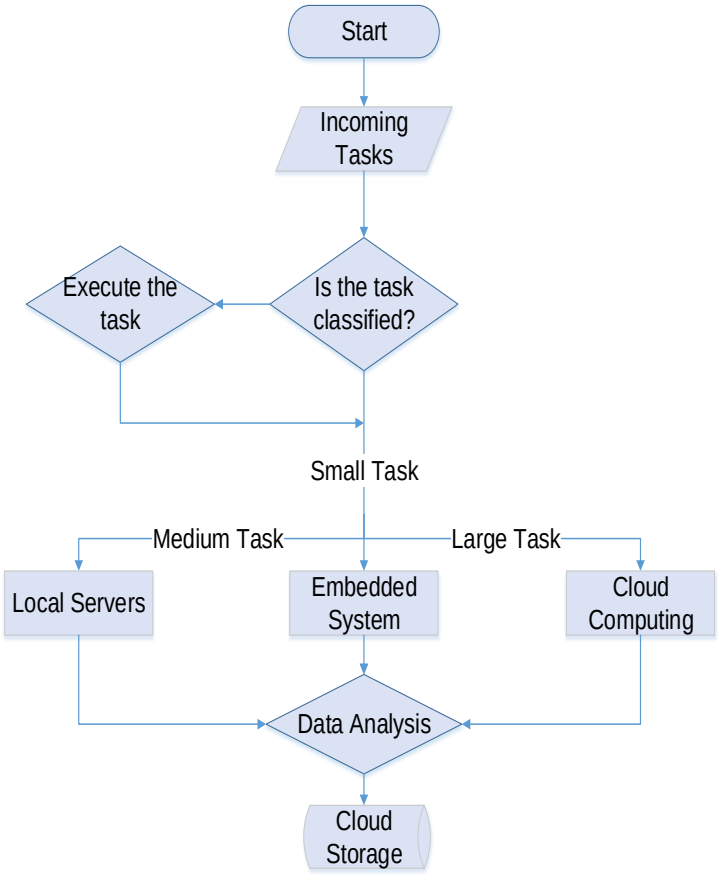


Figure 3.3: Flowchart of the proposed scheduling algorithm

3.3.2 Classification Task Analysis

High priority tasks are required to be executed in a high-security environment with low latency, so local servers resources are the best choice in order to execute these types task.

Algorithm 1 presents the proposed model to be applied to local scheduler. The phrase "embedded systems" is occasionally used to describe IoT devices with the built-in process-

Table 3.1: Logic to switch between available resources according to priority and computation power required

	Low priority	**High priority**
Low computation	IoT devices will be utilized, or if local IoT resources are busy with other tasks, the new task can be executed in the cloud.	IoT devices will be utilized.
Medium computation	Local servers will be utilized, or if the local servers are busy with other tasks, the new task can be executed in the cloud.	Local servers will be utilized.
High computation	Cloud computing will be utilized.	Cloud computing with high-security mechanisms.

ing power. The algorithm input has two parameters: number of incoming requests or tasks, t_1, t_2....t_n, and available resources, including Embedded System (ES), Local Servers (LS), and Cloud Servers (CS). The algorithm output is the assignment of tasks t_1, t_2....t_n to the appropriate computation resources (ES,LS, or CS). At the beginning of the proposed algorithm, *TaskResAllocationList* and *AvailableResList* are initialized with a *Null* value (line 1 and 2). The *TaskResAllocationList* is a list for storing the appropriate allocation for each task based on ***classification of the tasks*** and the ***computation power required***.

The scheduling algorithm transfers the classified task or incoming requests to the appropriate resources (ES,LS, or CS) depending on the healthy resources. Initially, the algorithm checks if an incoming task is classified or not, so if it is classified, it will be normally scheduled and executed in the appropriate compute resources. Otherwise, If the task is unclassified, the algorithm attempts to execute the task for a period of time. We assume that the maximum execution time is 600 ms, but this value can be changed based on the system requirements.

Then, the task will be terminated to the appropriate resources whether the task has been completely executed or not (line 5 - 11). If the task execution time is less than 300

Algorithm 1: Proposed algorithm to be applied in local scheduler

Input : T: Incoming tasks (t_1, t_2....t_n)
AvailableResList: Available Resources
Output: Task resources allocation *(ES,LS,or CS)*

```
1  TaskResAllocationList = Null;
2  AvailableResList = Null
3  T ← (t1, t2....tn)
4  foreach t in T do
5  |   find all available resources
6  |   AvailableResList ←
7  |   if (taskType == classified) then
8  |   |   Assign task to an appropriate resources
9  |   |   TaskResAllocationList ←
10 |   else
11 |   |   Execute the task for a specific time, say, 600ms
12 |   |   if (taskExecutionTime < 300ms) then
13 |   |   |   Assign task to ES or IoT devices
14 |   |   |   TaskResAllocationList ←
15 |   |   else if (300ms ⩽ taskExecutionTime ⩽ 600ms) then
16 |   |   |   Assign task to LS
17 |   |   |   TaskResAllocationList ←
18 |   |   else
19 |   |   |   Assign task to CS
20 |   |   |   TaskResAllocationList ←
21 |   |   end
22 |   end
23 end
```

ms, it can be considered a low computational task. In this case, the task will be transferred and executed on the IoT devices. Otherwise, if the task execution time is between 300-600 ms, the task will be transferred and executed on the local servers. Finally, if the task execution time exceeds 600 ms, the task will be assigned and performed in the cloud (lines 12 - 20).

Task scheduling plays a significant role in combining local resources and cloud computing. In the proposed solution, we apply the scheduling technique in order to map users' tasks to appropriate resources based on the task criticality and computation types, as shown in Table 3.1. We classify the task into two types: high priority and low priority. Also, computational tasks have been classified into three types: high, medium, and low. Thus, the scheduling algorithm will be configured based on some policies in order to minimize the execution time and execution cost of using cloud computing resources. Small tasks should not be assigned to high-computing resources, which wastes time and cost of using the cloud. In this case, small tasks can be handled by IoT devices in a low response time and high performance without relying on cloud computing. Pseudocode for a scheduling algorithm to offload tasks is shown in Algorithm1.

3.3.3 Scalability and Availability in the Cloud

The proposed framework has been designed to provide scalability and high availability using cloud services. After applying the scalability techniques, the proposed framework

can efficiently serve a large number of tasks. Horizontal scaling will be selected as this type of scaling allows the system to scale in or scale out based on the system requirements.

Auto-scaling service is one of the best features of cloud computing, and it could be done according to some available metrics, such as CPU average. If the average CPU usage for all operating VMs is more significant than 50%, cloud users can automatically set metrics to create three additional virtual machines (VMs).

When we launch a single VM, we are launching a VM running on a physical server at one of the data centers. As a result, one of the following events could take down the VM:

- The VM itself could fail, or the hard drive volume could become corrupted;
- The physical server on which the VM resides could fail;
- The data center, which houses physical machines, could fail.

The following section introduces a failure prediction model that can be applied to the load balancer algorithm to increase the reliability and availability of cloud computing resources and local IoT resources.

3.4 Job and Task Failure Analysis

Modern applications, such as smart cities, home automation, and eHealth, demand a new approach to improve cloud application dependability and availability. Due to the enormous

scope and diversity of the cloud environment, most cloud services, including hardware and software, have encountered failures. In this thesis, we first analyze and characterize the behaviour of failed and completed jobs using publicly accessible traces. Then, in the next section, we have designed and developed a failure prediction model to determine failed jobs before they occur. The primary objective of this work is to enhance the understanding of job failure in cloud computing environments. The results show a clear correlation between failed jobs and requested resources, including memory, CPU, and disk space. Based on our results, we find that many techniques can be applied to increase the reliability and availability of cloud applications, such as developing scheduling algorithms, predicting job failure, limiting task resubmission or changing the priority policies.

Failure analysis is different from failure prediction. Failure analysis techniques play an important role in observing some misbehaviour in a running cloud system to identify the cause of hardware and software failure. On the other hand, the main goal of failure prediction is to detect faults before they occur early. A failure prediction model is developed based on measuring the most important metrics and characteristics of cloud applications. The primary aim of failure analysis and prediction is to study the frequent changes and the behaviour of cloud applications to increase cloud applications' performance and reduce the number of failed tasks. Figure 3.4 presents the relationship between failure analysis and failure prediction [77]. We are interested in studying the effect of job/task scheduling and their priority policies on failure.

As researchers, we value how well prediction models work when tested against real-

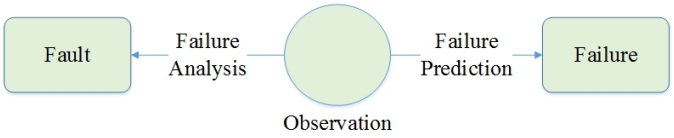


Figure 3.4: Distinction between failure analysis and failure prediction

world workloads. However, anyone who has tried to find data to accomplish this kind of analysis knows that there are few available public workload traces. Legal and cultural barriers to disclosing data are often responsible for this data availability. Even when data sets are made publicly available, they are presented in noncanonical forms, remove elements valuable to researchers, and are published separately on websites that eventually become inaccessible to researchers [14].

3.5 Failure Prediction using Machine Learning

This section discusses the main components of the proposed framework for task failure prediction. The failure prediction model has been designed and implemented to predict the termination status of submitted tasks before the execution time. The proposed model aims to enhance resource consumption and cloud application efficiency. We evaluate the proposed model using publicly available cloud traces such as Google cluster traces [103]. In addition, the traces are also subjected to various machine learning and deep learning models to find the most accurate one. Several solutions, such as predicting job failure, developing scheduling algorithms, changing priority policies, or limiting re-submission of

tasks, can improve the reliability and availability of cloud services.

Four important phases of the failure prediction model design are as follows: 1) The cloud management system monitors the application metrics and the requested resources such as memory, CPU, and disk space. Then, the monitored data will be stored in the cloud storage in order to be used for the data prepossessing and failure analysis and prediction phases. 2) Applying data prepossessing and filtration techniques is essential in data science. The quality of the failure prediction model is based on this step. 3) The machine learning algorithm will be applied to the cloud workload traces in order to predict failed jobs or tasks. 4) Finally, the cloud management system will make the appropriate decision based on the prediction results. If the job is predicted as a successful job, the job continues to be submitted and typically scheduled to the available nodes. Otherwise, failure mitigation techniques will be applied if the job is predicted as a failed job. Figure 3.5 presents the phases of the failure prediction model.

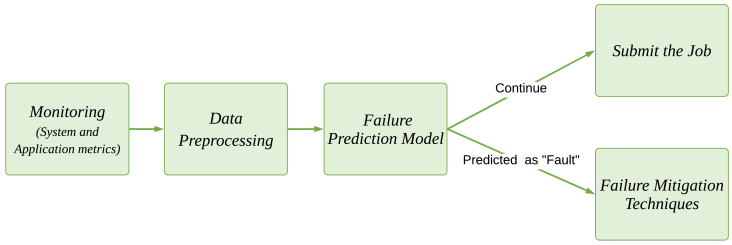


Figure 3.5: Failure predication model

3.5.1 Failure Prediction Model

The main aim of the proposed framework is to predict the job/task failure in the cloud with high accuracy using available machine learning classification algorithms, and this technique assists in reducing waste cloud resources and improving the utilization of cloud infrastructures.

Algorithm 2 describes the failure prediction model. The algorithm inputs cloud trace workload D, which includes a number of tasks. The algorithm also applies and tests different feature selection techniques and classifier models to the selected cloud trace. The algorithm returns the termination status *failed/finished*. Set D represents the dataset that is extracted from the input cloud trace. The data preprocessing has applied to the selected dataset. The data preprocessing steps include the cleaning and filtering out of all tasks that were submitted hundreds of times. After that, the management system kills them after they have consumed many resources. The selected cloud trace is used for training and testing the prediction models. The M indicates the selected model used in the prediction for classifying *failed* and *finished* tasks. At the beginning of the proposed algorithm, *performanceList* and *topRankFeaturesList* are initialized with a *Null* value (line 1). The *topRankFeaturesList* is a list for storing the best features for each feature selection technique to be used as input for the selected model M. The algorithm then tests different feature selection techniques *(SelectKBest, Feature Importance, RFE)* with different prediction models *(RF, DS, NB, QDA, KNN, XGBoost, and Gradient boosting)*, which present in

Algorithm 2: Failure Prediction Algorithm

Input : D: Data Workload Trace
M: Model
F: Feature Selection Technique

Output: Termination status *Failed/Finished*

$performanceList = Null$; $topRankFeaturesList = Null$;
$M \leftarrow (RF, DS, NB, QDA, KNN, XGBoost, GBoosting)$
$F \leftarrow (RFE, FeatureImportance, SelectBest)$
Data preprocessing
for $d \in D$ **do**
    Clean d;
    Delete mising data;
    Filter out those tasks that resubmitted hundreds of times;
end
forall *elements of* F **do**
    Apply feature selection technique to the D
    Select the top ranked features
    $topRankFeaturesList \leftarrow$
    forall *elements of* M **do**
        Apply classifer to the D
        Evalute performance (accuracy, precision, recall, F1-score);
        Add *(Acc,Prec.,Rec.,F1-score)* to *performanceList*;
    end
end
return *[performanceList]*
Select the best model performance from performanceList
foreach *task in the job* **do**
    predict the task termination status;
    if *(terminationStatus* > 0*)* **then**
        terminationStatus == "Failed"
        Apply failure mitigation techniques;
    else
        terminationStatus == "Finished"
    end
end

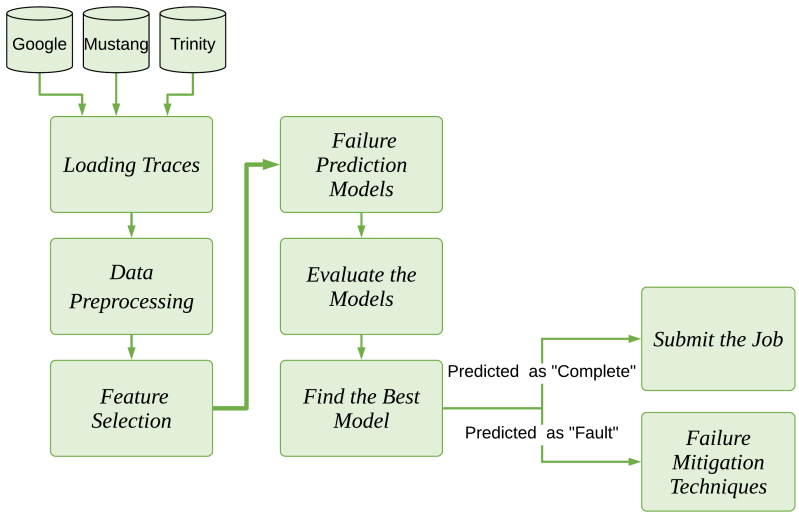


Figure 3.6: Evaluation process

lines 2 and 3. The algorithm also considers data preprocessing as one of the most important steps before applying the proposed model (lines 4-9). The algorithm then applied different selection techniques to different prediction models (lines 10-15). Then, the algorithm evaluates the performance and stores the results to the *performanceList* (line 16 and 17); after that, the algorithm selects the proposed model based on the best performance results (line 21). The algorithm predicts the termination status (line 23) of using the failure prediction model. If the termination status of greater than 0 is predicted as *"failed"*, and then apply one of the failure mitigation techniques (lines 24-26). Otherwise, if the termination status of t is predicted as *"finished"* (line 28). Additionally, the proposed evaluation process for the failure prediction model is depicted in Figure 3.6.

We can summarise the proposed model's process as follows:

1) The workload data was loaded from three distinct traces, with the primary goal of ensuring that the proposed model can be applied to traces of varying lengths.

2) Analysis, pre-processing, and filtering techniques have been used to ensure that the data is ready for classification and modeling.

3) Three feature selection techniques have been applied to the traces to improve the proposed model's accuracy and performance. After that, we can rank the most important features based on the results.

4) To predict failed and finished jobs, seven machine learning classification approaches were used to the traces.

5) Finally, the cloud management system determines the appropriate failure prediction model based on the best prediction findings. If the job is predicted to be *finish*, it will be submitted and typically scheduled to available nodes. Otherwise, failure mitigation strategies will be applied if the incoming job is predicted to be *fail*; however, it is important to highlight that this phase will be handled in future work.

The primary goal of the failure prediction model is to early predict *failed/finished* tasks in the cloud-IoT applications with high rate of accuracy using ML classification algorithms. The proposed model helps to reduce computational time and resource consumption, and it increases the efficiency and performance of cloud-IoT infrastructure.

3.5.2 Data Preprocessing and Filtering

The model's input is a set of features that can describe the *job/task* attributes and the behaviour of a cloud system. We have filtered out all event types of *jobs/tasks* that have not occurred within the Google trace window (from May 1 to May 29), and these jobs and tasks are presented as a time of 0 in the trace window. We have also converted all jobs and tasks timestamp microseconds to daytime. In Google cluster trace, we investigated why the number of failed tasks is very high compared to the number of failed jobs, so we found that some tasks are resubmitted thousands of times to be successfully finished. However, these tasks were killed after they had consumed considerable resources. As a result, we have removed these tasks, including these types of tasks, because they are considered outlier cases. During this period, we expect a data centre outage to occur.

3.5.3 Feature Selection Algorithms

First, we manually select the most relevant characteristics to save training time for the classification algorithm and avoid over-fitting. Then, in order to increase the accuracy of the proposed model, we used feature selection algorithms including Feature Importance, SelectKBest and Recursive Feature Elimination (RFE).

3.5.4 Prediction Techniques

The Decision Trees (DTs) classifier is one of the supervised learning algorithms that we employ in our research. In order to learn from data characteristics, the DTs algorithm incorporates many decision-making principles, which are discussed more below. The "entropy" of a split is the criterion we employ to evaluate its overall quality. The DTs algorithm uses "entropy" to determine the homogeneity (number of equivalent values). The cost complexity of a tree may be calculated using the number of leaves in the tree and the error rate of the tree. It may be necessary to employ other classification algorithms to discover the most accurate, efficient, and dynamic model solution within a reasonable situation. We utilized seven Machine Learning (ML) classification algorithms: RF, DTs, KNN, QDA, Gradient Boosting, XGBoost and NB.

3.6 Failure Prediction using Deep Learning

Many cloud service providers face significant challenges in preventing hardware and software failure from occurring. Due to the large-scale and heterogeneous nature, cloud services continue to experience failures in their components. Although a number of prior studies have concentrated on characterizing unsuccessful jobs, others have focused on improving failure prediction models by enhancing their accuracy. This thesis presents the development and implementation of a failure prediction model using a deep learning approach. The proposed model can identify and detect failed tasks early on before they occur. The

key feature of the failure prediction model is to improve the performance of cloud applications by reducing the number of failed jobs. In order to investigate the behaviour of failure and apply the prediction of failure to the large-scale environment, we used three different traces, namely Google Cluster Trace, Mustang and Trinity. Moreover, we have evaluated the proposed model performance using different evaluation metrics to ensure that the proposed model provides the highest accuracy of predicted values. The proposed model is designed and implemented to achieve high accuracy for failure prediction, regardless of whether the model uses a large or small trace size.

In recent times, deep learning algorithms have been used in many areas, such as cloud computing, computer vision, and the Internet of Things (IoT). Deep learning algorithms require a high volume of data in order to achieve high accuracy of prediction. Figure 3.7 presents our model architecture for failure prediction. There are three essential stages in order to complete the failure prediction process. First, the model is designed to check the number of trace observations to assist in ensuring that the utilized trace has enough observations for prediction, and it can be utilized for training a deep learning model. After that, data preprocessing will be applied to the utilized traces to ensure that the data is ready for analysis and prediction. Second, if the trace has enough observations for deep learning, the Artificial Neural Networks (ANNs) algorithm will be applied to the trace. Otherwise, traditional machine learning algorithms will be applied. This technique will assist in saving cloud resources that can be consumed for deep learning training and testing time, which require high-performance computing. Finally, the failure prediction

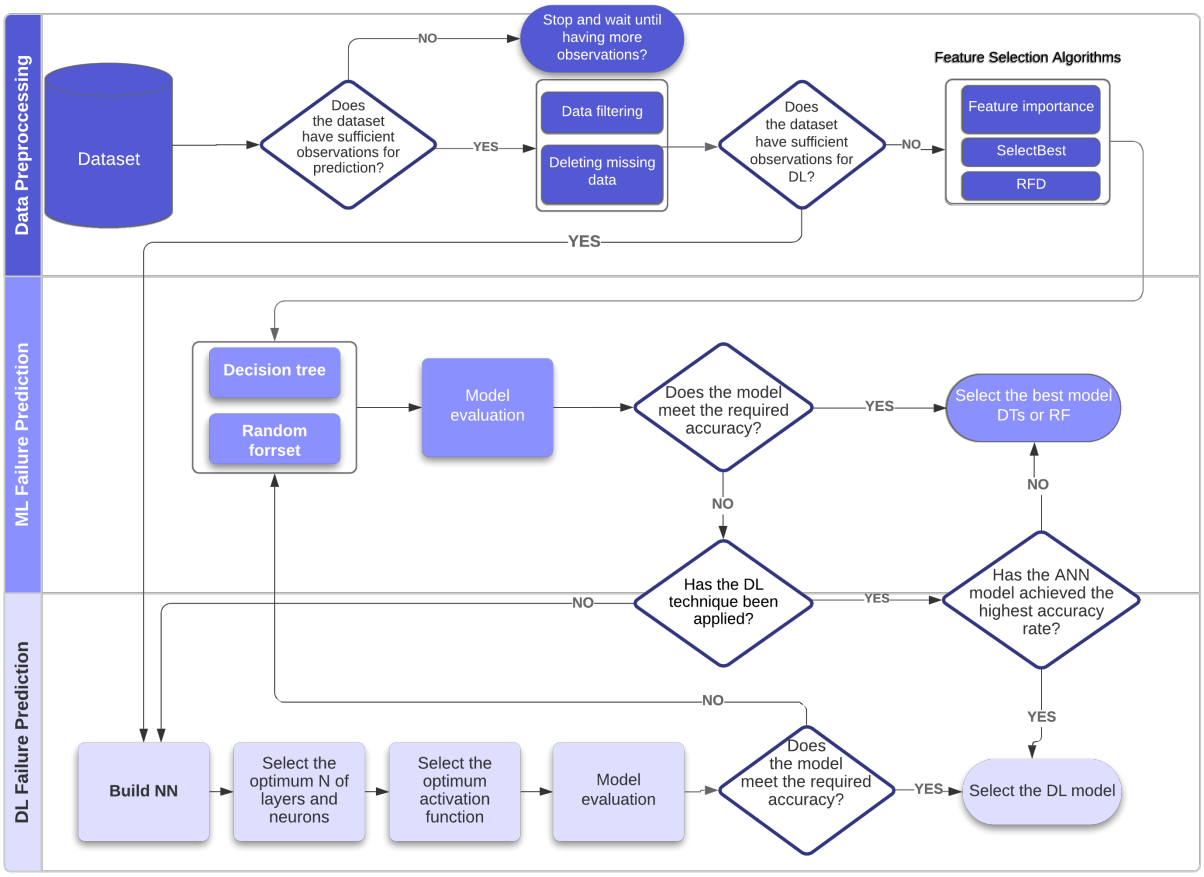


Figure 3.7: Failure prediction model based on the deep learning model

model will be evaluated to ensure that the model has achieved the expected accuracy. If the ANN-based model is unable to achieve high accuracy, the traditional ML algorithms will be applied. Then, the proposed model will be selected based on the higher accuracy.

The main objective of the proposed model is to apply a proactive fault-tolerance approach that can predict the status of the failure in a large-scale environment such as cloud computing. The proposed model has been designed and implemented based on machine

learning and deep learning classification algorithms. This approach can increase cloud application efficiency by reducing wasted resources and enhancing the utilization of cloud resources and other existing local computational infrastructure.

3.6.1 Data Pre-processing

The input of our failure prediction model is a feature set that represents the most important attributes of job and task events. It is important to note that some observations were filtered out from the utilized Google traces because they are considered outliers and might affect the model accuracy. Section 4.2 presents the data preprocessing steps for Google cluster trace in detail.

3.6.2 Feature Selection

At the beginning of our experiment for machine learning algorithms such as RF and DTs, we manually select the most important features because selecting effective and relevant features is crucial for classification. This helps avoid the impact of extremely high data dimensionality, decrease preprocessing training time, and reduce the risk of over-fitting. Then, we have increased the accuracy of our proposed model by applying different selection algorithms such as SelectKBest, Feature Importance, and Recursive Feature Elimination (RFE). In contrast, deep learning algorithms do not require the use of feature selection methods. Deep learning algorithms learn the best features from data, without any human

guidance.

3.6.3 Traditional Machine Learning and ANNs Algorithms

Several supervised learning algorithms have been applied to evaluate and select the proposed model based on the performance metrics, including precision, recall, and F1-score. The following subsection explains most algorithms used in this thesis.

Decision Trees

We have applied Decision Trees (DTs) to the three different traces with three different sizes. Google trace has a large number of observations, while Mustang and Trinity have medium and small sizes, respectively. DTs are one of the most popular supervised learning models in data science. The main objective of applying a DTs classifier to cloud traces is to build a model that accurately predicts the value of a target variable from multiple input variables. The splitting criterion that we have utilized in order to evaluate the quality of a split is "entropy". The decision trees classifier has used the entropy in order to calculate the homogeneity of the sample. The cost complexity can be evaluated using error rating and the number of leaves in the decision tree.

Random Forest

We also applied the Random Forest (RF) classifier to the utilized traces. RF is implemented based on an ensemble learning method that works by constructing a series of decision trees at training time. Accordingly, the Random Forest algorithm builds several decision trees by randomly sampling a certain subset. In order to achieve high efficiency, RF reduces the correlation between trees by randomly selecting the most important features that can be used to increase the prediction accuracy. RF classifier provides more advantages than other classifiers such as Naïve Bayes, SVM, KNN, and Logistic Regression. Some of these advantages include: RF can overcome and avoid the risk of over-fitting. Second, RF has the substantial advantage of being less sensitive to outliers than other classification algorithms. Third, the RF classifier can estimate the importance of each feature that can be used in the model. Finally, RF provides high accuracy of prediction on many types of applications.

Artificial Neural Networks

The majority of deep learning architecture has been built based on the framework of ANNs. ANNs architecture is constructed of many interconnected nodes/neurons. These nodes are constructed in layers, as illustrated in Figure 3.8. Neurons that are not incorporated in the input or output layers are called hidden units. Each hidden unit stores a set of weights W, which are updated during the model's training.

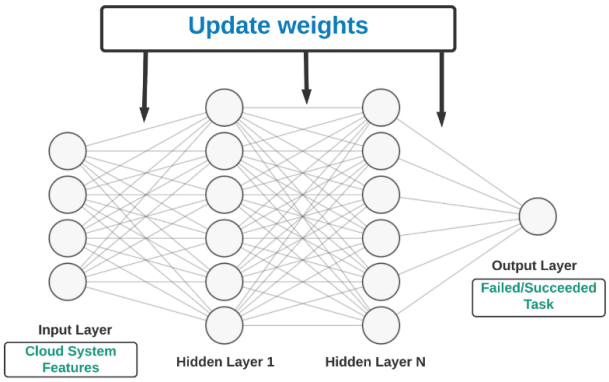


Figure 3.8: A sample neural network architecture

3.7 IoT Application using Edge-Cloud Architecture

The most important features of the IoT application architecture are connectivity, detection, scalability, intelligence and integration. The IoT architecture should be developed and implemented to be scaled up or down, depending on the business and application requirements. Cloud computing has faced various challenges due to its rapid expansion. As the number of IoT devices and their data grows, cloud computing is unable to respond and maintain its quality-of-service criteria, such as low latency. Edge computing models are urgently needed to develop the IoT applications. In this thesis, we also investigate the effects of combining IoT, cloud, and edge computing for failure analysis and prediction. Furthermore, based on the Edge-Cloud architecture, we offer a highly reliable and available IoT application that can support the new paradigm of IoT applications. The proposed model can reduce the number of failed tasks for cloud-IoT applications. We have also examined how many tasks fail when different architectures are used. The evaluation results show that

failed tasks and CPU usage have decreased in the use of the "Edge-Cloud" architecture. In addition, using "Edge-Cloud" architecture can also control network traffic compared to other architectures.

Edge computing, such as intelligent terminals, is close to the data source. The data is stored and processed quickly, securely in real-time at the edge of the network. Edge computing can also help reduce the challenge of high energy consumption compared to cloud computing, reduce costs, and reduce network bandwidth congestion. Edge computing is used in a variety of industries, including manufacturing, energy, smart homes, and transportation [20].

Edge computing models are urgently needed to develop the Internet of Things (IoT). We study the impact of of IoT and edge computing integration for failure analysis and prediction. In addition, we present an architecture for a highly reliable and available IoT application based on the development of the Edge-Cloud architecture, which can support the new model of IoT applications.

Figure 3.9 shows the model architecture for an Edge-Cloud architecture. After analyzing the workload of various IoT applications, the management system will classify these applications according to several features, including network and security requirements, backup services, bandwidth, throughput, response time and latency. The required level of availability and computation is estimated based on all of the relevant features. After calculating the required availability and computation for each application, the management system transfers incoming tasks to the cloud or edge computing according to the appli-

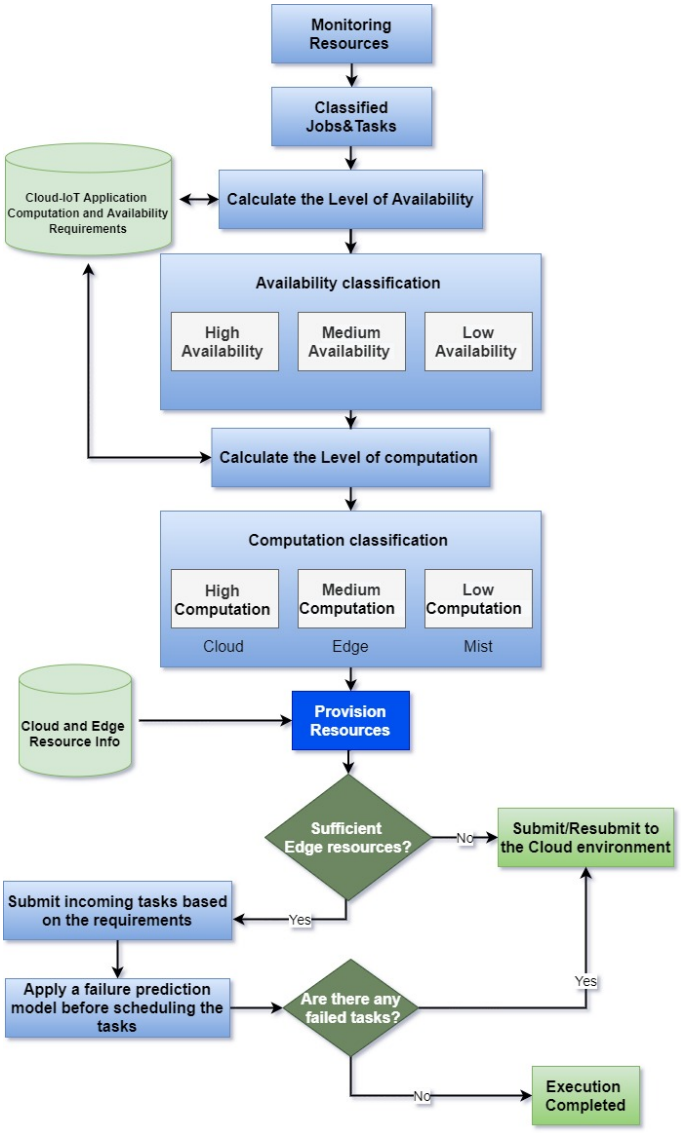


Figure 3.9: Proposed framework

Table 3.2: Different types of IoT applications and their requirements

IoT applications	Requirements	Execution location
Elderly Care Monitoring	High availability, low computation, low latency, high security	Edge computing
Smart home automation	Low computation, low latency, high security	Edge computing
Predictive machine maintenance	High computation	Cloud computing
Storing and backup services	High reliability and availability	Cloud computing
Software failure analysis and testing	High computation	Cloud computing

cation requirements. For example, Table 3.2 presents various IoT applications and their requirements. The management system then checks whether the edge computing resources are sufficient or not. If edge resources are sufficient, incoming tasks are scheduled in the edge environment. Otherwise, the computation is performed in the cloud, even if we do not achieve the desired response time. Finally, the failure prediction model is used before the management system schedules the submitted tasks in order to predict the failed tasks early. Applying the failure prediction model to all unclassified tasks helps decrease the number of failed tasks by scheduling them to more available resources with more computation power.

The proposed scheduling algorithm [61] is designed and implemented to check the computation required for each application. As a result, if the incoming task requires a high computation, it is transmitted to the cloud. Otherwise, if the incoming task belongs to an application that requires low computation, it is transmitted to the edge. For example, all tasks belonging to an intensive computation, including machine and deep learning tasks, are performed in a cloud computing environment. The eHealth and real-time applications

do not require high computation, so their tasks are executed in edge computing.

3.8 Resource Allocation and Cloud Load Balancing

In order to decrease the cost of using the cloud, we have classified the availability levels into three types: high, medium, and low, as shown in Figure 3.10. The main objective of offering different availability levels is to provide the cloud consumers multiple options for availability in order to choose the desired level of fault tolerance based on their application requirements.

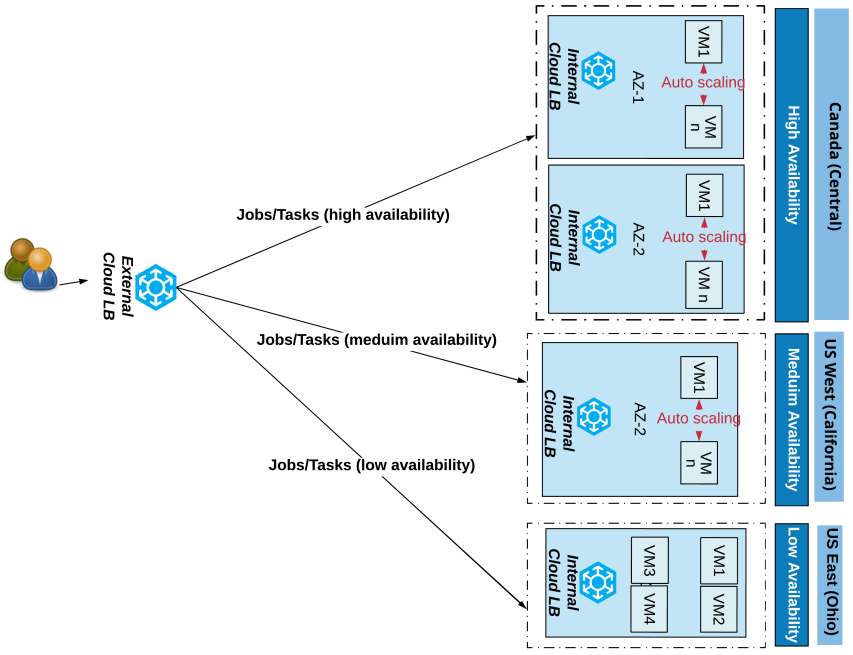


Figure 3.10: Resource allocation architecture

3.9 Summary

This chapter has presented a fault-tolerance framework to increase the reliability and availability of cloud-IoT applications. This framework considers local IoT resources and cloud resources, so we developed a cloud-based solution to expand local IoT resources. We have developed a scheduling algorithm for transferring incoming tasks based on the required computation. We have developed machine learning and deep learning models for task failure. The proposed model can be applied to large data centers in order to detect failed jobs before the cloud management system schedules them. The following chapter presents the experimental evaluation and results of our proposed framework.

Chapter 4

Experimental Evaluation and Results

This chapter describes the experimental design and analysis of the evaluation results for the previously proposed framework in Chapter 3. It includes a detailed implementation and evaluation for integrating local IoT and cloud resources to improve cloud-IoT application scalability and availability. Following that, a presentation of failure analysis to characterize the behaviour of failed and finished tasks using publicly accessible traces. This chapter then presents a performance evaluation of failure prediction models, followed by a simulation analysis of an Edge-Cloud architecture for IoT applications and evaluation results.

Integrating local IoT resources and the cloud increases the redundant components in multiple availability zones in local resources, edge, and cloud computing levels. The data center receives a user's application request via a job, which is consisting of several tasks after a scheduling module allocates computing and storage resources. Fault tolerance

strategies based on redundancy have been invented to minimize the occurrence of failures and maintain acceptable failure rates. However, rather than adding redundancy after fault happens, it is preferable to predict failures and take proactive responses. In general, failure prediction can be accomplished by examining the cloud platform's monitoring log and analyzing the relationship between task termination status and system and dynamic parameters during execution. At an earlier stage of the task's lifecycle, forecasting the task's termination status and applying appropriate scheduling actions will save more resources and effectively reduce the task's running time.

4.1 Extending IoT Local Resources

This section describes the design and implementation architecture for integrating local resources with cloud computing services. Then, a comprehensive implementation of the local load balancer and scheduling algorithm, responsible for allocating incoming tasks to appropriate resources, is provided.

4.1.1 Experimental Setup

The design and implementation architecture of the integration between local resources and cloud computing resources is presented in this section. As shown in Figure 4.1, the Raspberry Pi has been used to collect data from different types of sensors, such as temperature and humidity sensors. The local load balancer and scheduling algorithm transfer incoming

tasks to appropriate resources. In the Azure cloud provider, we also configure a load balancing technique that plays an important role in order to increase the availability of IoT applications by handling incoming tasks across multiple VMs.

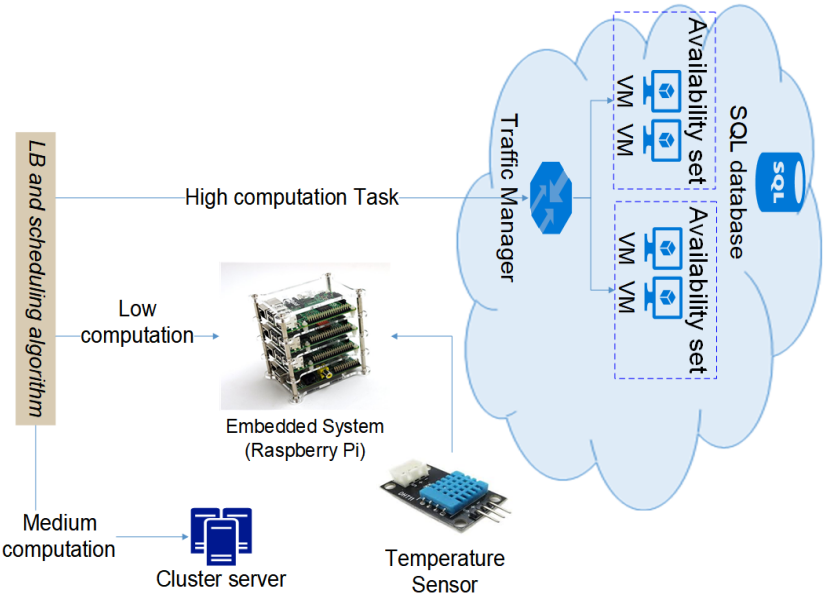


Figure 4.1: Prototype implementation

4.1.2 Core Implementation Components

Local resources in this prototype consist of three components: an embedded system –an IoT device–, a local server, and a local load balancer. On the other hand, the main components of the cloud resources are the virtual machines (VMs) and the Azure load balancer. In this implementation, the Microsoft Azure cloud has been utilized. The following subsection discusses the components of the architecture and technologies applied to implement this

framework.

Local Resources

Local resources in this prototype consist of three components: an embedded system, a local server, and a local load balancer.

- **Embedded System**: A Raspberry Pi 2 Model B with 900MHz quad-core ARM Cortex-A7 CPU and 1GB RAM has been used as an embedded system.
- **Local Server**: A 64 bit Windows 10 PC with Intel(R) Core(TM) i5-4210U CPU @ 1.70 GHz 2.40 GHz processor and 8GB RAM has been setup to serve as the local server in this prototype implementation.
- **Load Balancer and Scheduling Algorithm**: A 64 bit Windows 10 PC with Intel(R) Core(TM) i7-3630QM CPU @2.40 GHz processor and 16GB RAM is used as the load balancer. The load balancer in this implementation has been coded in C# in order to implement the scheduling algorithm shown in Figure 4.1.

Cloud Resources

The main components of the cloud resources are the virtual machines (VMs) and Azure Load Balancer. The cloud Microsoft Azure has been used in this implementation. [30, 37].

- **Virtual Machines (VMs)**:

  In the implementation, 4 VMs have been created across EAST US and CENTRAL US regions with 2 VMs each. The VMs used Windows Server 2012 as the operating system. All VMs are in the standard pay-as-you-go plan provided by Azure and come with eight cores and 14 GB of memory, as shown in Figure 4.2.

- **Cloud Load Balancer**: In each region where VMs are deployed, there will be an internal load balancer in that region to balance the load between the deployed VMs. Also, there will be an external load balancer to balance the load between the regions as well, as shown in Figure 4.2.

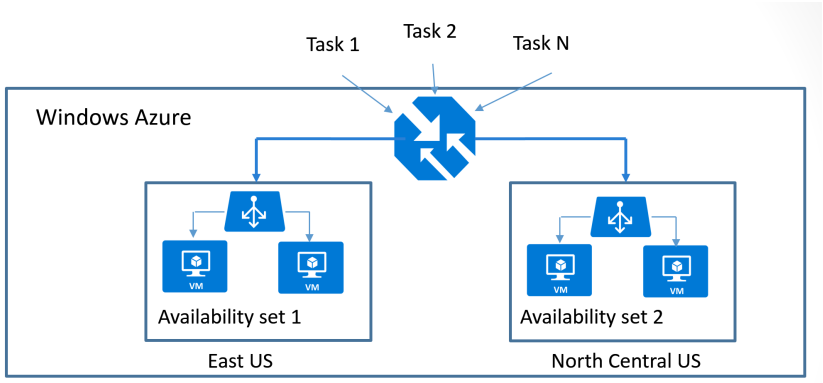


Figure 4.2: Implementation of cloud components

4.1.3 Local Load Balancer and Scheduling Algorithm

The scheduling algorithm plays a significant role in transferring incoming requests based on task and computation types. We classify the task in terms of priority into two types:

low priority task and *high priority task*. In addition, we classify the task in terms of computation into three types *low, medium, high computation.*

For the purpose of simulating various types of computing task *(low, medium, and high)*, the Bubble sort algorithm was used. For the prototype implementation, three sets having set 1:{1-10}, set 2:{1-100} and set 3:{1-1000} numbers have been considered. Sorting set 1 using *Bubble sort algorithm* will be considered a low computational task, sorting set 2 as a medium computational task, and sorting set 3 as a high computational task. In order to do the evaluation, a local load scheduler has been implemented on a local machine. A task will be generated whenever the load balancer is called, which could be a low, medium or high computation task. If a low computation task is generated, it will be executed on Raspberry Pi which consider as an embedded local resource. When a medium computation task occurs, an application that does bubble sort will be called, running on the local server.

4.1.4 Evaluation Results

Evaluation of the prototype implemented has been done based on :

- *Response time*: measuring the performance of the system by comparing the average response when using local resources only and when integrated with the cloud. Response time is the difference between the time when the request was sent and the time when the response has been fully received.
- *Throughput*: it is the number of requests per unit of time. We divide the total average

time by 1000 to convert from ms to seconds, as shown in equation 4.1.

$$Throughput = \frac{Numberofrequests}{TotalResponseTime/1000} \tag{4.1}$$

All test cases are done with various ranges of the number of requests to understand the effect of loading. In order to simulate this scenario, we used ***Apache JMeter***, which could generate a large number of tasks to the assigned server and get back the results such as response times and throughput [66].

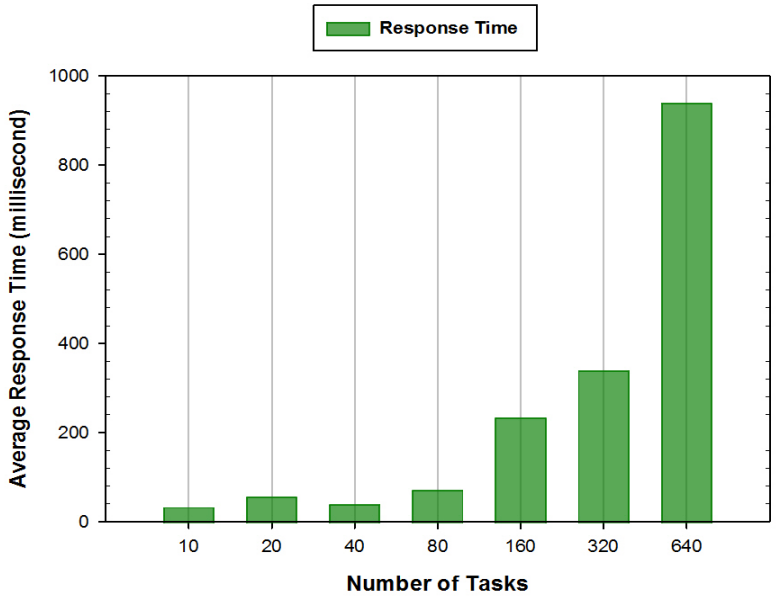


Figure 4.3: System performance using only local resources.

Figure 4.3 shows the performance result of executing the classified tasks using local resources. The results indicate that the system has high performance when the number of tasks is less than 80. Following this, the average response time increases, to a peak of 220 ms, for executing 160 tasks. In addition, we noticed that, at this point, some tasks are executed with a low-performance peak of 1433 ms. There was a sharp increase in the average response time when the number of tasks rose to 640. At this point, we noticed that some requests failed while others took a longer period to be processed. As a result, when a low number of tasks are executed on the local resources, the system can handle the incoming tasks with high performance. However, when the number of tasks is increased, the average response time peaks to high response time. Thus, local resources are not able to execute a large number of tasks with high performance.

In order to complete the experiment, the local resources have been extended using Azure, and we added a cloud endpoint to the scheduling algorithm and load balancer. Then, we generated a different number of tasks. Figure 4.4 shows the performance result of executing different types of tasks after integrating local resources with the cloud. The results indicate that the system's performance improved after extending the local resources to the cloud. Thus, it is evident that, after using the cloud, the system was able to serve up to 640 tasks with high performance within a low response time. We noticed that when requests surpassed 640, the response time of some requests increased.

As shown in Figure 4.5, the throughput also has been increased significantly when cloud resources are integrated into the system.

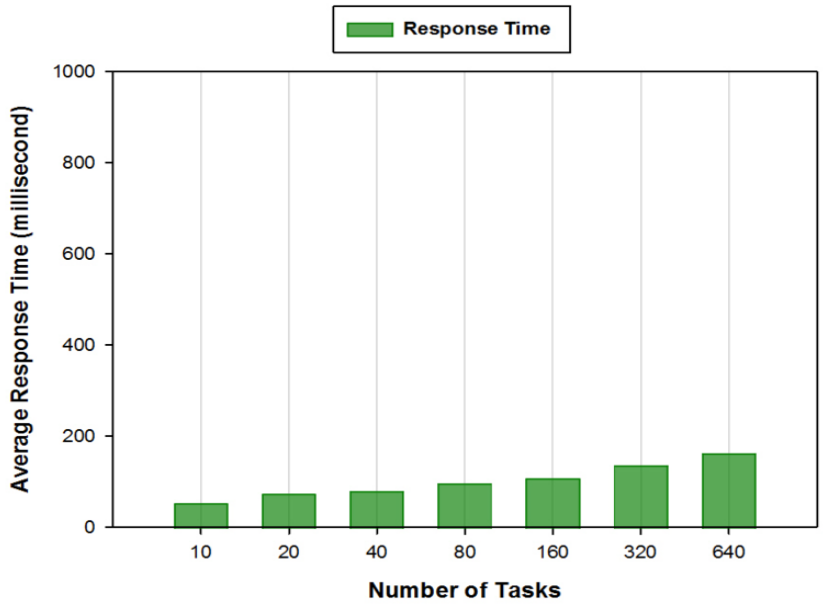


Figure 4.4: System performance after integrating local and cloud resources.

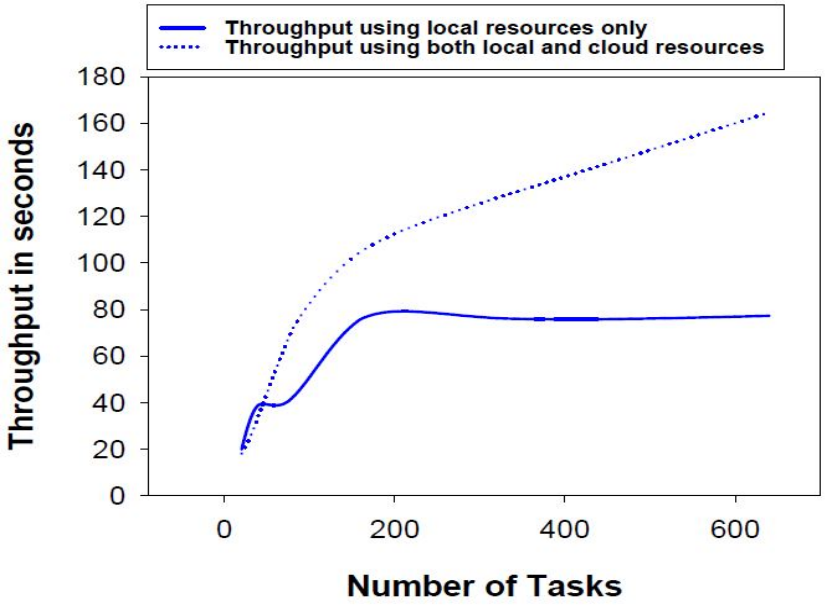


Figure 4.5: Local resources throughput vs combined throughput

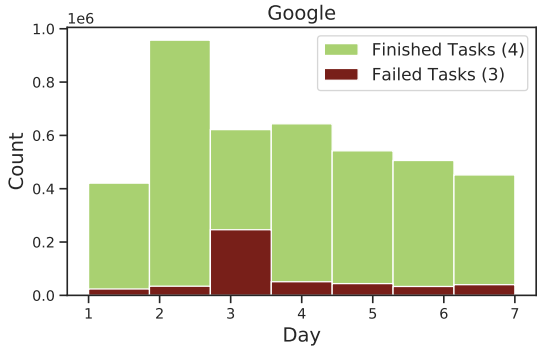
In summary, we have designed and implemented a framework for extending the local resources of IoT devices, which are very limited in terms of computing, memory, and storage, using cloud computing. This framework provides scalability and high availability for IoT applications. Furthermore, a scheduling algorithm is designed to reduce the execution time and computing cost associated with using the cloud, and it operates according to a logic that is directly dependent on the priority and computation requirements of the tasks being scheduled. Windows Azure services have been used to implement the framework, and it has been evaluated in terms of performance. The results reveal that after integrating the local resources, including the embedded system and IoT devices, to the cloud, the framework's performance and throughput efficiency increased by 55%.

4.2 Failure Analysis of Traces

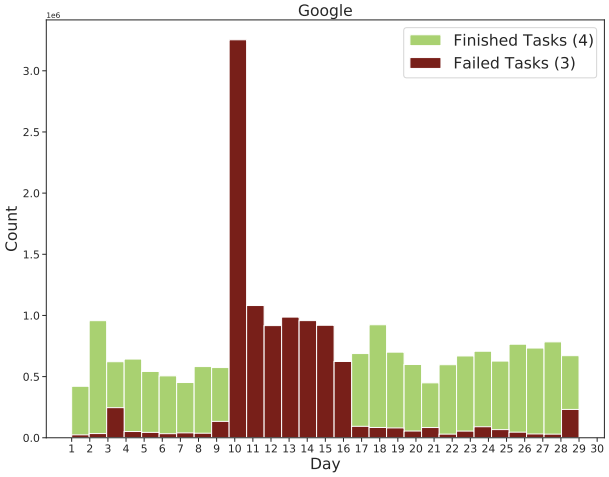
We have analyzed and utilized three large-scale traces accumulated by a variety of entities, including Google and the Los Alamos National Laboratory (LANL). The following subsections provide a more extensive explanation of the traces utilized. The basic description and characteristics of the clusters having traces in the Atlas repository and Google cluster traces are shown in Table 4.1. In comparison to other traces, the Google trace has a significant failure rate, but it also contains over 28 million tasks submitted, making it the most comprehensive trace available. The LANL Mustang trace has the longest trace time compared to the other traces.

Table 4.1: Basic cluster description and attributes from the Atlas repository and Google cluster traces.

Dataset	Nodes	Sample size	Features	Failed ratio (%)	Length
Google Cluster	12550	28,546,501	11	36.2	29 days
LANL Mustang	1600	2,113,175	9	7.2	5 years
LANL Trinity	9408	20,277	14	16.5	3 months



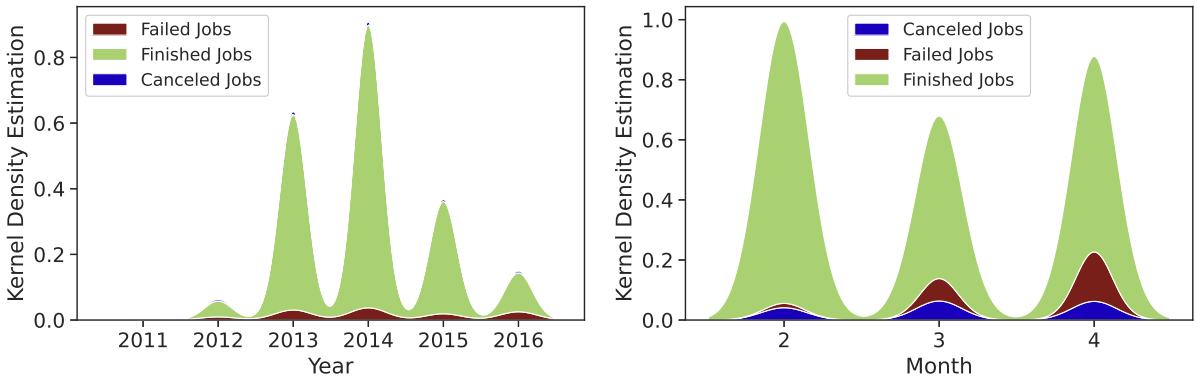
(a) Google trace in 7 days



(b) Google trace in 29 days

Figure 4.6: Distribution of task status for Google traces

We used all of the tasks submitted during the 29-day Google trace (500 files). All job events that did not occur in the trace window have been filtered out. This section aims to examine failure behaviour by concentrating on the most significant events ("Fail" and "Finish"), which are represented as "3", and "4" in the Google trace, respectively. Figure 4.6 (a) depicts the Google trace distribution for failed and completed tasks over seven days. On the first and second days of the trace, around 97% of tasks are performed, with a minimal rate of failed jobs (roughly 3%). Between the third and fourth days of the Google trace, the percentage of finished tasks decreased to approximately 71%, while the rate of failed tasks increased to 29%. Figure 4.6 (b) depicts the Google trace distribution



(a) Mustang trace

(b) Trinity trace

Figure 4.7: Distribution of job status for Mustang and Trinity traces

for failed and completed tasks over 29 days. Between the day 10 and day 16, the number of unsuccessful tasks increased sharply.

Figure 4.7 (a) shows the distribution of Mustang trace for failed and finished tasks. The number of tasks submitted in 2011 and 2012 is very low. However, the number of tasks submitted has increased from the end of 2012 to the first three months of 2014. We also noticed that the number of completed tasks had increased significantly between the last four months of 2013 and the first three months of 2014.

Figure 4.7 (b) depicts the distribution of the Trinity trace for successful and unsuccessful tasks. For the second, third, and fourth months, the Trinity trace received 8,601, 5,699, and 7,203 jobs, respectively. Thus, the submitted jobs in the second and fourth months are high compared to the third month. However, the failure rate is approximately 15% for the fourth month, which is very high compared to the failure rate in the months two and three 1% and 8%, respectively. The overall failure rate is 7.2%, and almost 65% of failure

happens in month four of the trace.

4.2.1 Google Cluster Traces

Google has made a massive dataset called Google cluster traces publicly accessible. This dataset contains a large number of monitored data files that were created by a extensive system with over 12,500 nodes. There are 672,074 jobs and over 28 million tasks in the Google cluster traces submitted between May 1st and May 29th. In this study, we have used a version 2 of this dataset which is ClusterData2011-2 [106]. Each job consists of one or several tasks, and each task contains some processes. In addition, we have found that each task has different event types, including submit, fail, finish, kill, and evict. Also, the submitted tasks have different characteristics such as priority, class schedule, requested resources (CPU, memory, disk space), and actual resource usage. This dataset includes five different tables: Jobs Events, Task Events, Task Resource Usage, Machine Events, and Machine Attribute tables. A brief description of each of the tables are presented as follows:

Task Events Table

This table presents a detailed description of each task related to a particular job. Each job consists of one or several tasks, and each task contains some processes. Task Events Table includes thirteen features, including a timestamp, missing information, index of tasks, event type, the priority of a task which can be free priorities or a production priority or

perhaps a monitoring priority, information about resource requests. Our analysis primarily focuses on this table because it includes many features that can be used for failure analysis, such as task scheduling class, priority level, and requested resources.

Task Resource Usage Table

Task Resource Usage Table consists of a report of resource usage every 5 minutes. This table includes sixteen features, including a start time and end time, task index, CPU rate, memory usage information, the amount of assigned memory, cache memory usage, disk I/O time, sampling rate, cycles, MAI. Local disk space usage and aggregation type as the fields.

Machine Events Table

This table gives a detailed description of each machine, including machine ID and timestamp that indicates when this machine was started, the size of RAM, CPU capacity, and the status of the different events, which are ADD (0), REMOVE (1) and UPDATE (2).

Machine Attributes Table

This table shows properties description of available machines such as a version of machine kernel and clock speed. Machine Attributes Table contains five features, including timestamp, name, attribute value, and machine ID.

In this study, we are interested in analyzing the scheduling jobs and tasks behaviour in the available workload, especially "Fail" and "Finish" jobs/tasks. Thus, we have used the "Job and Task Events Tables" because they include the event type of each task and other features related to the required CPU and RAM for each task.

Table 4.2: Google trace overview

Trace characteristic	**Value**
Total number of users	933 users
Submitted jobs	676,975 jobs
Scheduling jobs	676,967 jobs
Finished jobs	386,218 jobs
Failed jobs	10,133 jobs
Killed jobs	272,609 jobs
Evict jobs	22 jobs
Lost jobs	16 jobs
Submitted tasks	48,330,301 tasks
Scheduling tasks	47,306,307 tasks
Finished tasks	18,187,970 tasks
Failed tasks	13,828,583 tasks
Killed tasks	10,337,327 tasks
Evict tasks	5,864,223 tasks
Lost tasks	8754 tasks

A further observation is that each task has distinct features depending on its scheduling class, priority, requested resources, and actual resource consumption. We summarize basic statistical information of Google traces in Table 4.2. In the following subsections, we explain the correlation between failed tasks and priority levels, scheduling classes and requested resources.

The basic scenario of the job or task life cycle is shown in Figure 4.8; a job is submitted and then inserted into a pending queue. Then, the job is scheduled onto an available

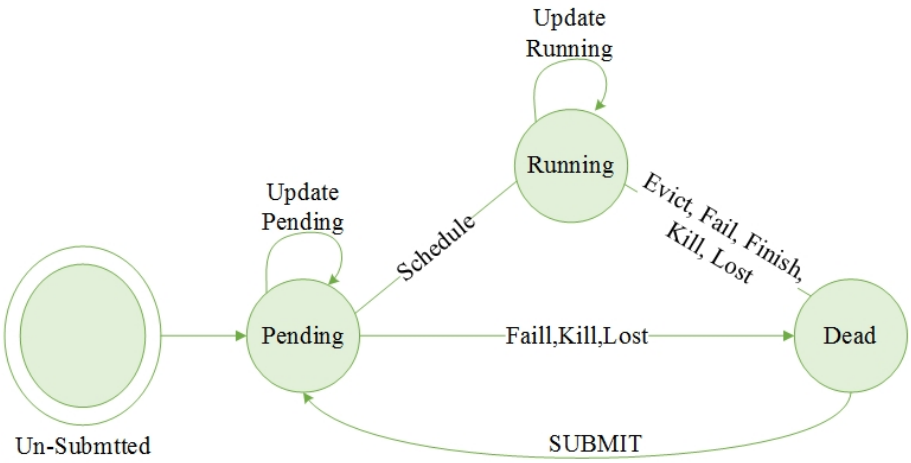


Figure 4.8: Event types

machine, and it starts running. Then, after a period which considers as execution time, the job is finished successfully. If a job or task has just failed, the cluster system attempts to restart tasks that have failed. Some jobs have been recorded with the timestamps are 0, these jobs or tasks were submitted before the trace began. There is no information about the cause of task failure. More details on Google cluster trace can be found in [106].

Trace processing and Data Filtering

Data preprocessing is the most important step in the data science world. Thus, we have to filter out some records from the dataset in order to be ready for our purpose analysis. The preprocessing steps are presented as follows:

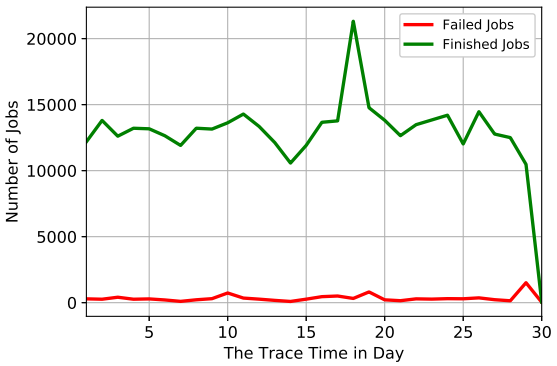
1. Filtering out all jobs/tasks events that did not occur within the trace window which are presented as a time of 0.

2. Converting timestamp from microsecond to daytime in order to be more usable.

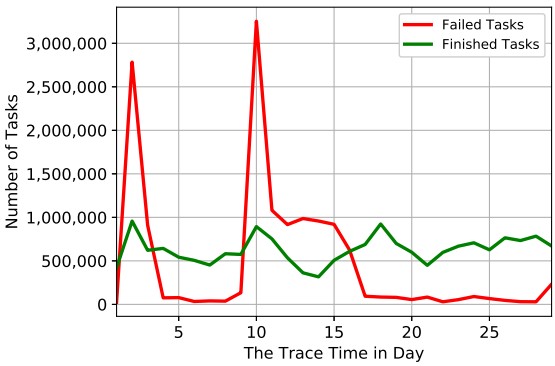
3. Our analysis is based on the first week of the trace. Basic statistics about Google cluster trace is summarized in Table 4.2.

4. Once the Google cluster trace has been analyzed, we observed that the number of failed tasks is much higher than the number of failed jobs in the Job Event table. As a result of our investigation, we found that some tasks have been resubmitted numerous times before being successfully finished. However, these tasks were killed after lots of resources is being consumed. Thus, we decided to filter out these five jobs because they are considered as outlier cases. Moreover, these jobs have increased the number of task failure up to 3,942,709. We also note that most failures occur on the second day, especially between 1 a.m. and 8 a.m. The number of failures has significantly increased on the second day to be 2,450,884, as shown in Figures 4.9 and 4.10. Thus, we expect that a part of the data center had an outage during this time.

We investigate the failure behaviour for 29 days in the preprocessing data phase. Figure 4.9 shows a comparison between job and task event failure behaviour in 29 days of Google trace. In Figure 4.9(a) show that failed and finished jobs follow normal behaviour. However, as shown in Figure 4.9(b), the number of failures has significantly increased in the second and tenth days of Google's trace. Figure 4.10 presents many failed tasks during the Google trace period, focusing on the trace's second and tenth days. On the second day of the

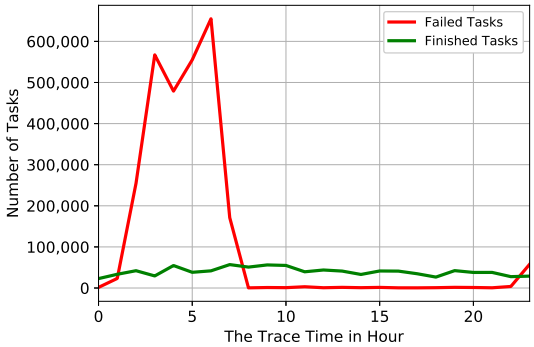


(a) Failed and finished jobs for 29 days of the Google trace

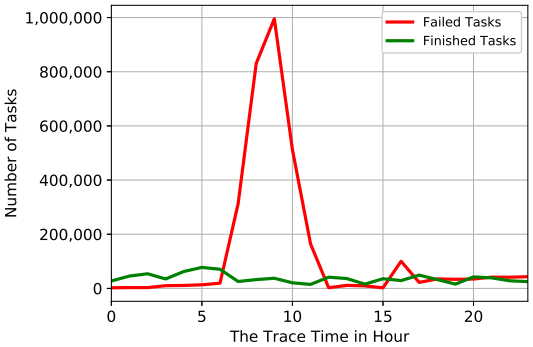


(b) Failed and finished Tasks for 29 days of the Google trace

Figure 4.9: A comparison between job and task event failure behaviour in 29 days of Google trace



(a) Day 2 of the Google traces



(b) Day 10 of the Google traces

Figure 4.10: Number of failed tasks for the Google trace, focusing on days 2 and 10

trace, we noticed that most task failures occurred between 1 a.m. and 8 a.m., as shown in Figure 4.10(a). On the tenth day of the trace, most task failures occurred between 6 a.m. and 12 p.m, as shown in Figure 4.10(b). Therefore, we estimate a portion of the data centre to be unavailable during this time.

Failure and Priority Levels

A priority level is assigned to each task, represented by a tiny number that is translated into a sorted collection of values, with (0) being the lowest priority (least significant). Tasks with higher priority numbers often receive more resources than tasks with lower priority numbers. The average priority level for failed and completed jobs during the trace period is shown in Figure 4.11. The results indicate that most unsuccessful jobs, notably on the third day of Google's trace, are medium or high priority. In the case of tasks with priority levels of more than 3, the likelihood of them falling increases significantly. As a result, a positive correlation exists between failed tasks and task priority levels. In order to increase the productivity of cloud infrastructure and applications while reducing the risk of failure, priority strategies must be developed. Nearly half of all tasks have a priority level of (0) out of (9), and 28.4% have a medium priority level of (4).

Moreover, resource demands indicate the maximum amount of CPU, memory, or disc space used by a task. Tasks that use more than their allocation of resources may be throttled or terminated. Consider the possibility that a machine's scheduler may over-commit resources. Even if each job is using less than its maximum, there may not be enough resources to meet all of the task's runtime requirements. This might result in the termination of one or more lower-priority tasks.

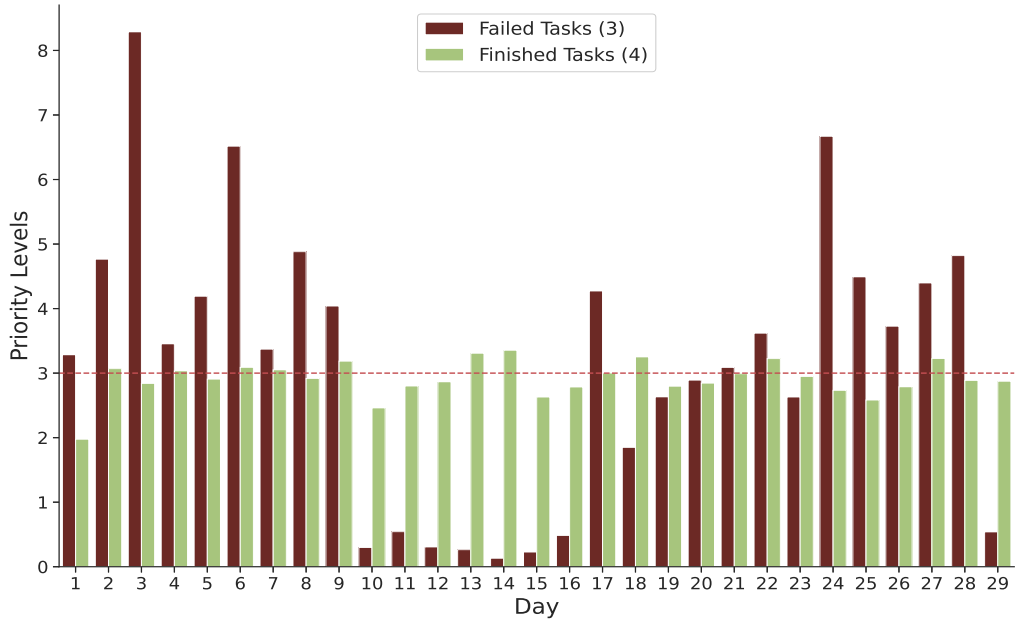


Figure 4.11: Priority level for failed and finished tasks

Failure and Scheduling Class

All jobs and tasks are assigned a scheduling class that corresponds to how latency-sensitive they are. A single number expresses the scheduling class, with 3 denoting a more latency-sensitive job. The relationship between the scheduling class and *failed/finished* tasks is depicted in Figure 4.12. Only roughly 1% of all tasks have a scheduling class of (3) while 73% and 18%, respectively, have scheduling classes of (0) and (1). There is no obvious correlation between failed or completed tasks and lower scheduling classes. However, most completed and unsuccessful jobs have a lower scheduling class (0,1), while a smaller percentage of completed and failed tasks have a medium scheduling class (2). All tasks with a high scheduling class (3) are also unsuccessful. As a result, there is a correlation between

unsuccessful tasks and having a high scheduling class (3). As a result, a scheduling algorithm must be applied in order to improve the distributed efficiency of incoming tasks in terms of system availability and computing requirements.

Figure 4.12: Scheduling class for failed and finished tasks

The primary aim of the requested resource is to present the maximum amount of CPU, memory and disk space allowed. As a result, no task should be allowed to surpass its limits. When some jobs reach their limits, the management system can terminate them because they consume additional resources, such as memory. After submitting the tasks to be processed in certain cases, these tasks do not find sufficient resources to be fully executed, even if those tasks do not exceed their limits. Therefore, one or more tasks with low priority can be killed [106].

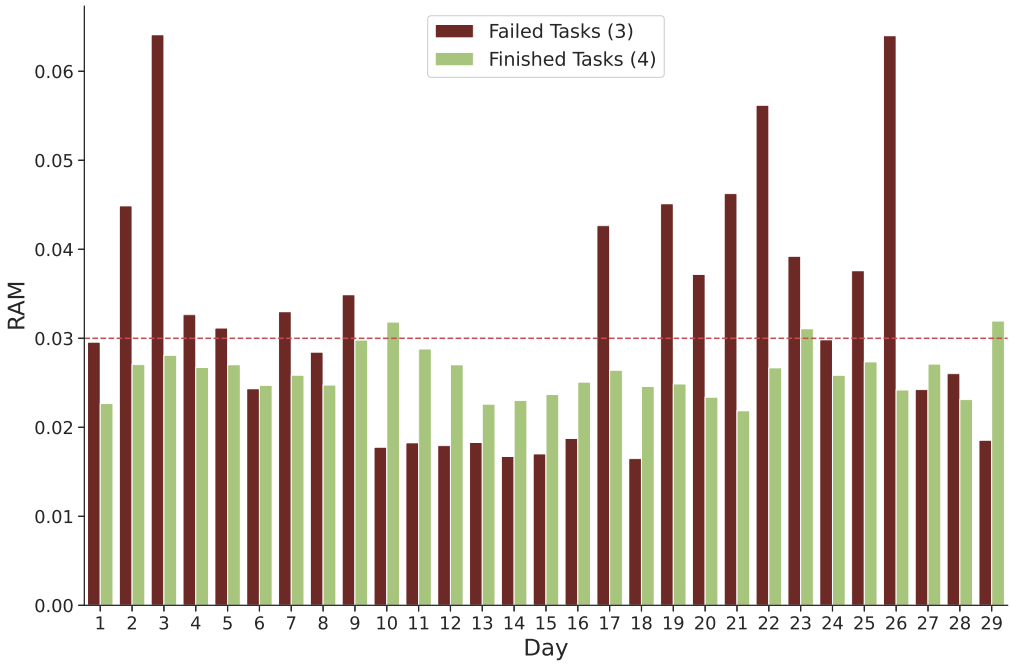


Figure 4.13: Memory was requested for both unsuccessful and finished tasks

Failure and Requested Memory

The failure of the task and the required memory have a clear relationship, as illustrated in Figure 4.13. Google trace does not provide the actual memory size consumed by tasks but rather its scaled value in comparison to the maximum memory and CPU capacity of each node. Assume the maximum memory capacity of a host is 64GB. 0.03 memory size corresponds to 0.03 x 64 = 1.92GB [41]. On the third day of the trace, for example, the number of failed tasks has climbed dramatically, and we see that most of these tasks demand more memory than the completed tasks. When the incoming tasks need a medium/high quantity of memory resources, the chances of failure have increased. When the requested

memory is greater than 0.03, the majority of tasks fail. On the other hand, completed tasks occur when the requested memory is less than 0.03. Scheduling algorithms need to consider the required amount of computation for each task to distribute incoming tasks to the right level of resource availability. As a result of this strategy, the number of unsuccessful tasks is reduced, resulting in enhanced cloud application availability.

It is important to note that the majority of resource use and request metrics have been normalized, including memory (bytes), disk space (bytes), CPU (core count or core-seconds/second), and disk time fraction (I/O seconds/second). Separate normalizations are computed for each of the primary variables. The scaling used for normalization is based on the greatest resource capacity of each machine in the trace (which is 1.0) [106].

Failure and Requested CPU

A positive correlation is also found between task failure and the requested CPU. Figure 4.14 shows how much the CPU has requested to perform most failed and completed tasks. When the requested CPU is more than 0.03, most tasks fail. On the other hand, finished tasks occur when the desired CPU is less than 0.03.

In the second and third days of the trace, the number of failed tasks has increased dramatically, similar to the desired memory performance. These tasks require a higher amount of memory than the tasks completed. As a result, we conclude that improving the existing scheduling strategy would increase the availability of cloud applications while also controlling the cloud infrastructure's resource usage.

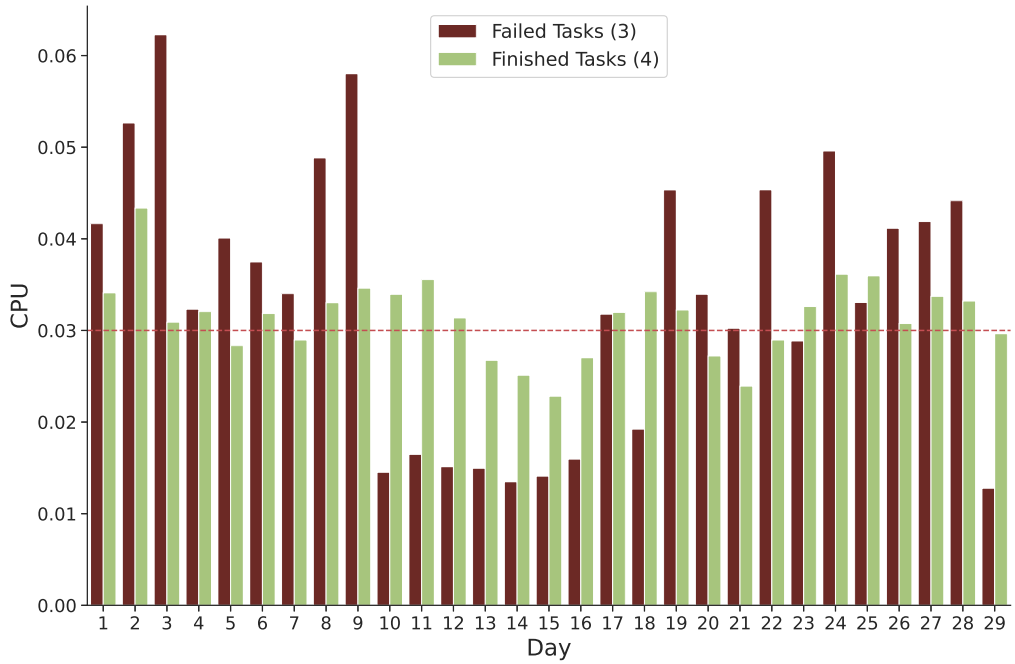


Figure 4.14: CPU was requested for both unsuccessful and finished tasks

Failure and Requested Disk Space

There is no clear correlation between task failure and the requested disk space, except on the seventh day of the trace, which shows a considerable increase in failed tasks when the requested disk space is more than 0.0006, as shown in Figure 4.15.

To this end, the results of our study contribute to investigating the behaviour of failed jobs and the relationship between failed jobs and other factors, including task scheduling and task priority. The job failure analysis can be used as a starting point for developing a new failure prediction model that can predict the types of cloud job events *failed/finished* in advance.

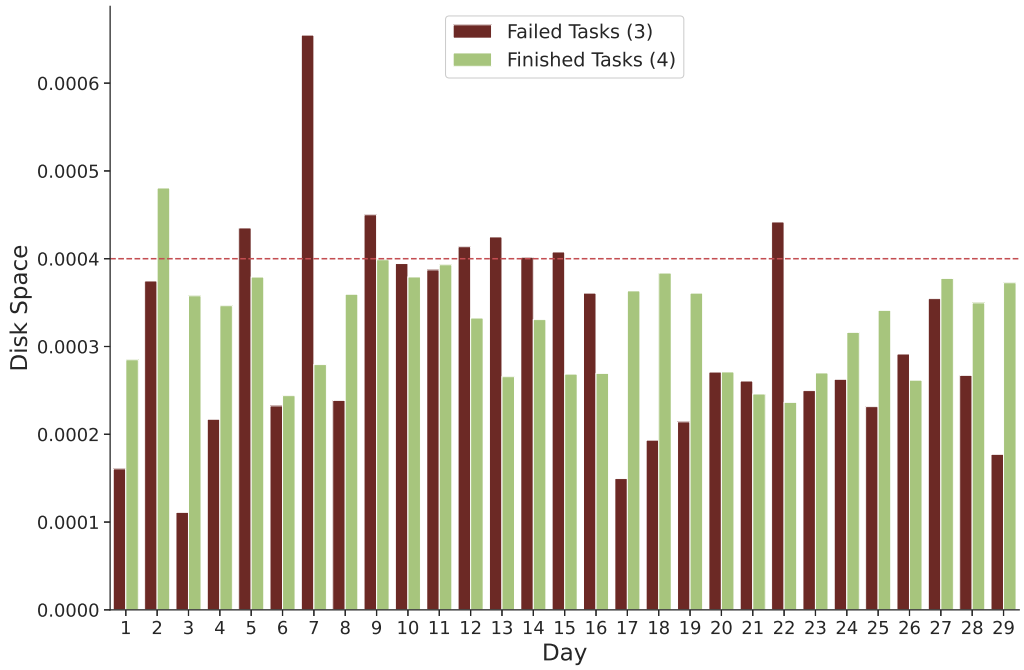


Figure 4.15: Disk space was requested for both unsuccessful and finished tasks

Cloud providers can employ novel solutions to limit unsuccessful tasks and jobs. For example, cloud providers can optimize a scheduling algorithm to maximize the efficiency of a cloud load balancer. The load balancer distributes incoming jobs based entirely on the resource computation required. Consequently, the number of unsuccessful tasks will be reduced, as all tasks will be allocated to the resources required regardless of resource availability, and all resources will be utilized effectively and efficiently.

Furthermore, we have found that failed jobs have significantly longer running times and consume significantly more resources than completed jobs. As a result, significant resources were spent on jobs that were not completed. The development of an early failure

prediction model would assist minimizing failure. We also noticed that failed jobs have a high number of resubmissions. In order to save cloud resources, it is recommended to reduce the number of resubmissions. Jobs and tasks at a high priority level are more likely to fail. As a result, the relationship between job priority and job failure should be taken into account when developing a fault prediction model.

4.2.2 LANL Mustang Cluster

Mustang is one of the HPC clusters used by LANL from October 2011 to November 2016 for capacity computers. We can thus conclude that the Mustang trace is the longest publicly available trace to date. The Mustang consisted of 1600 computing nodes with 102 TB RAM and 38,400 AMD Opteron 6176 2.3 GHz cores. The cluster was fundamentally used by software developers, engineers, and scientists at LANL. The trace contains 2.1 million jobs, and 565 users submit these jobs. Collected data include: start and end time for a job, job event types such as completed, failed, and cancelled [14, 15].

A positive correlation is also found between failed jobs and the number of tasks requested for each job. Figure 4.16 shows the average number of tasks requested between 2011 and 2016 for cancelled, failed and finished jobs. Most failed and cancelled jobs occurred when the average requested tasks were approximately over 500 tasks. In contrast, on average, tasks have been completed when the number of requested tasks is approximately 300 or less, particularly from 2013 to 2016.

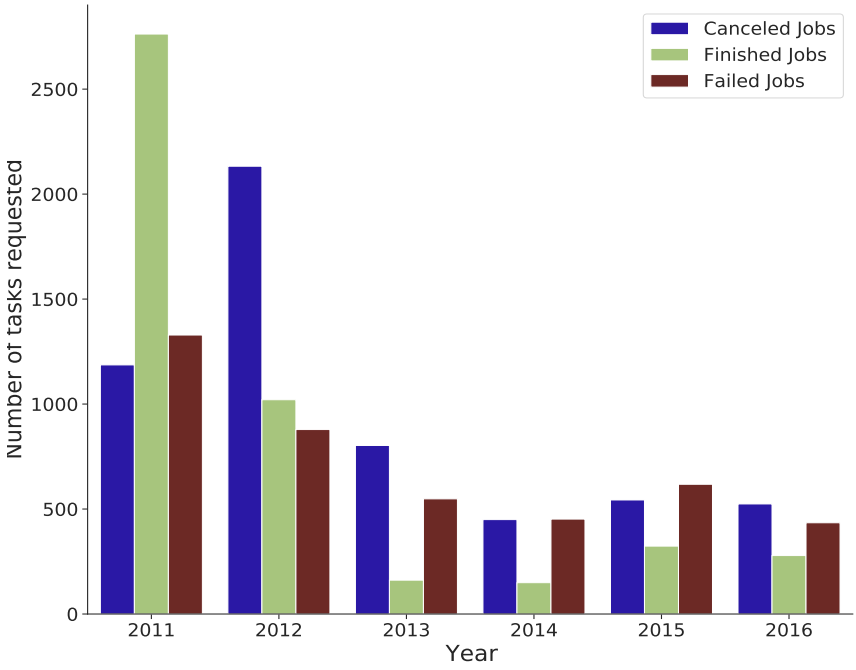


Figure 4.16: The average number of tasks requested from 2011 to 2016 for cancelled, failed and finished jobs

Additionally, there is a strong association between failed jobs and the number of nodes assigned to each job. Figure 4.17 shows the average number of nodes for cancelled, failed and finished jobs between 2011 and 2016. Most failed and cancelled jobs occurred when nodes were approximately over 18. In most cases, completed tasks occur when the number of nodes is about 11 tasks, especially from 2013 to 2016. Figure 4.18 illustrates the average number of nodes for cancelled, failed, and finished jobs at monthly intervals between 2011 and 2016.

There is also a significant correlation between the execution time and the number of failed, cancelled and finished jobs. As shown in Figure 4.19, the majority of failed jobs occurred when the average execution time exceeded 17500 seconds. However, most finished

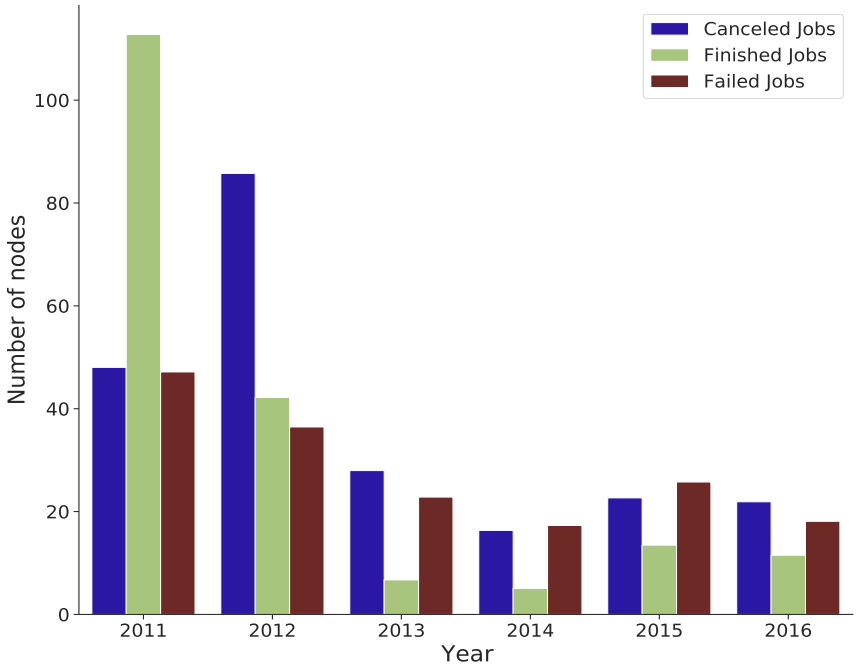


Figure 4.17: The average number of nodes from 2011 to 2016 for cancelled, failed and finished jobs

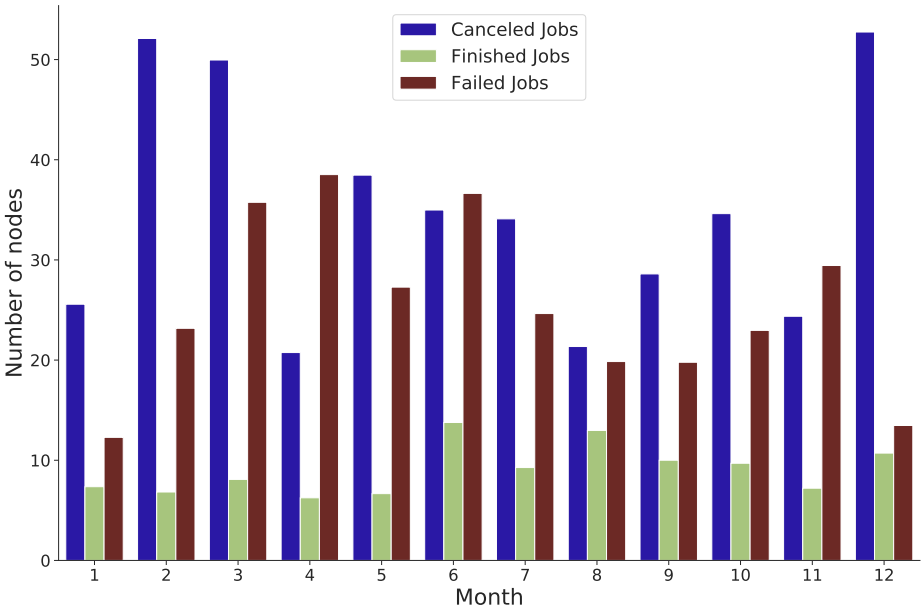


Figure 4.18: The average number of nodes for cancelled, failed and finished jobs in the month intervals between 2011 and 2016

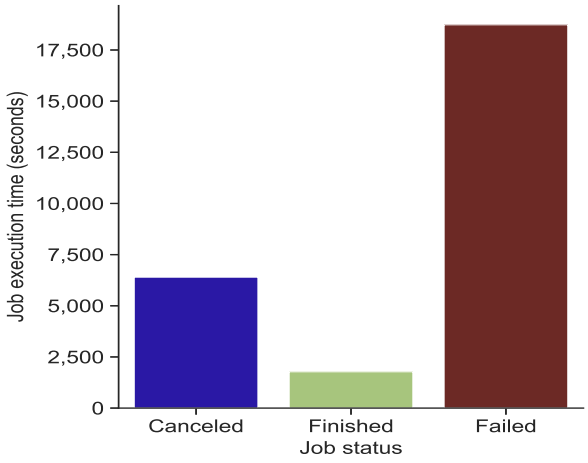


Figure 4.19: The correlation between the execution time and the failed, cancelled and finished jobs

jobs were successfully completed, with a short execution time of fewer than 2500 seconds.

4.2.3 LANL Trinity Supercomputer

Trinity is the largest supercomputer at Los Alamos National Laboratory (LANL) and is used for capacity computing. Trinity has 9408 compute nodes, each having 301,056 Intel Xeon E5-2698v3 2.3 GHz cores and 1.2 PB RAM. As a result, Trinity Trace is the largest cluster with a publicly available trace in terms of CPU cores. From February to April 2017, this data collection covers three months. This data set consists of 25,237 multi-node jobs issued by 88 users [14, 15].

As shown in Figure 4.20, Trinity trace has three categories of required class: "STANDARD", "DAT" and "CCM_QUEUE". Only 9% of submitted jobs are classified as "DAT" and less than 1% as "CCM_QUEUE", compared to more than 60% of submitted jobs

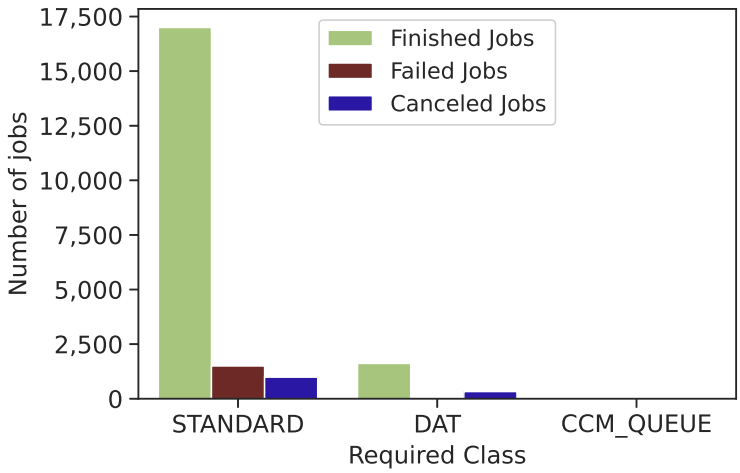


Figure 4.20: The correlation between types of Trinity required class and job status

classified as "STANDARD". We have found that most failed jobs are classified as "STANDARD". However, no failed job classified as "CCM_QUEUE". As a result, there is a high correlation between failed jobs and the required class attribute of the submitted jobs in the LANL Trinity trace. Hence, future work should consider designing and implementing a load balancer responsible for transferring incoming tasks based on the required class.

As presented in Figure 4.21, Trinity trace contains two distinct categories of resources: "trinity" and "internal". approximately 60% of submitted jobs were scheduled to utilize "internal" resources, whereas roughly 40% were scheduled using "trinity" resources. We have found that most failed jobs are scheduled in internal resources. There is a high correlation between failed jobs and the attribute of the resource manager. We expected that one or more components could fail due to a failure of a network, hardware or software component.

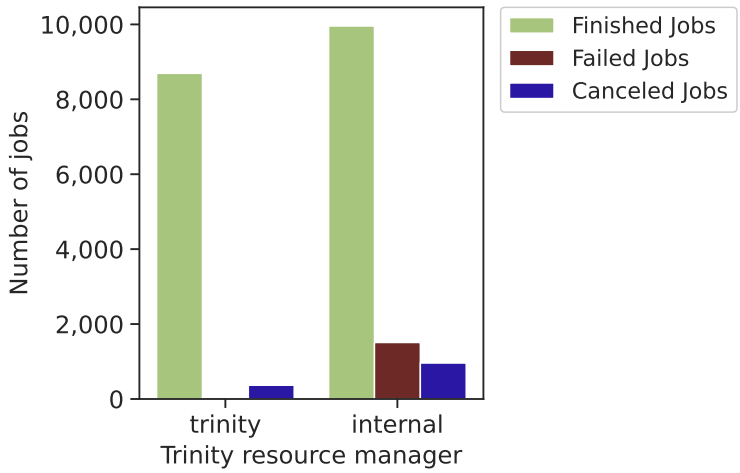


Figure 4.21: The correlation between types of Trinity computing resources and job status

4.3 Failure Prediction Model using ML

This section describes how the proposed failure prediction model is implemented using several machine learning algorithms. The evaluation results are then presented by comparing our proposed model to other ML models using different evaluation matrices, including precision, recall, and F1-score.

4.3.1 Experimental Setup

Comma Separated Value (CSV) files are used to store all used traces. Google, Mustang, and Trinity have a trace size of around 15 GB, 280 MB, and 14 MB, respectively. Python data frames are used to store data. The failure prediction model is then implemented using scikit-learn, a python machine learning program [97]. We have performed this experiment

on “Google Colab Pro+” because Google Trace has a big data size that requires a high computational processing power to analyze and predict. The Google Colab Pro+ resources that we used were 64GB of RAM and 256 GB of disc storage.

4.3.2 Evaluation Metrics

An essential part of data science is evaluating the accuracy of a model's predictions. It is crucial to take into account the following metrics in the evaluation of the proposed model: accuracy, precision, recall and F1-score. Before we dive into the detailed examination of model performance, we introduce the concept of the confusion matrix and evaluation metrics.

Confusion matrix

A confusion matrix is a table indicating the effectiveness of a classification model. The matrix rows correspond to observations in a target class, while the columns correspond to observations in an actual class. All evaluation metrics can be calculated using the values of the confusion matrix. True Positive (TP) and True Negative (TN) observations are made for the correct prediction results. Because False Positive (FP) and False Negative (FN) are observations of inaccurate prediction results, we are interested in lowering false positive and false negative to increase the accuracy of our failure prediction model.

Accuracy

The first important step in evaluating the classification model is to verify the accuracy of prediction. The ratio of correctly predicted samples to total samples is used to determine the accuracy. It is incorrect to guarantee that our model is the best if we have only achieved high accuracy. If the False Negative Rate (FNR) and False Positive Rate (FPR) are nearly equal, accuracy is an acceptable metric. As a result, other metrics must be considered when evaluating the proposed model results. For our failure prediction model, we have achieved a high accuracy of 0.99. The following equation is used to determine the accuracy.

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

Precision

Another metric of model performance is precision, which is calculated as the ratio of correctly predicted positive samples to total predicted positive samples. This metric will address a critical question: how many jobs/tasks succeed (finish) out of all jobs/tasks tagged as finished? We have a high precision since we have a low FPR. We achieved high precision in both event classes (failure and success), with the precision of 0.99 and 0.99, respectively. The precision has been measured using the following formula:

$$Precision = \frac{TP}{TP + FP} \tag{4.3}$$

Recall

Another metric for measuring model performance is recall, which is calculated by dividing the number of correctly predicted positive samples by the total number of samples in the actual class. The recall provides an important answer to an important question: how many jobs/tasks did we correctly label as completed (finish) out of all the jobs/tasks that have been successful? We achieved a high recall in both classes of Google traces (failure and success), with values of 0.99 and 0.99, respectively. The overall recall rate is 0.99. The recall was computed using the following equation:

$$Recall = \frac{TP}{TP + FN} \tag{4.4}$$

F1-score

The weighted average of recall and precision is the F1-score. In some instances, the F1-score is more significant than accuracy. F1-score, in many cases, is more important than accuracy. We have achieved a high F1-score in both classes of event types (fail, success), which are 0.99 and 0.99, respectively. The overall average of the F1-score is 0.99. Based on the following equation, the F1-score was calculated:

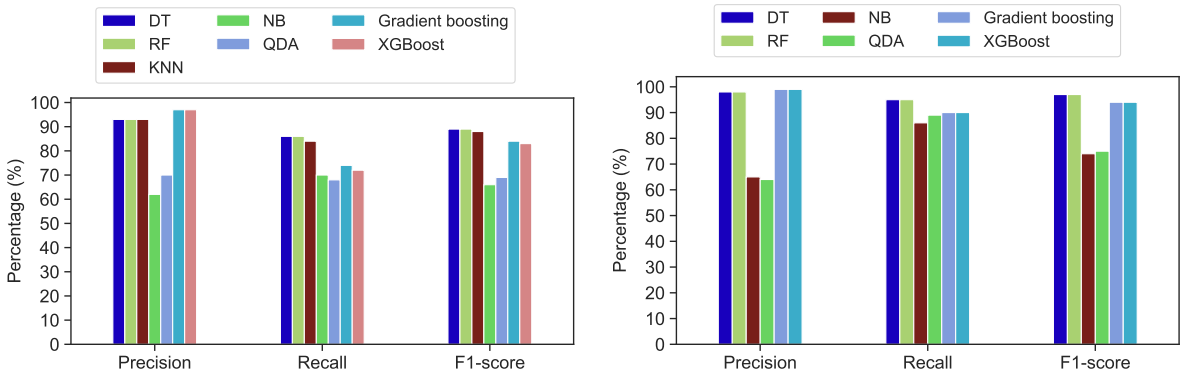
$$F1 - score = \frac{2 * (Recall * Precision)}{Recall + Precision} \tag{4.5}$$

4.3.3 Classifiers and Prediction Techniques

The aim is to build a prediction model that can predict the value of the target variables after preprocessing the datasets. Several ML classification algorithms can be applied to the cloud workload traces. However, we are interested in finding the most accurate classifiers, which can be learned with limited training data.

For Google Trace, our proposed model [60] has achieved high accuracy in predicting failed jobs using DTs and RF algorithms. However, our proposed model should be generic and applicable to various datasets. Therefore, we are interested in applying multiple classification algorithms to three different traces in order to select the best model with the highest accuracy. This section examines some classification algorithms, which can be applied and evaluated with different traces. The performance of each algorithm is then assessed using multiple evaluation metrics. We separated the Google, Mustang, and Trinity traces into two parts: a training set containing 75% of the data and a testing set containing 25% of the data.

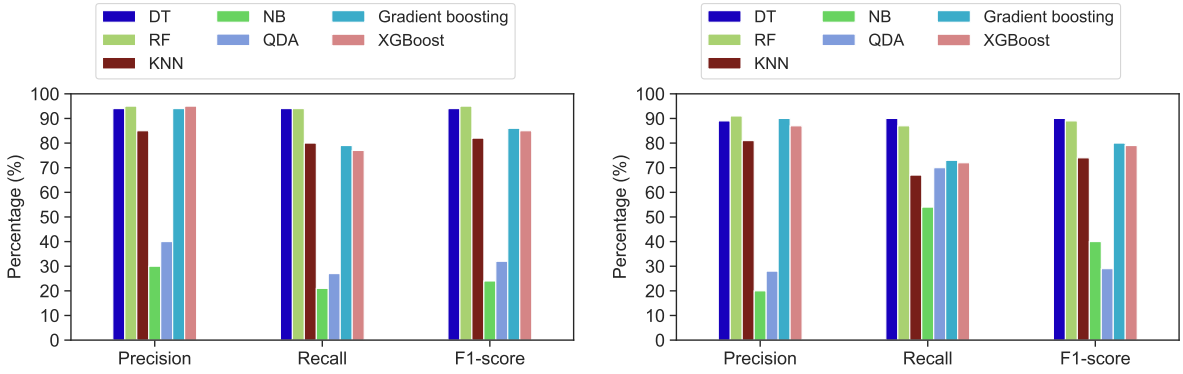
Figure 4.22(a) depicts the evaluation results of various classification algorithms when applied to the first week of the Google trace. Precision, Recall, and F1-score accuracy for both the RF and DT classifiers are 93%, 86%, and 89%, respectively. However, the results of the evaluation metrics have increased to 98%, 95% and 97%, respectively, after applying DTs and RF to all Google cluster trace observations (one-month period of the trace) as shown in Figure 4.22(b).



(a) Google trace in 7 days

(b) Google trace in 29 days

Figure 4.22: Performance evaluation of different algorithms applied to the Google trace



(a) Mustang trace

(b) Trinity trace

Figure 4.23: Performance evaluation of different Machine Learning algorithms applied to the Mustang and Trinity traces

The same classifiers were applied to the traces of Mustang and Trinity, as shown in Figure 4.23. The results show a comparison of the performance of various algorithms on the Mustang and Trinity traces. RF and DTs have the highest accuracy compared to other classifiers. The DTs algorithms' precision, recall, and F1-score for Mustang trace are 94%, 94%, and 94%, respectively. The precision, recall, and F1-score accuracy for the RF algorithms are 95%, 94%, and 95%, respectively. Also, for medium and small traces, the

RF classifier has achieved the highest accuracy compared to other classifiers.

For Trinity trace, the number of observations is 20,277, which is less than other traces. Therefore, all classifiers in Trinity have achieved lower accuracy than the other traces. The DTs algorithm has precision, recall, and F1-score accuracy of 88%, 85%, and 87%, respectively. The RF has precision, recall, and F1-score accuracy of 89%, 85%, and 87%, respectively. The DT and RF have achieved the highest accuracy for Trinity trace than other classifiers.

KNN, XGBoot, and Gradient Boosting have performed well compared to NB and QDA, with the lowest evaluation metrics for all traces. However, DTs and RF classifier has achieved the highest accuracy compared to other classifiers. Table 4.3 shows the training and testing time of Google, Mustang, and Trinity traces for utilized classifiers.

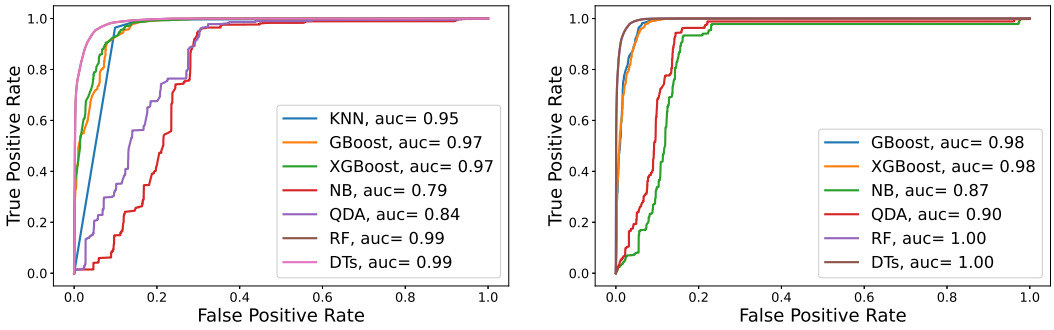
Table 4.3: Training and testing time and accuracy for all applied ML classifiers

		DTs	**RF**	**KNN**	**NB**	**Gradient boosting**	**XGBoost**
Google (29 days)	Training time	53.8	247.6	45.7	5.2	2093.8	2180.5
	Testing time	1.03	11	–	1	10.9	20.5
	Accuracy	98	98	–	78	96	96
Google (7 days)	Training time	13.23	75.7	5.6	0.8	344.46	182.1
	Testing time	0.16	1.9	2114.3	0.2	1.3	3.1
	Accuracy	98	98	97	92	97	97
Mustang	Training time	4.8	11.5	2.9	0.16	67.7	37.8
	Testing time	0.03	0.3	5.8	0.06	0.3	0.3
	Accuracy	99	99	95	86	97	97
Trinity	Training time	0.08	0.21	0.05	0.01	0.84	0.21
	Testing time	0.0009	0.007	0.16	0.001	0.007	0.007
	Accuracy	96	96	92	65	94	93

The results show that the DTs and RF can reach the highest accuracy, precision, recall, and F1-score. On the other hand, the RF-based model has the longest time, at

247.6 seconds, with a Google trace of 29 days, while DT has only 53.8 seconds. Thus, DT has less complexity than RF.

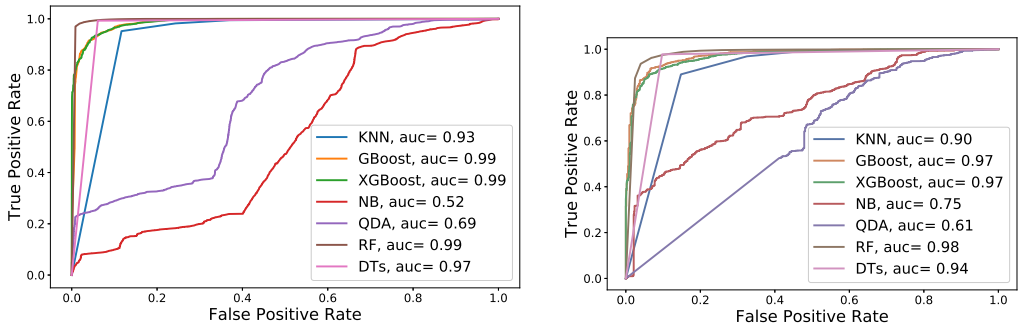
We attempted to apply KNN to all observations of Google cluster trace, but the processing time took more than 24 hours then the memory error occurred. Thus, even though the KNN ranks third in terms of all accuracy evaluation metrics, KNN is not recommended for failure prediction due to its log processing time.



(a) ROC for the first week of Google trace

(b) ROC for the all Google trace observations

Figure 4.24: Performance evaluation of different algorithms applied to the Google cluster trace



(a) ROC for Mustang trace

(b) ROC for Trinity trace

Figure 4.25: Performance evaluation of different algorithms applied to the Mustang and Trinity traces

Figure 4.24 depicts the Receiver Operating Characteristic (ROC) curves of the various

machine learning models for the first week of the Google trace compared to one month of the Google trace. When all Google trace observations are considered, DTs and RF have reached the maximum AUC (Area under Curve) with a high score of 100%, as shown in Figure 4.24(b). Gradient Boosting and XGBoost, on the other hand, come in second place with 98%.

Figure 4.25 depicts the Receiver Operating Characteristic (ROC) curves of the various machine learning models for the Mustang trace compared to the Trinity trace. RF has reached the maximum AUC (Area under Curve) for Mustang and Trinity with a high score of 99% and 98%, respectively, as shown in Figure 4.25(a) and Figure 4.25(b).

In the following part, we will discuss how to increase the accuracy of these two classifiers, RF and DTs, by using additional methods, such as feature selection.

4.3.4 Feature Selection Algorithms

Feature selection is one of the most important methods to improve the accuracy of our model by automatically selecting several features that contribute significantly to model performance. The predictive accuracy model will be reduced if the model has been trained based on irrelevant features. There are other advantages of using feature selection, such as reducing overfitting and training time.

SelectKBest:

The SelectKBest technique uses statistical analyses to select attributes with the closest relationship to the target variable. We obtained the most important features after using this technique for Google trace: job ID, task index, machine ID, priority, and scheduling class.

Feature Importance:

Bagged decision trees, such as RF, can be used to estimate which features contribute the most and can then be used as input to a failure prediction model to predict failure accurately. We obtained the most important features after applying this technique to Google trace: job ID, day, machine ID, disk space, and scheduling class.

Recursive Feature Elimination:

One of the most well-known feature selection techniques is recursive feature elimination (RFE). RFE works by eliminating features one by one and building a model based on the ones that remain. RFE uses the model accuracy to determine the most useful features in predicting the output target. After applying the RFE feature selection technique to Google trace, we obtained the most relevant features: job ID, hour, task index, machine ID and CPU.

Table 4.4 presents the evaluation results for the various feature selection algorithms

Table 4.4: Evaluation results of performing different feature selection techniques

		Decision Tree Classifier					Random Forest Classifier				
		Prec.	Rec.	F1-score	Train.(t)	Test.(t)	Prec.	Rec.	F1-score	Train.(t)	Test.(t)
Google	SelectKBest	97%	97%	97%	491.3	4.17	98%	97%	97%	2683	39.5
	Feature Importance	93%	93%	93%	518.3	7.9	95%	93%	94%	2924	60.40
	RFE	99%	99%	99%	542.18	3.39	99%	99%	99%	2812	36.43
Mustang	SelectKBest	92%	93%	93%	0.9	0.03	94%	94%	94%	9.66	0.32
	Feature Importance	94%	94%	94%	1.2	0.09	95%	94%	95%	10.3	0.56
	REF	93%	93%	93%	1.4	0.12	94%	93%	93%	11.1	0.73
Trinity	SelectKBest	72%	69%	70%	0.05	0.0008	72%	65%	69%	0.27	0.008
	Feature Importance	88%	85%	87%	0.07	0.001	89%	85%	87%	0.38	0.02
	RFE	84%	83%	84%	0.09	0.008	85%	81%	83%	0.58	0.09

used. The DTs and RF classifiers used the RFE technique to achieve the highest precision, recall and F1-score to predict failed classes, which were 99%. As a result, feature selection algorithms can improve the accuracy of the proposed model. When the feature selection algorithms are applied to Mustang training data, the RF classifier can more accurately predict the failed class. We achieved the highest accuracy for Trinity trace when combining the feature importance algorithm and the RF classifier. Precision, recall, and F1-score are 89%, 85%, and 87%, respectively.

We found that Decision Tree (DT) and Random Forest (RF) provided accurate results after evaluating the performance of various classification algorithms using three cloud traces. However, when it comes to the traces of Trinity and Mustang, the RF classifier surpasses the DTs. As a result, we have chosen the RF algorithm as a failure prediction model, which can be used on various traces to produce the most reliable results.

El-Sayed et al. [46] have developed a failure prediction model based on the Random

Forest (RF). The findings show that they can correctly recall up to 94% of all failed jobs with at least 95% accuracy. We have the highest precision, recall, and F1-score compared to previous studies. The DTs and RF classifiers used the RFE technique to obtain the highest accuracy for predicting failed class, which was 99% for precision, recall, and F1-score. However, we did not achieve the desired precision in the other two traces due to the small number of observations in Mustang and Trinity. Therefore, the classifier does not have sufficient data for the failed class to distinguish between the two classes.

4.4 Failure Prediction Model using Deep Learning

This section details the implementation of the proposed failure prediction model using the ANNs deep learning approach. The evaluation results are then provided by comparing the proposed deep-learning model with previously applied machine-learning models using a variety of evaluation matrices, including precision, recall, and F1-score. The following subsection presents a brief description of the data sets, experimental setup, model hyper-parameters and performance, and evaluation results.

4.4.1 Experimental Setup

All utilized traces are stored in Comma Separated Value (CSV) files. The workload trace sizes of Google, Mustang and Trinity are almost 15 GB, 280 MB and 14 MB, respectively. Datasets were stored in Python data frames. We have used two libraries of Python that

are designed for implementing Neural Networks, which are Keras and TensorFlow. Keras is a simple tool for building a neural network, and it is a high-level neural network API written in the Python programming language. Keras is running on top of Tensorflow or Theano. Tensorflow is an open-source software library for machine learning that is used in Google environments. Keras can be used as a deep learning library, and it also supports Convolutional and Recurrent Neural Networks. Keras operates seamlessly on both CPU and GPU [35, 36]. Our objective is to implement a failure prediction model that can early predict failure before happening. We run this experiment on our local server because Google Trace has big data that require high-performance computing (HPC) for analysis and prediction. Table 4.5 provides a comprehensive characterization of the server used for the experiment.

Table 4.5: The description of the server used for the experiment

Size	CPU Cores	Memory	Disk Size
Cisco UCS C220	16	128 GB	1024 GB

Our experiments use three different data sets with massive traces collected by various organizations, namely Google and Los Alamos National Laboratory (LANL). The following subsections describe in more detail the used traces, and Table 4.6 provides a general characterization of each trace. The Google cluster trace has a high failure rate compared to other traces. However, Google trace has more than 28 million of submitted tasks, so Google trace is the most extensive available trace.

Table 4.6: Basic description of each trace

Dataset	Num of nodes	Sample size	Features	Failed sample ratio (%)
Google (29 days)	12550	28,546,501	11	36.2
Google (7 days)	12550	4,615,419	11	10.2
Mustang	1600	997,961	13	10.6
Trinity	9408	20,277	14	16.5

4.4.2 Building Neural Network

There are three different types of layers, namely input, hidden and output layers. We pass our matrix of features and labels through the input layer. The perceptron functions are responsible for calculating an initial set of weights and then passing them on to a number of hidden layers. The final solution is presented in the output layer. The network's hyperparameters are the number of layers and the number of neurons in each layer. In order to find the most accurate solution, the data scientist should experiment with alternative hidden layer counts and hyperparameters. Figure 4.26 presents ANN architecture based on the proposed model.

In Google trace, the input is a feature of 11 values, and the output is a failed or successful task. We have tested several combinations of hyperparameter values to find the best selection that can achieve the highest accuracy, as shown in Table 4.7. Comparing the results of applying multiple hyperparameters requires a significant amount of processing time to test various values, resulting in expensive and intensive computations. To avoid this, we focused on the three most essential factors: the number of epochs, batch size, and optimizer.

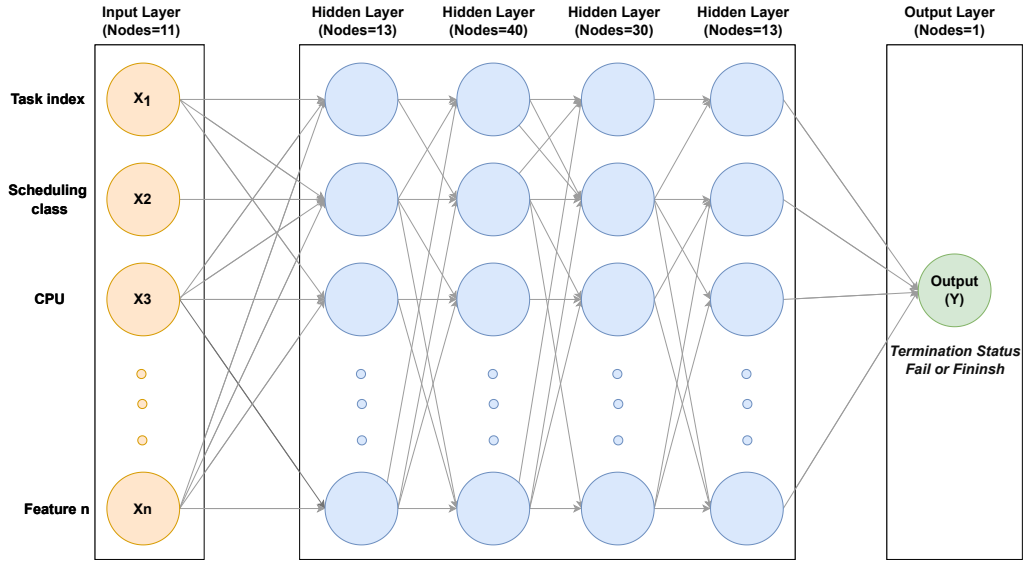


Figure 4.26: Architecture of Artificial Neural Network (ANN) based on the proposed model

Table 4.7: Hyperparameter tuning

Hyperparameter	Value Tested	Best Value
Number of epochs	10,30, 70, 100, 150	30
Batch size	32,64,128, 256, 512	512
Optimizer	“Adam”, “Adagrad” “SGD”, “RMSprop”	Adam

Hidden layers

Our neural network consists of four hidden layers. The hidden layers have 13, 40, 30, and 13 nodes, respectively. We have tried another number of layers until we finally find the optimal number of hidden layers and dimensions.

Activation function

The most significant aspect of an ANN architecture is the activation functions. The specified activation function has a major impact on the neural network's performance, and different activation functions are used in different parts of the model. We have used the Rectified Linear Unit (ReLU) function as an activation function for hidden layers. The ReLU function is a popular non-linear activation function in deep learning. We select the Sigmoid function for the output layer as our activation function. Equation (4.6) represents the ReLU function, while equation (4.7) represents the Sigmoid function.

$$f(x) = max(0, x) \tag{4.6}$$

$$f(x) = \frac{1}{1 + e^{-x}} \tag{4.7}$$

Loss function

Loss or error function is a method of determining how effectively a specific algorithm models the provided data. If the predicted outcome differs considerably from the actual output, the loss function will return a large value. Gradually, with the aid of some optimization function, the loss function learns to minimize prediction error. The main aim of the neural network (NN) is to minimize the loss function. We have selected "binary cross-entropy" as a loss function because our proposed model is designed to solve a binary classification

problem. Thus, because we have applied "binary cross-entropy" as a loss function, the outcome value is passed via a sigmoid activation function, with the output range being (0–1).

Optimizer

We have experimented with different optimizers such as Adaptive Moment Estimation (Adam), Stochastic Gradient Descent (SGD), and RMSprop. The best accuracy was obtained when the Adam optimizer was used with a learning rate of 0.001. In our experiment, we were particularly interested in evaluating the model's behaviour when different optimizers were used. Figures 4.27 and 4.28 present the accuracy and loss of our model when the different optimizers have applied to the Google trace. The Adam optimizer has achieved a lower test loss rate than other optimizers, as well as Adam and RMSprop optimizers have the highest accuracy rate. However, we selected Adam to be the optimal optimizer for the proposed model because it has less test loss rate than RMSprop. Additionally, we experiment with various learning rates for the Adam optimizer in order to determine the optimal rate for achieving the highest accuracy. Figures 4.29 and 4.30 present the accuracy and loss of our model when the different learning rates have applied to the Google trace.

Training model

We split the utilized traces into 75% for training and 25% for testing sets. Our proposed model has trained many times using different epochs 10, 30, 60, 100, and 150. An epoch

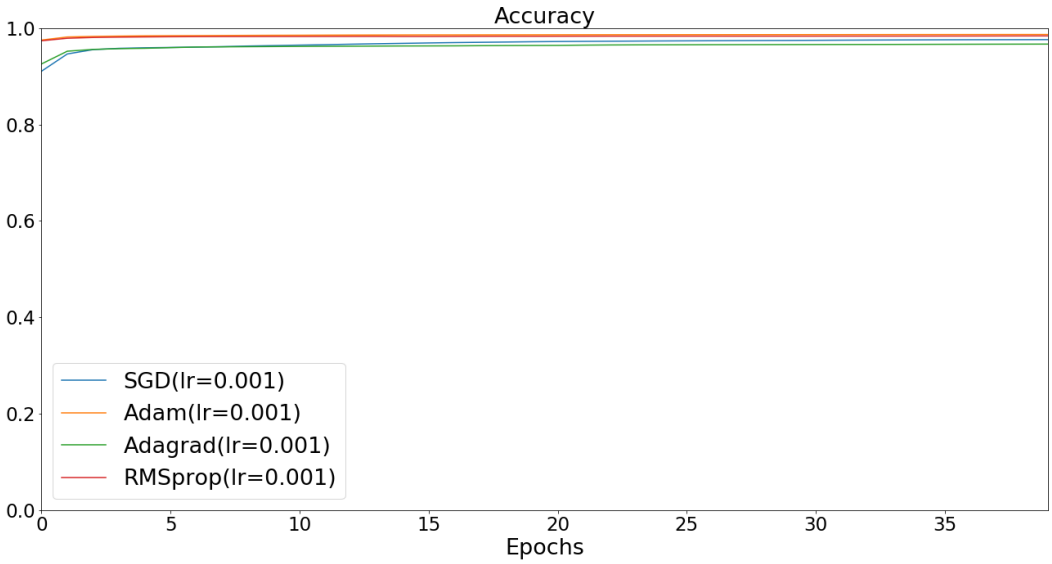


Figure 4.27: The model accuracy rate of applying different optimizers

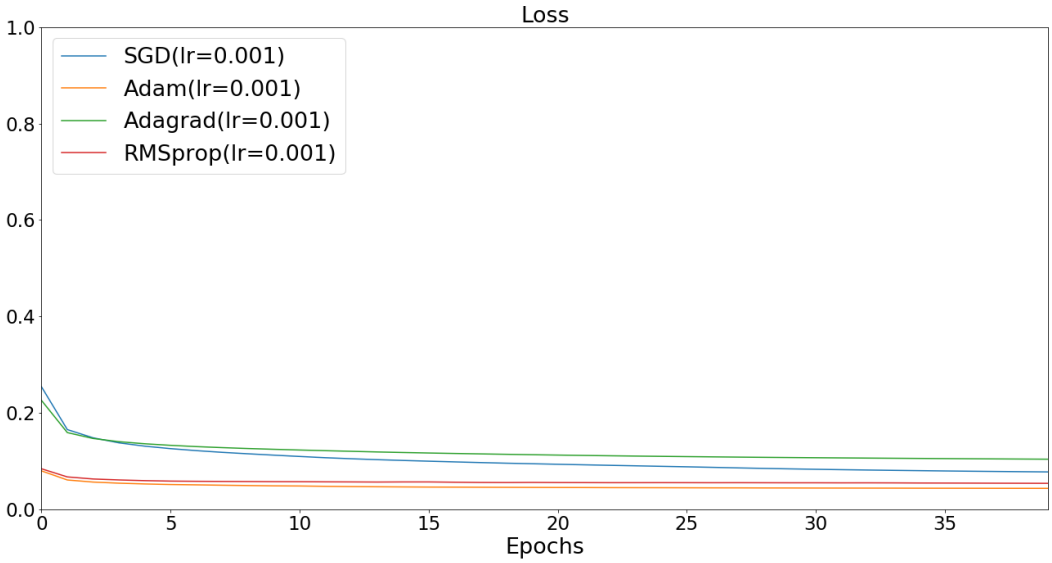


Figure 4.28: The model loss rate of applying different optimizers

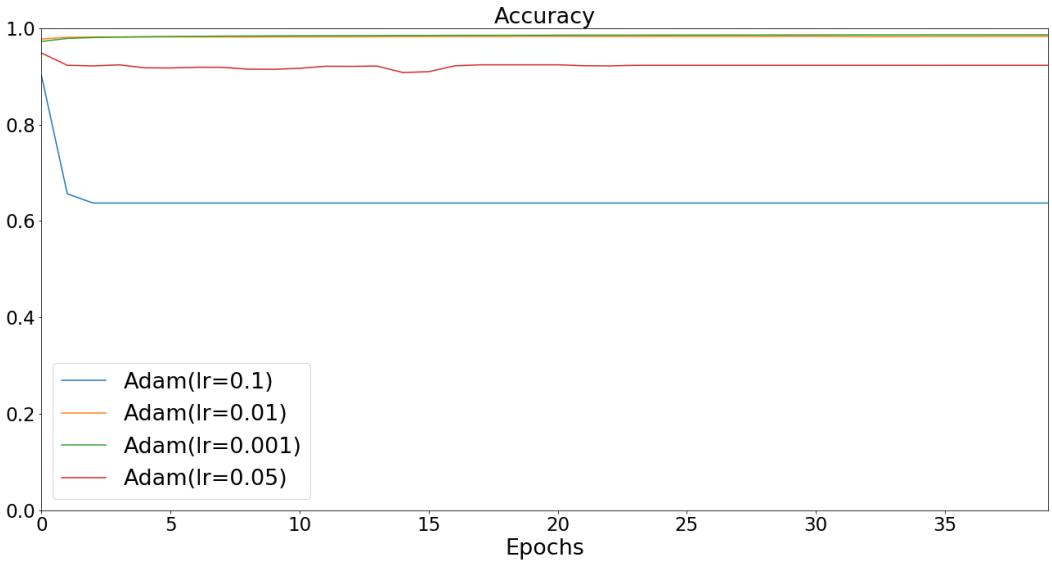


Figure 4.29: The model accuracy rate history of applying different learning rates

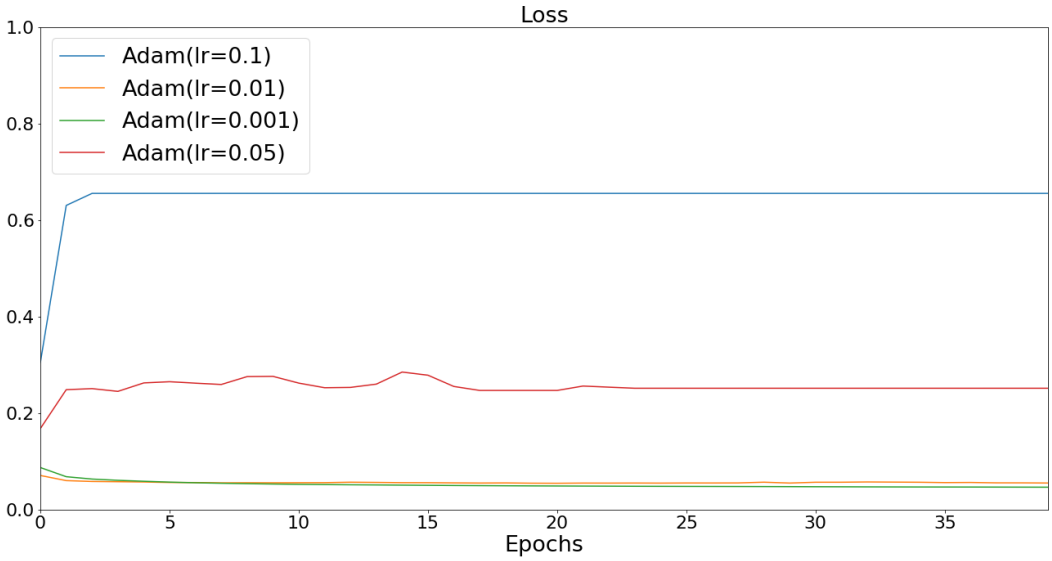


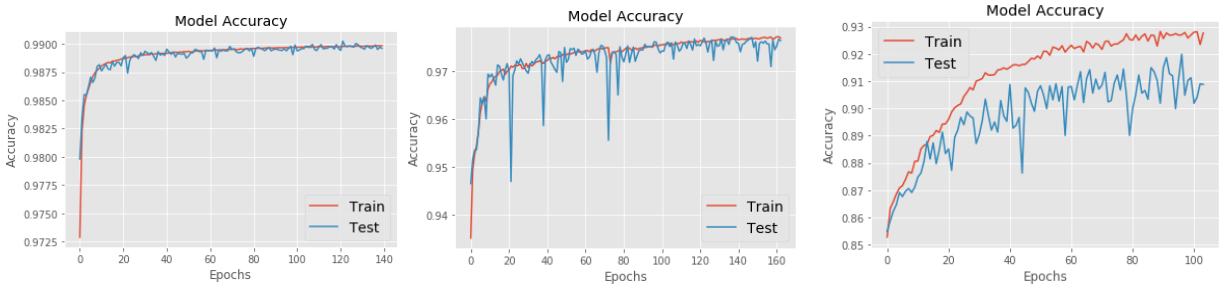
Figure 4.30: The model loss rate history of applying different learning rates

is a single pass through the entire training set, followed by testing the verification set. We find that the required number of epochs is different among all traces. Thus, we have applied the early-stopping technique. We have applied the early stopping technique, which is an important technique that helps the model stop early to avoid over-fitting. The early stopping technique has been designed based on the loss function so that whenever the loss function stops changing, the loss is minimized, and the model has achieved the best accuracy by reaching the minimum. The advantage of applying the early stopping technique is to save computing power by using as few iterations as possible. Moreover, the early stopping technique prevents over-fitting.

We utilize batch size since the dataset is big and so cannot fit all observations simultaneously. This technique divides the utilized trace into small batches equal to the selected batch size. Only this number of observations is loaded and processed into the machine memory. When a batch is completed, the system flushes it from memory and begins processing the next batch.

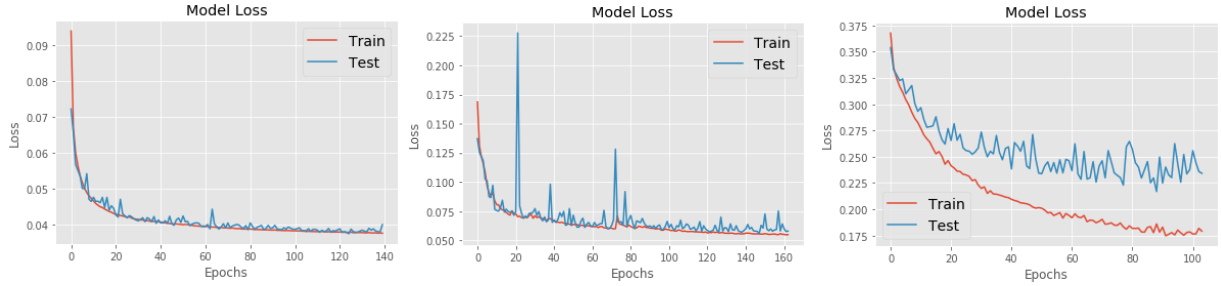
4.4.3 Evaluation Results

In order to ensure that our failure prediction model is general, we apply an ANN deep learning approach to three different traces. We examined many classification models in a previous experiment and then picked the best one to perform as a failure prediction model. We discover that the DTs and RF achieve the highest levels of accuracy.



(a) Model accuracy for Google trace (b) Model accuracy for Mustang trace (c) Model accuracy for Trinity trace

Figure 4.31: The model accuracy for different three traces



(a) Model loss for Google trace (b) Model loss for Mustang trace (c) Model loss for Trinity trace

Figure 4.32: The model loss rate for different three traces

The accuracy and loss graphs for training and testing operations on various traces are shown in Figures 4.31 and 4.32. The line plots for both accuracy and loss show satisfying convergence behaviour for Google trace. The model is well configured, given no sign of under or over-fitting. The model has a high degree of accuracy while the loss score on the training set is fast decreasing, indicating that the network is efficiently learning to classify failed and successful tasks. However, once we apply the proposed model to the Trinity trace, the accuracy decreases compared to other traces. The drop in accuracy is due to the model lacking sufficient observations for training in the Trinity trace. We used a callback mechanism to save the weights when validation loss is minimal. We also use early stops to

Table 4.8: Classification precision, recall, F1-score, training and testing time (in second) achieved through the ANN, DTs, and RF (in percentage)

	Artificial Neural Network					Decision Tree Classifier					Random Forest Classifier				
	Prec.	**Rec.**	**F1-score**	**Train.(t)**	**Test. (t)**	**Prec.**	**Rec.**	**F1.score**	**Train.(t)**	**Test.(t)**	**Prec.**	**Rec.**	**F1.score**	**Train.(t)**	**Test.(t)**
Google (29 days)	99%	99%	99%	11631.2	451.8	99%	99%	99%	542.18	3.39	99%	99%	99%	2812	36.43
Google (7 days)	99%	92%	95%	3244.8	46.4	93%	94%	94%	34.63	0.24	97%	93%	95%	133.82	2.37
Mustang	92%	85%	89%	4686.5	16.8	94%	94%	93%	1.2	0.09	95%	94%	95%	10.3	0.56
Trinity	87%	62%	72%	75.83	0.34	88%	85%	87%	0.07	0.008	89%	85%	89%	0.38	0.02

avoid over-fitting of training data, which can occur when the number of epochs is increased.

Table 4.8 shows the evaluation results of the implementation when we apply three different classifiers to the utilized workload traces. When we apply the classifiers to the Google trace collected in 29 days, all classification models have achieved the highest accuracy with an average of 99% for precision, recall and F1-score. In order to increase the accuracy of DTs and RF classifiers, we applied three different feature selection techniques to all utilized traces. The RFE has achieved the best accuracy for Google trace. As a result, incorporating feature selection techniques into our task failure prediction model has a considerable impact on the model's accuracy.

The highest performance achieved with the Google cluster trace is 99%. In comparison, the accuracy of the Mustang trace is 95%, indicating that when we apply the model to Google trace, all classification algorithms achieve the highest accuracy. As a result, with enough observations, the model can have a high rate of accuracy, precision, recall, and F1-score. The accuracy of the model, on the other hand, reduces when the model includes a small number of observations, such as the Trinity trace.

When applied to medium and small traces, RF and DTs have better accuracy results than an ANN-based algorithm. As a result, we strongly recommend that machine learning

techniques be used to small-sized traces. The model does not achieve a high accuracy rate when the ANN-based deep learning is applied to the Trinity trace, which is considered a small-sized trace.

4.5 IoT Application using Edge-Cloud Architecture

This section describes the deployment and evaluation of our proposed Edge-Cloud architecture using simulation analysis. The purpose of this analysis is to examine the performance of Edge-Cloud integration in order to improve the reliability and availability of cloud-IoT applications. Following that, evaluate reliability, availability, and scalability using metrics such as CPU usage, failure rate, and network delays.

4.5.1 Experimental Setup

We have conducted our experiment with PureEdgeSim simulator [82], which is implemented on the basis of CloudSim Plus [121] to simulate cloud, fog and edge computing environments. PureEdgeSim includes a network model, mobility model, and description of edge device features (such as CPU capacity, battery, mobility, storage, and so on), making it appropriate for Cloud-Edge application scenarios.

Our goal is to simulate the Hajj environment, as most applications in this environment depend on applications for the Internet of Things. These applications require a high

response time, as many sensors and cameras are monitored remotely. As a result, integration between the Cloud and the Edge is important in increasing efficiency, especially in the eHealth application, which requires high response time. During Hajj, response time and latency are crucial for this type of application, such as eHealth. For example, if the pilgrim patient wears a device to measure the heart rate and percentage of oxygen in the blood, the indicators suddenly drop from the normal range. In this case, it is crucial to ensure that data transmission speed is very high and there is no delay.

4.5.2 Simulation Scenario

We designed a scenario to implement three separate applications: e-Health, real-time pilgrim crowd monitoring, and Artificial Intelligence (AI) for IoT applications during Hajj time to evaluate the performance of our proposed solution. Masjid al-Haram, Mina, Arafat and Muzdalifah are the most important religious sites that we are interested in covering in the simulation area. The specified area does not exceed 15 square kilometres.

Edge and Cloud Datacenters

Figure 4.33 describes the edge and cloud resources created in different locations to increase the availability and reliability of cloud-IoT applications and meet business and application requirements.

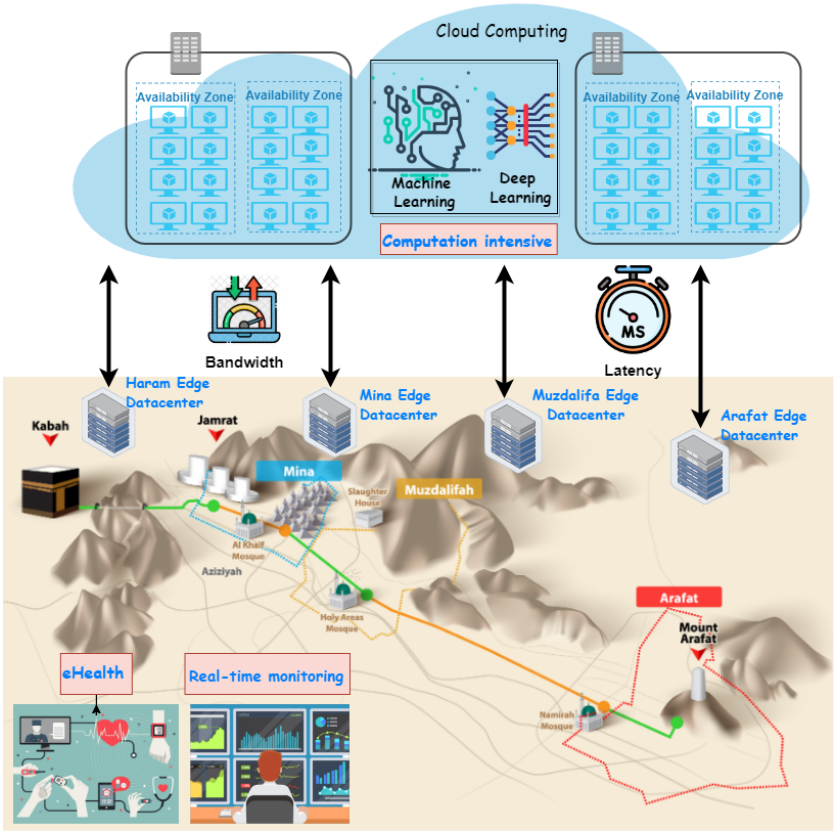


Figure 4.33: Architecture components and computing resources for the Hajj environment

Edge devices types

The eHealth application uses various medical sensors, including body temperature, heart rate, blood pressure and SPO2. Some latency-sensitive applications will offload their tasks to edge computing data centres to minimize execution time. On the other hand, some tasks require high computation and are not latency-sensitive, so these tasks are performed in the cloud computing environment. Based on the requirements of Hajj applications, we carried

Table 4.9: Edge devices types

Edge devices features	Raspberry Pi 4	Smartwatches	Sensor
Mobility	No	Yes	No
Battery-powered	No	Yes	No
Generate tasks	Yes	Yes	Yes
Percentage of devices	20	40	40
Battery-capacity(Wh)	-	19	-
Idle energy consumption (Wh)	2	0.078	0.001
Max energy consumption (Wh)	29	4.3	1.5
CPU (GIPS)	80	30	-
CPU cores	4	-	-
RAM (Gbyte)	4	1	-
Storage(Gbyte)	128	4	-

out the experiment with 700 IoT devices, consisting of 280 smartwatches, 140 Raspberry Pi, 200 medical sensors, and 80 cameras. We have developed our algorithm for scheduling based on the "Trade-off" algorithm used in PureEdgeSim. Table 4.9 shows the features of the various types of Edge devices. All concepts and device features mentioned in Table 4.9 are explained in detail in [82].

Applications Types

Table 4.10 shows the characteristics of smart Hajj applications. We have selected various application types to ensure that the proposed model can be applied with different application requirements. Figure 4.33 shows the architecture components and computing resources for the Hajj environment. The following subsections cover the most important

benefits and functions of each application type.

Table 4.10: Application types

Application types	**eHealth**	**Real-time Monitoring**	**Computation intensive**
Usage percentage (%)	40	40	20
Generation rate (tasks per minute)	10	10	10
Latency sensitivity	Yes	Yes	No
Task length in Giga Instructions (GI)	60	80	200
Task size (Kbytes)	80	800	20000
Results size (Kbytes)	60	100	5000

- eHealth application: Many patients are served remotely through the eHealth application on the busiest days. The eHealth application is designed and implemented to prevent delays in arriving patients' medical data to the health care providers, particularly in emergency and accident situations. Also, the system enables medical users to determine the patient conditions that require immediate transport to the hospital. The primary goal of the eHealth system is to provide pilgrims with a high-quality health care system.

- Real-time crowd monitoring: The real-time monitoring system is intended to keep an eye on the crowding percentage of pilgrims in order to assure everyone's safety.

- Computation intensive: Sensor data is analyzed by ML and DL applications, which extract the essential information that can be utilized to predict task failures using

machine and deep learning approaches. The last type of application requires high computational power.

Table 4.11: The simulation parameters

Parameters	**value**
Simulation duration	30 (min)
Min number of Edge devices	100
Max number of Edge devices	700
Edge devices counter step size	700
Speed of mobile devices	1.4 (meters/seconds)
Edge devices range	20 (meters)
WLAN bandwidth	500 (Mbits/seconds)
WAN bandwidth	500 (Mbits/seconds)
Orchestration algorithm	Trade-off
Architectures	Cloud-Only, Edge-only, Edge-and-Cloud

Simulation parameters

Incoming tasks will be routed to the appropriate resources based on the amount of computing required and the required delay or latency. If the incoming task requires high computation and is not latency-sensitive, it will be transmitted to the cloud data centers. The edge data centers are used if incoming jobs demand low computation and are time-sensitive. Table 4.11 identifies the simulation parameters for this scenario.

4.5.3 Evaluation Results

In this section, our proposed model will be evaluated. We aim to evaluate the performance of the integration of edge and cloud environments in order to increase the reliability and availability of cloud-IoT applications. Therefore, we will focus on the evaluation metrics that show reliability, availability and scalability, such as CPU utilization, failure rate and network delays.

Figure 4.34 shows the number of failed tasks for different architectures. The number of failed tasks has increased sharply when all incoming tasks are submitted to the Cloud only. The delay is the main reason for the failure, as most tasks failed to meet the deadline for meeting the application requirements. For example, the maximum delay for the eHealth application is five seconds. However, it is clear that when we apply the "Edge only" and "Edge-Cloud" architectures, the incoming tasks have a reasonable number of failures compared to the number of failed tasks when using the "Cloud only" architecture. In addition, the "Edge-Cloud" architecture has achieved the best performance because it has been able to control the number of failures, even though the number of devices has increased after each iteration during simulation time. Thus, integrating edge and cloud computing has numerous benefits, including greatly improving the reliability and availability of Edge-Cloud applications and decreasing the time and cost associated with cloud computing. The use of the "Edge-Cloud" architecture also plays an essential role in reducing latency time to meet the new modern IoT applications, such as smart cities and eHealth systems.

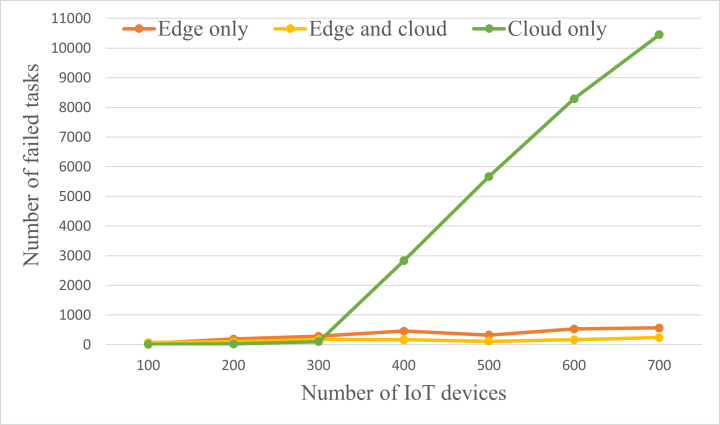


Figure 4.34: Number of failed tasks for different architectures

In addition, the use of the "Edge-Cloud" architecture can control network traffic compared to other architectures. As shown in Figure 4.35, when the "Edge-Cloud" architecture is applied to the system, total network traffic is less than 1,500,000 MB for the use of 700 devices that generated thousands of tasks. The overall network traffic for "Cloud only" and "Edge only" has grown dramatically to roughly 2,500,000 and 3,800,000 Mbytes, respectively. As a result, the "Edge-Cloud" architecture reduces the distance that data from devices must travel, decreasing latency and addressing bandwidth challenges, especially when new technologies such as 5G become available. The "Edge-Cloud" architecture enables network traffic control since the scheduling algorithm distributes incoming tasks between the cloud and the edge based on the required computing power and latency. As a result, the management system monitors the CPU utilization of cloud and edge VMs to schedule incoming tasks to the best VMs with a lower percentage of CPU utilization.

The average VM CPU utilization for the Edge data centers is shown in Figure 4.36. The

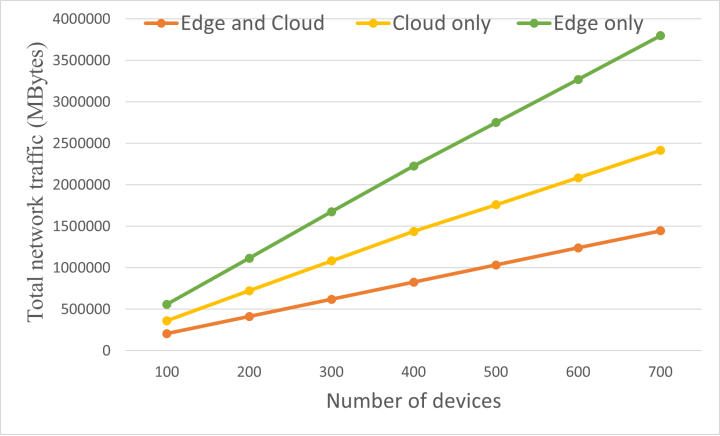


Figure 4.35: Network traffic

"Edge only" architecture has consumed almost 60% of Edge VMs CPU usage to execute a large number of tasks generated from 700 devices. Thus, in this case, all incoming tasks are submitted to the Edge data centers, increasing the workload to serve a large number of devices. On the other hand, when The "Edge-Cloud" architecture is applied to the system, the workload of incoming tasks is distributed between edge and cloud to help the management system control the VMs CPU usage with low consumption to be less than 12%.

In addition, the "Edge-Cloud" architecture has reduced the cost of using cloud services by decreasing the number of tasks executed in the cloud, as shown in Figure 4.37. When using the architecture of "Cloud only", all generated tasks will be transmitted to the cloud to increase the workload in the cloud environment. As shown in Figure 4.37, more than 90,000 tasks were executed in the cloud, which increased the time and cost of using the cloud. The "Edge-Cloud" architecture has reduced the time and cost associated

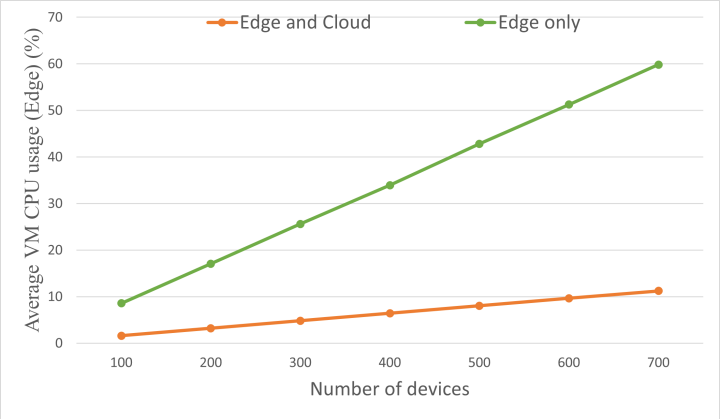


Figure 4.36: Average VM CPU utilization for the edge data centers

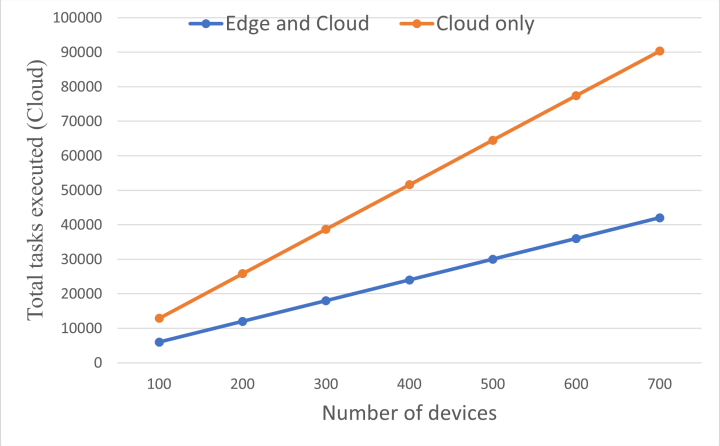


Figure 4.37: Total tasks executed in the edge data centers

with cloud computing, with about 41,000 tasks conducted in the cloud. As a result, this architecture can significantly minimize the cost and time associated with cloud computing. Thus, modern IoT applications require the "Edge-Cloud" architecture to increase reliability, availability, and performance, decrease application latency time, and reduce cloud computing costs.

4.6 Discussion

This section discusses our findings and evaluation results to investigate and analyze failure behaviour for three different traces: Google cluster, Mustang and Trinity. The analysis results show the behaviour of failed and finished jobs to understand the correlation between failed jobs and other cloud application attributes. We have studied job failure as a starting point, which helps develop a new model for failure prediction. The primary goal of our failure prediction model is to predict failed tasks in cloud applications with a high accuracy rate using ML classification algorithms. The proposed model reduces computational time and resource consumption, and increases the efficiency and performance of cloud infrastructure.

As a result of our findings, we notice that failed jobs with long-running times consume considerable cloud resources compared to finished jobs. Another point regarding the job's priority is that high-priority jobs are likely to fail. Therefore, the correlation between job failure and job priority should be taken into account in developing a failure prediction model. Cloud providers can develop new policies for cloud applications based on our failure analysis results to decrease unsuccessful tasks and jobs. For instance, new scheduling algorithms can be applied to increase availability and reliability. Instead of transferring incoming jobs or tasks using the Round-robin technique, the incoming tasks can be transferred based on the required level of availability and computation. This technique will be considered in future work.

We found that Decision Trees (DTs) and Random Forest (RF) provided accurate results after evaluating the performance of various classification algorithms using three cloud traces. However, when it comes to the traces of Trinity and Mustang, the RF classifier surpasses the DTs. Therefore, our selected model is based on the RF method, as it applies to a wide range of traces and provides the most accurate results. El-Sayed et al. [46] developed a RF-based failure prediction model, and their findings show that the presented model can accurately recall up to 94% of all unsuccessful jobs with an accuracy of at least 95%. We have the highest precision, recall, and f1 score compared to previous studies. Our proposed model has achieved the best precision, recall, and f1-score for predicting failed classes, which was 99% for all evaluated metrics. However, due to the small number of observations in Mustang and Trinity, we could not obtain the desired results in the other two traces. Thus, the classifier does not have enough data to distinguish between the two classes. The failure prediction model is trained and tested offline, as it requires a long training time and a high computation power. The workload traces used in this research were collected from the cloud and high-performance computing environments, such as Google cluster traces. Monitoring data indicate cloud characteristics, such as the resources required to perform a task (CPU, memory, disk storage). As a result, the proposed failure prediction model has been designed and implemented for cloud deployment. The task failure prediction model could be adapted to various forms of computing, such as edge, fog or local servers because the observed data is similar.

4.7 Threats to Validity

Because the proposed model for failure analysis and prediction is more focused on the Google cluster, it may not apply to other cloud and IoT infrastructures. This point can be considered a threat to the validity of the study; hence other cloud workload failures should be studied to mitigate this threat. As a result, we have applied the failure analysis and failure prediction models to the other two cloud traces: Mustang and Trinity. We also looked for IoT workload traces; however, there is no publicly accessible failure data from real-world cloud-IoT apps that can be classified as IoT traces. Because there were no public IoT datasets with task characteristics (CPU, memory, and disk space), we evaluated our model on three different datasets of varying sizes to ensure it is applicable in various scenarios (IoT to the cloud or edge). Trinity trace has a small number of observations that can simulate the IoT and edge computing resources. Because Google cloud provider is concerned about privacy, the Google traces are limited and unidentified, which poses a major internal threat to validity. There are a few drawbacks to using Google cluster traces that should be considered:

- The workload traces provide no information about the programs or applications, such as whether they were MapReduce tasks or other types of task applications. Also, there is no information has been provided about the job schedulers or other applications operating on the nodes stored in task characteristics.
- It is unclear who the users are, their workflows, and why they perform these jobs

and tasks. As a result, it is impossible to determine anything about the impact of failures on the entire user experience.

- A task may fail due to poor performance such as lack of resources, or low reliability such as software, hardware, and network failures. The workload traces do not include sufficient evidence to determine task failure reasons.

4.8 Summary

This chapter presents and discusses the different experimental scenarios to verify the performance of the various contributions in this thesis concerning the research objectives. We also compare the results of the proposed failure prediction models with other leading studies and algorithms in this domain. The comparisons confirm our methodology that our contributions in this thesis improve the reliability, availability, and scalability of cloud-IoT applications with higher performance, including precision, recall, and F1-score.

Chapter 5

Conclusion and Future Work

5.1 Conclusion

This thesis presents a framework for proactive fault tolerance in cloud-IoT applications. The main objective of the proposed framework is to design and develop a cloud-IoT framework that provides high reliability and availability for IoT applications using proactive fault tolerance mechanisms such as the failure prediction model. The framework also aims to decrease the number of task failures and minimize the time and cost of using the cloud. This framework includes three major components: integrating local IoT resources with the cloud, designing and implementing the task failure prediction model, and developing Edge-Cloud architecture to support the new modern IoT applications. We have also addressed job failure in cloud computing environments by performing failure analysis on cloud

services, analyzing the behaviour of failed tasks, and correlating failed jobs with required resources such as memory, CPU, and disk space. Firstly, we introduce a framework for extending the local resources of IoT devices, which are very limited in computing, memory, and storage, using cloud computing. Then, the proposed scheduling algorithm has minimized the execution time and computation cost of using the cloud. The results show that extending local resources, including IoT devices, to the cloud improves framework performance and throughput efficiency by 55%. Although this framework shows impressive results in increasing the availability and scalability of cloud-IoT applications, this framework does not consider fault tolerance techniques. Thus, modern IoT applications require a high level of reliability and availability that can be achieved by studying the job and task failure behaviour and the correlation between failed tasks and requested resources.

Secondly, in order to address the limitations of the initial cloud-IoT integrating framework, we study job failures for cloud applications, analyze the behaviour of failed tasks, and examine the correlation between failed jobs and requested resources such as memory, CPU, and disk space. Then, we introduce a proactive fault-tolerance framework that can accurately predict incoming task event types (fail, success) before the management system schedules them. The failure prediction model can decrease resource waste, enhance resource utilization, and increase the efficiency of cloud applications. We have the highest precision, recall, and f1 score compared to previous studies. Based on machine learning, our proposed model has the highest precision, recall and F1-score for predicting failed classes, at 99% for all evaluated metrics. However, the model developed to predict failure

requires a long time to identify the most relevant features in the dataset, and it requires experimenting with several feature selection methods to determine which one is combined with the classifier to achieve the best results.

Thirdly, we have increased the accuracy of our RF-based machine learning by applying different feature selection algorithms such as SelectKBest, Feature Importance, and RFE. In contrast, deep learning algorithms do not require the use of feature selection methods. Deep learning algorithms learn the best features from data without any human guidance. Thus, we present a developed failure prediction model based on an ANN-based deep learning approach to solve the limitation described in our prior framework, designed and implemented based on a machine learning approach. We have also noticed that when combined both RF-based and ANN-based models, the proposed framework can select the optimal model based on the trace size. The IoT revolution has expanded the number of IoT devices and IoT applications deployed in a wide variety of industries. However, due to the limitation of availability and scalability of these applications, the IoT applications face the scaling challenge. Moreover, cloud computing has experienced many challenges due to its rapid expansion. With the sharp increase in the number of IoT devices and data they generate, cloud computing cannot meet quality-of-service criteria such as low latency.

Fourthly, to solve the challenge of availability and scalability of the previous cloud-IoT frameworks, we introduce a highly reliable and available framework based on developing an "Edge-Cloud" architecture to serve the new generation of cloud-IoT applications. We have analyzed cloud-IoT applications to see how well the Edge-Cloud integration performs.

Using evaluation metrics to compare results before and after implementing the Edge-Cloud architecture, we investigated essential elements of IoT applications, including CPU use, failure rate, and network delays. The evaluation results show that failed tasks and CPU usage have decreased in the use of the "Edge-Cloud" architecture. In addition, using "Edge-Cloud" architecture can also control network traffic compared to other architectures.

Our experimental results in all the research described above confirm our proposed solution, as we have obtained results that have outperformed previous frameworks that used similar data for validation. We can recognize limitations at each phase of the process, and then present a new contribution that resolves the challenges identified.

5.2 Future Work

The following issues of research will be investigated in the future:

- **Availability Levels:** We divided the availability levels into three categories to reduce the cloud cost: high, medium, and low. The primary objective of providing different availability levels is to offer cloud consumers a variety of options for availability so that they can have the level of fault tolerance that best suits their needs. Therefore, the availability levels will be considered in future studies, with each level's performance being designed, implemented, and analyzed.

- **LSTM Deep Learning:** We are interested in utilizing Long Short Term Memory

(LSTM) to improve time interval accuracy. LSTM is a deep learning model has developed based on a Recurrent Neural Network (RNN).

- **Online Failure Prediction:** Online failure prediction uses measurements of entire system parameters taken at runtime to calculate the likelihood of a breakdown occurring in the next few seconds or minutes. As a result, for modern cloud-IoT applications, real-time failure prediction is essential. In comparison to traditional dependability methods, online failure prediction is based on runtime monitoring and various models and methodologies that take into account the current state of a system and, in many cases, previous experiences. The outcome of an online failure prediction model can be either a binary decision or a continuous metric determining whether the present situation is more or less likely to fail.

- **Mitigation Policies and Techniques:** Further study needs to consider mitigation techniques to be applied after predicting failed tasks, including migrating the tasks to another availability zone with higher reliability and scalability. Thus, determining why large-scale jobs fail in cloud-IoT applications is essential, followed by proposing and evaluating task failure mitigation techniques.

- **Scheduling Algorithms:** The development of new scheduling algorithms will be a primary focus of future work, as scheduling tasks between cloud and edge node resources continue to be a significant challenge. The scheduling algorithm can decrease the number of failed tasks. The proposed model for the scheduling algorithm will be

compared to new scheduling techniques, such as those provided by the Kubernetes type of orchestrator. The scheduling algorithm will be scalable and able to adapt its decisions on the basis of LSTM prediction outcomes.

References

[1] Mohammad Aazam and Eui-Nam Huh. Fog computing: The cloud-iot/ioe middleware paradigm. *IEEE Potentials*, 35(3):40–44, 2016.

[2] Wiem Abderrahim and Zied Choukair. Brokerage-based dependability integration in cloud computing services. *The Journal of Supercomputing*, 74(7):3359–3387, 2018.

[3] Ivor D Addo, Sheikh I Ahamed, and William C Chu. A reference architecture for high-availability automatic failover between paas cloud providers. In *Trustworthy Systems and their Applications (TSA), 2014 International Conference on*, pages 14–21. IEEE, 2014.

[4] Brian Adler and S Architect. Load balancing in the cloud: Tools, tips and techniques, 2012.

[5] Kazi Main Uddin Ahmed, Manuel Alvarez, and Math HJ Bollen. Characterizing failure and repair time of servers in a hyper-scale data center. In *2020 IEEE PES*

Innovative Smart Grid Technologies Europe (ISGT-Europe), pages 660–664. IEEE, 2020.

[6] Mohammad Al-Zinati, Reem Alrashdan, Basheer Al-Duwairi, and Moayad Aloqaily. A re-organizing biosurveillance framework based on fog and mobile edge computing. *Multimedia Tools and Applications*, 80(11):16805–16825, 2021.

[7] Yanal Alahmad, Tariq Daradkeh, and Anjali Agarwal. Proactive failure-aware task scheduling framework for cloud computing. *IEEE Access*, 2021.

[8] Mansaf Alam, Kashish Ara Shakil, and Shuchi Sethi. Analysis and clustering of workload in google cluster trace based on resource usage. In *2016 IEEE Intl conference on computational science and engineering (CSE) and IEEE Intl conference on embedded and ubiquitous computing (EUC) and 15th Intl symposium on distributed computing and applications for business engineering (DCABES)*, pages 740–747. IEEE, 2016.

[9] Mohammed O Alannsary and Jeff Tian. Measurement and prediction of saas reliability in the cloud. In *Software Quality, Reliability and Security Companion (QRS-C), 2016 IEEE International Conference on*, pages 123–130. IEEE, 2016.

[10] Ameen Alkasem and Hongwei Liu. A survey of fault-tolerance in cloud computing: Concepts and practice. *Research Journal of Applied Sciences, Engineering and Technology*, 11(12):1365–1377, 2015.

[11] Ayman Amin, Alan Colman, and Lars Grunske. An approach to forecasting qos attributes of web services based on arima and garch models. In *2012 IEEE 19th International Conference on Web Services*, pages 74–81. IEEE, 2012.

[12] Zeeshan Amin, Harshpreet Singh, and Nisha Sethi. Review on fault tolerance techniques in cloud computing. *International Journal of Computer Applications*, 116(18), 2015.

[13] George Amvrosiadis, Michael Kuchnik, Jun Woo Park, Chuck Cranor, Gregory R Ganger, Elisabeth Moore, and Nathan DeBardeleben. Atlas: Analysis of traces from los alamos supercomputers. https://www.pdl.cmu.edu/ATLAS/, 2018.

[14] George Amvrosiadis, Michael Kuchnik, Jun Woo Park, Chuck Cranor, Gregory R Ganger, Elisabeth Moore, and Nathan DeBardeleben. The atlas cluster trace repository. 43(4), 2018.

[15] George Amvrosiadis, Jun Woo Park, Gregory R. Ganger, Garth A. Gibson, Elisabeth Baseman, and Nathan DeBardeleben. On the diversity of cluster workloads and its impact on research results. In *2018 USENIX Annual Technical Conference (USENIX ATC 18)*, pages 533–546, Boston, MA, July 2018. USENIX Association.

[16] Strategy Analytics. Strategy analytics: Internet of things now numbers 22 billion devices but where is the revenue. *Retrieved August*, 17:2020, 2019.

[17] Michael Armbrust, Armando Fox, Rean Griffith, Anthony D Joseph, Randy Katz, Andy Konwinski, Gunho Lee, David Patterson, Ariel Rabkin, Ion Stoica, et al. A view of cloud computing. *Communications of the ACM*, 53(4):50–58, 2010.

[18] Vani Vathsala Atluri and Hrushikesha Mohanty. Handling faults in composite web-services. In *Webservices*, pages 99–117. Springer, 2019.

[19] Algirdas Avizienis, Jean-Claude Laprie, Brian Randell, et al. *Fundamental concepts of dependability*. University of Newcastle upon Tyne, Computing Science, 2001.

[20] Saurabh Bagchi, Tarek F Abdelzaher, Ramesh Govindan, Prashant Shenoy, Akanksha Atrey, Pradipta Ghosh, and Ran Xu. New frontiers in iot: Networking, systems, reliability, and security challenges. *IEEE Internet of Things Journal*, 7(12):11330–11346, 2020.

[21] Anju Bala and Inderveer Chana. Intelligent failure prediction models for scientific workflows. *Expert Systems with Applications*, 42(3):980–989, 2015.

[22] Jeff Barr, Attila Narin, and Jinesh Varia. Building fault-tolerant applications on aws. *Amazon Web Services*, pages 1–15, 2011.

[23] Marco Bozzano and Adolfo Villafiorita. *Design and safety assessment of critical systems*. Auerbach Publications, 2010.

[24] Mike Brown and Hersey Cartwright. *VMware vSphere 6.7 Data Center Design Cookbook: Over 100 practical recipes to help you design a powerful virtual infrastructure based on vSphere 6.7*. Packt Publishing Ltd, 2019.

[25] Brad Calder, Ju Wang, Aaron Ogus, Niranjan Nilakantan, Arild Skjolsvold, Sam McKelvie, Yikang Xu, Shashwat Srivastav, Jiesheng Wu, Huseyin Simitci, et al. Windows azure storage: a highly available cloud storage service with strong consistency. In *Proceedings of the Twenty-Third ACM Symposium on Operating Systems Principles*, pages 143–157. ACM, 2011.

[26] Bin Cao, Long Zhang, Yun Li, Daquan Feng, and Wei Cao. Intelligent offloading in multi-access edge computing: A state-of-the-art review and framework. *IEEE Communications Magazine*, 57(3):56–62, 2019.

[27] Keyan Cao, Yefan Liu, Gongjie Meng, and Qimeng Sun. An overview on edge computing research. *IEEE access*, 8:85714–85728, 2020.

[28] Zenon Chaczko, Venkatesh Mahadevan, Shahrzad Aslanzadeh, and Christopher Mcdermid. Availability and load balancing in cloud computing. In *International Conference on Computer and Software Modeling, Singapore*, volume 14, 2011.

[29] Vinay Chamola, Chen-Khong Tham, and GS S Chalapathi. Latency aware mobile task assignment and load balancing for edge cloudlets. In *2017 IEEE International Conference on Pervasive Computing and Communications Workshops (PerCom Workshops)*, pages 587–592. IEEE, 2017.

[30] David Chappell. Introducing windows azure. *Microsoft, Inc, Tech. Rep*, 2009.

[31] Weiwei Chen and Ewa Deelman. Fault tolerant clustering in scientific workflows. In *2012 IEEE Eighth World Congress on Services*, pages 9–16. IEEE, 2012.

[32] 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.

[33] 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.

[34] Xin Chen, Charng-Da Lu, and Karthik Pattabiraman. Failure prediction of jobs in compute clouds: A google cluster case study. In *2014 IEEE International Symposium on Software Reliability Engineering Workshops*, pages 341–346. IEEE, 2014.

[35] Francois Chollet. Keras: Deep learning for humans. https://github.com/fchollet/keras, 2015.

[36] Francois Chollet. *Deep learning with Python*. Simon and Schuster, 2021.

[37] Marshall Copeland, Julian Soh, Anthony Puca, Mike Manning, and David Gollob. Microsoft azure. *New York, NY, USA:: Apress*, 2015.

[38] Domenico Cotroneo, Roberto Pietrantuono, Leonardo Mariani, and Fabrizio Pastore. Investigation of failure causes in workload-driven reliability testing. In *Fourth*

international workshop on Software quality assurance: in conjunction with the 6th ESEC/FSE joint meeting, pages 78–85. ACM, 2007.

[39] Jeffrey Dean and Sanjay Ghemawat. Mapreduce: simplified data processing on large clusters. *Communications of the ACM*, 51(1):107–113, 2008.

[40] Ruilong Deng, Rongxing Lu, Chengzhe Lai, and Tom H Luan. Towards power consumption-delay tradeoff by workload allocation in cloud-fog computing. In *2015 IEEE International Conference on Communications (ICC)*, pages 3909–3914. IEEE, 2015.

[41] Sheng Di, Derrick Kondo, and Franck Cappello. Characterizing cloud applications on a google data center. In *2013 42nd International Conference on Parallel Processing*, pages 468–473. IEEE, 2013.

[42] Sheng Di, Derrick Kondo, and Walfredo Cirne. Characterization and comparison of cloud versus grid workloads. In *2012 IEEE International Conference on Cluster Computing*, pages 230–238. IEEE, 2012.

[43] Catello Di Martino, Daniel Chen, Geetika Goel, Rajeshwari Ganesan, Zbigniew Kalbarczyk, and Ravishankar Iyer. Analysis and diagnosis of sla violations in a production saas cloud. In *Software Reliability Engineering (ISSRE), 2014 IEEE 25th International Symposium on*, pages 178–188. IEEE, 2014.

[44] Catello Di Martino, Santonu Sarkar, Rajeshwari Ganesan, Zbigniew T Kalbarczyk, and Ravishankar K Iyer. Analysis and diagnosis of sla violations in a production saas cloud. *IEEE Transactions on Reliability*, 66(1):54–75, 2017.

[45] Shridhar G Domanal and G Ram Mohana Reddy. Load balancing in cloud environment using a novel hybrid scheduling algorithm. In *Cloud Computing in Emerging Markets (CCEM), 2015 IEEE International Conference on*, pages 37–42. IEEE, 2015.

[46] 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. IEEE, 2017.

[47] Thomas Erl, Ricardo Puttini, and Zaigham Mahmood. *Cloud computing: concepts, technology, & architecture*. Pearson Education, 2013.

[48] Hamid Fadishei, Hamid Saadatfar, and Hossein Deldari. Job failure prediction in grid environment based on workload characteristics. In *Computer Conference, 2009. CSICC 2009. 14th International CSI*, pages 329–334. IEEE, 2009.

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

[50] Peter Garraghan, Paul Townend, and Jie Xu. An analysis of the server characteristics and resource utilization in google cloud. In *2013 IEEE International Conference on Cloud Engineering (IC2E)*, pages 124–131. IEEE, 2013.

[51] Peter Garraghan, Paul Townend, and Jie Xu. An empirical failure-analysis of a large-scale cloud computing environment. In *2014 IEEE 15th International Symposium on High-Assurance Systems Engineering*, pages 113–120. IEEE, 2014.

[52] Steven Martin (General Manager in Microsoft Azure). Windows Azure service disruption from expired certificate. https://azure.microsoft.com/en-ca/blog/windows-azure-service-disruption-from-expired-certificate/, 2013.

[53] Geetika Goel, Arpan Roy, and Rajeshwari Ganesan. Identifying silent failures of saas services using finite state machine based invariant analysis. In *Software Reliability Engineering Workshops (ISSREW), 2013 IEEE International Symposium on*, pages 290–295. IEEE, 2013.

[54] Zhenhuan Gong, Xiaohui Gu, and John Wilkes. Press: Predictive elastic resource scaling for cloud systems. In *2010 International Conference on Network and Service Management*, pages 9–16. Ieee, 2010.

[55] Moin Hasan and Major Singh Goraya. Fault tolerance in cloud computing environment: A systematic survey. *Computers in Industry*, 99:156–172, 2018.

[56] Moin Hasan and Major Singh Goraya. Flexible fault tolerance in cloud through replicated cooperative resource group. *Computer Communications*, 2019.

[57] Tang Hongyan, Li Ying, Wang Long, Gu Jing, and Wu Zhonghai. Predicting misconfiguration-induced unsuccessful executions of jobs in big data system. In *2017 IEEE 41st Annual Computer Software and Applications Conference (COMPSAC)*, volume 1, pages 772–777. IEEE, 2017.

[58] 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.

[59] Mohammad Jassas and Qusay H Mahmoud. Failure analysis and characterization of scheduling jobs in google cluster trace. In *IECON 2018-44th Annual Conference of the IEEE Industrial Electronics Society*, pages 3102–3107. IEEE, 2018.

[60] Mohammad Jassas and Qusay H Mahmoud. Failure characterization and prediction of scheduling jobs in google cluster traces. In *2019 10th IEEE-GCC Conference and Exhibition*. IEEE, 2019.

[61] Mohammad Jassas, Jortin Mathew, Akramul Azim, and Qusay H Mahmoud. A framework for extending resources of embedded systems using the cloud. In *2017 IEEE 30th Canadian Conference on Electrical and Computer Engineering (CCECE)*, pages 1–5. IEEE, 2017.

[62] Mohammad S Jassas and Qusay H Mahmoud. Evaluation of a failure prediction model for large scale cloud applications. In *Canadian Conference on Artificial Intelligence*, pages 321–327. Springer, 2020.

[63] Mohammad S Jassas and Qusay H Mahmoud. A failure prediction model for large scale cloud applications using deep learning. In *2021 IEEE International Systems Conference (SysCon)*, pages 1–8. IEEE, 2021.

[64] Ravi Jhawar, Vincenzo Piuri, and Marco Santambrogio. A comprehensive conceptual system-level approach to fault tolerance in cloud computing. In *Systems Conference (SysCon), 2012 IEEE International*, pages 1–5. IEEE, 2012.

[65] Ravi Jhawar, Vincenzo Piuri, and Marco Santambrogio. Fault tolerance management in cloud computing: A system-level perspective. *IEEE Systems Journal*, 7(2):288–297, 2013.

[66] Apache JMeter. Apache software foundation. https://jmeter.apache.org/, 2010. Accessed: 2022-02-20.

[67] Chaitanya Kallepalli and Jeff Tian. Measuring and modeling usage and reliability for statistical web testing. *IEEE transactions on software engineering*, 27(11):1023–1036, 2001.

[68] Arijit Khan, Xifeng Yan, Shu Tao, and Nikos Anerousis. Workload characterization and prediction in the cloud: A multiple time series approach. In *2012 IEEE Network Operations and Management Symposium*, pages 1287–1294. IEEE, 2012.

[69] John Knight. *Fundamentals of dependable computing for software engineers*. Chapman and Hall/CRC, 2012.

[70] Aniruddh Kumar, Suman Pandey, and Ved Prakash. A survey: Load balancing algorithm in cloud computing. *Available at SSRN 3350328*, 2019.

[71] Kevin Lee, David Murray, Danny Hughes, and Wouter Joosen. Extending sensor networks into the cloud using amazon web services. In *Networked Embedded Systems for Enterprise Applications (NESEA), 2010 IEEE International Conference on*, pages 1–7. IEEE, 2010.

[72] Bijun Li and Rüdiger Kapitza. Bft-dep: automatic deployment of byzantine fault-tolerant services in paas cloud. In *Distributed Applications and Interoperable Systems*, pages 109–114. Springer, 2016.

[73] Yinglung Liang, Yanyong Zhang, Anand Sivasubramaniam, Morris Jette, and Ramendra Sahoo. Bluegene/l failure analysis and prediction models. In *Dependable Systems and Networks, 2006. DSN 2006. International Conference on*, pages 425–434. IEEE, 2006.

[74] Chunhong Liu, Jingjing Han, Yanlei Shang, Chuanchang Liu, Bo Cheng, and Junliang Chen. Predicting of job failure in compute cloud based on online extreme learning machine: a comparative study. *IEEE Access*, 5:9359–9368, 2017.

[75] Chunhong Liu, Jingjing Han, Yanlei Shang, Chuanchang Liu, Bo Cheng, and Junliang Chen. Predicting of Job Failure in Compute Cloud Based on Online Extreme Learning Machine: A Comparative Study. *IEEE Access*, 5:9359–9368, 2017.

[76] Zitao Liu and Sangyeun Cho. Characterizing machines and workloads on a google cluster. In *2012 41st International Conference on Parallel Processing Workshops*, pages 397–403. IEEE, 2012.

[77] Zitao Liu and Sangyeun Cho. Characterizing machines and workloads on a google cluster. In *2012 41st International Conference on Parallel Processing Workshops*, pages 397–403. IEEE, 2012.

[78] Chengzhi Lu, Kejiang Ye, Guoyao Xu, Cheng-Zhong Xu, and Tongxin Bai. Imbalance in the cloud: An analysis on alibaba cluster trace. In *2017 IEEE International Conference on Big Data (Big Data)*, pages 2884–2892. IEEE, 2017.

[79] Akash Malik and Hari Om. Cloud computing and internet of things integration: Architecture, applications, issues, and challenges. In *Sustainable Cloud and Energy Services*, pages 1–24. Springer, 2018.

[80] Wentao Mao, Ling He, Yunju Yan, and Jinwan Wang. Online sequential prediction of bearings imbalanced fault diagnosis by extreme learning machine. *Mechanical Systems and Signal Processing*, 83:450–473, 2017.

[81] Paul Marshall, Kate Keahey, and Tim Freeman. Elastic site: Using clouds to elastically extend site resources. In *Cluster, Cloud and Grid Computing (CCGrid), 2010 10th IEEE/ACM International Conference on*, pages 43–52. IEEE, 2010.

[82] Charafeddine Mechalikh, Hajer Taktak, and Faouzi Moussa. Pureedgesim: A simulation toolkit for performance evaluation of cloud, fog, and pure edge computing environments. In *2019 International Conference on High Performance Computing & Simulation (HPCS)*, pages 700–707. IEEE, 2019.

[83] Charafeddine Mechalikh, Hajer Taktak, and Faouzi Moussa. Pureedgesim: A simulation framework for performance evaluation of cloud, edge and mist computing environments. *Computer Science and Information Systems*, 18(1):43–66, 2021.

[84] Peter Mell, Tim Grance, et al. The nist definition of cloud computing. 2011.

[85] Mohammad Reza Mesbahi, Amir Masoud Rahmani, and Mehdi Hosseinzadeh. Dependability analysis for characterizing google cluster reliability. *International Journal of Communication Systems*, 2019.

[86] Anjali D Meshram, AS Sambare, and SD Zade. Fault tolerance model for reliable cloud computing. *International Journal on Recent and Innovation Trends in Computing and Communication*, 1(7):600–603, 2013.

[87] Sambit Kumar Mishra, Bibhudatta Sahoo, and Priti Paramita Parida. Load balancing in cloud computing: a big picture. *Journal of King Saud University-Computer and Information Sciences*, 2018.

[88] Fabio Jorge Almeida Morais, Francisco Vilar Brasileiro, Raquel Vigolvino Lopes, Ricardo Araújo Santos, Wade Satterfield, and Leandro Rosa. Autoflex: Service agnostic auto-scaling framework for iaas deployment models. In *2013 13th IEEE/ACM International Symposium on Cluster, Cloud, and Grid Computing*, pages 42–49. IEEE, 2013.

[89] Abderrahmen Mtibaa, Afnan Fahim, Khaled A Harras, and Mostafa H Ammar. Towards resource sharing in mobile device clouds: Power balancing across mobile devices. *ACM SIGCOMM Computer Communication Review*, 43(4):51–56, 2013.

[90] Thaha Muhammed, Rashid Mehmood, Aiiad Albeshri, and Iyad Katib. Ubehealth: a personalized ubiquitous cloud and edge-enabled networked healthcare system for smart cities. *IEEE Access*, 6:32258–32285, 2018.

[91] Arijit Mukherjee, Himadri Sekhar Paul, Swarnava Dey, and Ansuman Banerjee. Angels for distributed analytics in iot. In *2014 IEEE World Forum on Internet of Things (WF-IoT)*, pages 565–570. IEEE, 2014.

[92] Mukosi Abraham Mukwevho and Turgay Celik. Toward a smart cloud: A review of fault-tolerance methods in cloud systems. *IEEE Transactions on Services Computing*, 2018.

[93] V Muthumanikandan, C Valliyammai, and B Swarna Deepa. Switch failure detection in software-defined networks. In *Advances in Big Data and Cloud Computing*, pages 155–162. Springer, 2019.

[94] Mina Nabi, Ferhat Khendek, and Maria Toeroe. Upgrade of the iaas cloud: Issues and potential solutions in the context of high-availability. In *Software Reliability Engineering Workshops (ISSREW), 2015 IEEE International Symposium on*, pages 21–24. IEEE, 2015.

[95] P Padmakumari and A Umamakeswari. Task failure prediction using combine bagging ensemble (cbe) classification in cloud workflow. *Wireless Personal Communications*, 107(1):23–40, 2019.

[96] Xinghao Pan, Jiaqi Tan, Soila Kavulya, Rajeev Gandhi, and Priya Narasimhan. Ganesha: Blackbox diagnosis of mapreduce systems. *ACM SIGMETRICS Performance Evaluation Review*, 37(3):8–13, 2010.

[97] Fabian Pedregosa, Gaël Varoquaux, Alexandre Gramfort, Vincent Michel, Bertrand Thirion, Olivier Grisel, Mathieu Blondel, Peter Prettenhofer, Ron Weiss, Vincent Dubourg, et al. Scikit-learn: Machine learning in python. *Journal of machine learning research*, 12(Oct):2825–2830, 2011.

[98] Xuan-Qui Pham and Eui-Nam Huh. Towards task scheduling in a cloud-fog computing system. In *2016 18th Asia-Pacific network operations and management symposium (APNOMS)*, pages 1–4. IEEE, 2016.

[99] Carlo Puliafito, Enzo Mingozzi, Francesco Longo, Antonio Puliafito, and Omer Rana. Fog computing for the internet of things: A survey. *ACM Transactions on Internet Technology (TOIT)*, 19(2):1–41, 2019.

[100] Mazedur Rahman, Samira Iqbal, and Jerry Gao. Load balancer as a service in cloud computing. In *2014 IEEE 8th international symposium on service oriented system engineering*, pages 204–211. IEEE, 2014.

[101] Martin Randles, David Lamb, and A Taleb-Bendiab. A comparative study into distributed load balancing algorithms for cloud computing. In *Advanced Information Networking and Applications Workshops (WAINA), 2010 IEEE 24th International Conference on*, pages 551–556. IEEE, 2010.

[102] Partha Pratim Ray, Dinesh Dash, and Debashis De. Edge computing for internet of things: A survey, e-healthcare case study and future direction. *Journal of Network and Computer Applications*, 140:1–22, 2019.

[103] Charles Reiss, Alexey Tumanov, Gregory R Ganger, Randy H Katz, and Michael A Kozuch. Heterogeneity and dynamicity of clouds at scale: Google trace analysis. In *Proceedings of the third ACM symposium on cloud computing*, pages 1–13, 2012.

[104] Charles Reiss, Alexey Tumanov, Gregory R Ganger, Randy H Katz, and Michael A Kozuch. Heterogeneity and dynamicity of clouds at scale: Google trace analysis. In *Proceedings of the third ACM symposium on cloud computing*, pages 1–13, 2012.

[105] Charles Reiss, John Wilkes, and Joseph L Hellerstein. Google cluster traces. https://github.com/google/cluster-data, 2010. Accessed: 2022-02-20.

[106] Charles Reiss, John Wilkes, and Joseph L Hellerstein. Google cluster-usage traces: format+ schema. *Google Inc., White Paper*, 1, 2011.

[107] Andrea Rosa, Lydia Y Chen, and Walter Binder. Predicting and mitigating jobs failures in big data clusters. In *2015 15th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing*, pages 221–230. IEEE, 2015.

[108] Andrea Rosa, Lydia Y Chen, and Walter Binder. Predicting and mitigating jobs failures in big data clusters. In *2015 15th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing*, pages 221–230. IEEE, 2015.

[109] Arunava Roy and Hoang Pham. Toward the development of a conventional time series based web error forecasting framework. *Empirical Software Engineering*, 23(2):570–644, 2018.

[110] Li Ruan, Xiangrong Xu, Limin Xiao, Feng Yuan, Yin Li, and Dong Dai. A comparative study of large-scale cluster workload traces via multiview analysis. In *2019 IEEE 21st International Conference on High Performance Computing and Communi-*

cations; IEEE 17th International Conference on Smart City; IEEE 5th International Conference on Data Science and Systems (HPCC/SmartCity/DSS), pages 397–404. IEEE, 2019.

[111] Jorge Salas, Francisco Perez-Sorrosal, Marta Patiño-Martínez, and Ricardo Jiménez-Peris. Ws-replication: a framework for highly available web services. In *Proceedings of the 15th international conference on World Wide Web*, pages 357–366. ACM, 2006.

[112] Felix Salfner, Maren Lenk, and Miroslaw Malek. A survey of online failure prediction methods. *ACM Computing Surveys (CSUR)*, 42(3):1–42, 2010.

[113] Taghrid Samak, Dan Gunter, Monte Goode, Ewa Deelman, Gideon Juve, Fabio Silva, and Karan Vahi. Failure analysis of distributed scientific workflows executing in the cloud. In *2012 8th international conference on network and service management (cnsm) and 2012 workshop on systems virtualiztion management (svm)*, pages 46–54. IEEE, 2012.

[114] Areeg Samir and Nagy Ramadan Darwish. Reusability quality attributes and metrics of saas from perspective of business and provider. *International Journal of Computer Science and Information Security*, 14(3):295, 2016.

[115] Daniele Santoro, Daniel Zozin, Daniele Pizzolli, Francesco De Pellegrini, and Silvio Cretti. Foggy: A platform for workload orchestration in a fog computing environment. In *2017 IEEE International Conference on Cloud Computing Technology and Science (CloudCom)*, pages 231–234. IEEE, 2017.

[116] Mahadev Satyanarayanan. The emergence of edge computing. *Computer*, 50(1):30–39, 2017.

[117] Amazon Web Services. Summary of the Amazon S3 Service Disruption in the Northern Virginia (US-EAST-1) Region. https://aws.amazon.com/message/41926/, 2017.

[118] Muhammad Asim Shahid, Noman Islam, Muhammad Mansoor Alam, MS Mazliham, and Shahrulniza Musa. Towards resilient method: An exhaustive survey of fault tolerance methods in the cloud computing environment. *Computer Science Review*, 40:100398, 2021.

[119] Maryam Sheikh Sofla, Mostafa Haghi Kashani, Ebrahim Mahdipour, and Reza Faghih Mirzaee. Towards effective offloading mechanisms in fog computing. *Multimedia Tools and Applications*, pages 1–46, 2021.

[120] Jyoti Shetty, Rahul Sajjan, and G Shobha. Task resource usage analysis and failure prediction in cloud. In *2019 9th International Conference on Cloud Computing, Data Science & Engineering (Confluence)*, pages 342–348. IEEE, 2019.

[121] Manoel C Silva Filho, Raysa L Oliveira, Claudio C Monteiro, Pedro RM Inácio, and Mário M Freire. Cloudsim plus: A cloud computing simulation framework pursuing software engineering principles for improved modularity, extensibility and correctness. In *2017 IFIP/IEEE Symposium on Integrated Network and Service Management (IM)*, pages 400–406. IEEE, 2017.

[122] Jitendra Singh. Study of response time in cloud computing. *International journal of information engineering and electronic business*, 6(5):36, 2014.

[123] Saurabh Singh, Young-Sik Jeong, and Jong Hyuk Park. A survey on cloud computing security: Issues, threats, and solutions. *Journal of Network and Computer Applications*, 75:200–222, 2016.

[124] Marc Snir, Robert W Wisniewski, Jacob A Abraham, Sarita V Adve, Saurabh Bagchi, Pavan Balaji, Jim Belak, Pradip Bose, Franck Cappello, Bill Carlson, et al. Addressing failures in exascale computing. *The International Journal of High Performance Computing Applications*, 28(2):129–173, 2014.

[125] 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. IEEE, 2015.

[126] Azure status history. Microsoft's March 3 Azure East US outage. https://status.azure.com/en-gb/status/history/, 2020.

[127] Yuanyuan Sun, Lele Xu, Ye Li, Lili Guo, Zhongsong Ma, and Yongming Wang. Utilizing deep architecture networks of vae in software fault prediction. In *2018*

IEEE Intl Conf on Parallel & Distributed Processing with Applications), pages 870–877. IEEE, 2018.

[128] Jeff Tian, Sunita Rudraraju, and Zhao Li. Evaluating web software reliability based on workload and failure data extracted from server logs. *IEEE Transactions on Software Engineering*, 30(11):754–769, 2004.

[129] Wei-Tek Tsai, Qingyang Li, Charles J Colbourn, and Xiaoying Bai. Adaptive fault detection for testing tenant applications in multi-tenancy saas systems. In *Cloud Engineering (IC2E), 2013 IEEE International Conference on*, pages 183–192. IEEE, 2013.

[130] Kashi Venkatesh Vishwanath and Nachiappan Nagappan. Characterizing cloud computing hardware reliability. In *Proceedings of the 1st ACM symposium on Cloud computing*, pages 193–204. ACM, 2010.

[131] Kashi Venkatesh Vishwanath and Nachiappan Nagappan. Characterizing cloud computing hardware reliability. In *Proceedings of the 1st ACM symposium on Cloud computing*, pages 193–204. ACM, 2010.

[132] Wei-Jen Wang, Hung-Lin Huang, Shan-Hao Chuang, Shao-Jui Chen, Chia Hung Kao, and Deron Liang. Virtual machines of high availability using hardware-assisted failure detection. In *Security Technology (ICCST), 2015 International Carnahan Conference on*, pages 1–6. IEEE, 2015.

[133] Zhilong Wang, Min Zhang, Danshi Wang, Chuang Song, Min Liu, Jin Li, Liqi Lou, and Zhuo Liu. Failure prediction using machine learning and time series in optical network. *Optics Express*, 25(16):18553–18565, 2017.

[134] Ashkan Yousefpour, Caleb Fung, Tam Nguyen, Krishna Kadiyala, Fatemeh Jalali, Amirreza Niakanlahiji, Jian Kong, and Jason P Jue. All one needs to know about fog computing and related edge computing paradigms: A complete survey. *Journal of Systems Architecture*, 98:289–330, 2019.

[135] Ashkan Yousefpour, Genya Ishigaki, and Jason P Jue. Fog computing: Towards minimizing delay in the internet of things. In *2017 IEEE international conference on edge computing (EDGE)*, pages 17–24. IEEE, 2017.

[136] Junna Zhang, Ao Zhou, Qibo Sun, Shangguang Wang, and Fangchun Yang. Overview on fault tolerance strategies of composite service in service computing. *Wireless Communications and Mobile Computing*, 2018, 2018.

[137] Qi Zhang, Joseph L Hellerstein, and Raouf Boutaba. Characterizing task usage shapes in google's compute clusters. 2011.

[138] Wenbing Zhao, PM Melliar-Smith, and Louise E Moser. Fault tolerance middleware for cloud computing. In *Cloud Computing (CLOUD), 2010 IEEE 3rd International Conference on*, pages 67–74. IEEE, 2010.

[139] Xiaohui Zhao, Liqiang Zhao, and Kai Liang. An energy consumption oriented offloading algorithm for fog computing. In *International Conference on Heterogeneous Networking for Quality, Reliability, Security and Robustness*, pages 293–301. Springer, 2016.

[140] 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.