A Machine Learning Framework for Optimising File Distribution Across Multiple Cloud Storage Services

ABDULLAH F. ALGARNI

Ph.D
University of York
Computer Science

February 2017

Abstract

Storing data using a single cloud storage service may lead to several potential problems for the data owner. Such issues include service continuity, availability, performance, security, and the risk of vendor lock-in. A promising solution is to distribute the data across multiple cloud storage services , similarly to the manner in which data are distributed across multiple physical disk drives to achieve fault tolerance and to improve performance . However, the distinguishing characteristics of different cloud providers, in term of pricing schemes and service performance, make optimising the cost and performance across many cloud storage services at once a challenge. This research proposes a framework for automatically tuning the data distribution policies across multiple cloud storage services from the client side, based on file access patterns. The aim of this work is to explore the optimisation of both the average cost per gigabyte and the average service performance (mainly latency time) on multiple cloud storage services . To achieve these aims, two machine learning algorithms were used: (1) supervised learning to predict file access patterns; (2) reinforcement learning to learn the ideal file distribution parameters.

File distribution over several cloud storage services . The framework was tested in a cloud storage services emulator, which emulated a real multiple-cloud storage services setting in terms of service performance and cost. In addition, the framework was tested in various settings of several cloud storage services. The results of testing the framework showed that the multiple cloud approach achieved an improvement of about 42% for cost and 76% for performance. These findings indicate that storing data in multiple clouds is a superior approach, compared with the commonly used uniform file distribution and compared with a heuristic distribution method.

For all those who are most important to me and for whom I did not have enough time while doing this research, but without whom it would never have been possible.
For my parents; for my wife; for my son.

Contents

Abstract 1
List of Tables 8
List of Figures 12
Acknowledgements 13
Declaration 15

1 Introduction and Motivation **17**
1.1 Introduction 17
1.2 Thesis Structure 21

2 Cloud Computing: Background and Review of the Field **25**
2.1 Definitions of Cloud Computing 25
2.2 Business Drivers and Cloud Benefits 31
2.3 Risks and Challenges 32
2.4 Cloud Storage Services 33
2.5 Data Storage Strategies 35
2.6 Multiple Cloud Storage Solutions 37
2.7 Summary 40

3 Machine Learning: Background **43**
3.1 Definitions of Machine Learning 44
3.2 Machine Learning Components 46
3.2.1 Environment and Data Representation 47
3.2.2 Hypothesis Representation 50
3.2.2.1 Linear Model 52
3.2.2.2 Artificial Neural Network Model 53
3.2.3 Optimisation Algorithms 56
3.2.3.1 Regression Analysis 57
3.2.3.2 Back Propagation 59

3.3 Learning Paradigms . . . 60
3.3.1 Unsupervised Learning . . . 61
3.3.2 Supervised Learning . . . 62
3.3.3 Reinforcement Learning . . . 63
3.3.3.1 State Space Dimensions . . . 66
3.3.3.2 Action Space Dimensions . . . 66
3.3.3.3 Reward Properties . . . 67
3.3.3.4 Building Action Selection Policy . . . 68
3.4 RL with Artificial Neural Network: a Survey . . . 73
3.4.1 TD-Gammon . . . 74
3.4.2 Neural Fitted Q Iteration (NFQ) . . . 75
3.4.3 Deep Reinforcement Learning . . . 75
3.5 Summary . . . 76

4 Machine Learning Applications in Cloud Computing 79
4.1 Resource Allocation Management . . . 80
4.2 Energy Efficiency . . . 83
4.3 Summary . . . 84

5 System Architecture and Emulator 87
5.1 OFDAMCSS Framework Architecture . . . 87
5.2 Cloud Storage Emulator . . . 89
5.3 Summary . . . 91

6 File Access Pattern Prediction 93
6.1 Prediction of File Access Pattern: A Review . . . 93
6.2 Trace History Files: Collections and Structure . . . 94
6.3 Synthetic Data Generator . . . 100
6.4 Access Pattern Predictive Model . . . 103
6.5 Prediction Model Evaluation . . . 104
6.6 APPM System Overheads . . . 104
6.7 Summary . . . 106

7 Intelligent Framework for Optimising File Distribution Across Multiple Cloud Storage Services 107
7.1 Characteristics of the Reinforcement Learning System . . . 108
7.2 Artificial Neural Network with Reinforcement Learning . . . 109
7.3 Reinforcement Learning Model . . . 111
7.4 Summary . . . 117

8 System Evaluation 119
8.1 Experiment Settings . . . 119
8.2 Evaluation Methods . . . 122
8.2.1 Mirroring Approach . . . 122

8.2.2 Principle of Standard RAID Distribution Approach 123
8.2.3 Heuristic Distribution Approach 123
8.3 OFDAMCSS: Experiments and Analysis 125
8.3.1 Overall result 125
8.3.2 Analysis of Results 129
8.3.3 Developmental of the System 133
8.3.4 Impact of Access Pattern on Distribution Decisions 135
8.3.5 Parameter Effects in the OFDAMCSS Framework 144
8.4 OFDAMCSS Framework Overheads 146
8.5 Summary 148

9 Conclusion and Future Work **153**
9.1 Summary of the Thesis 153
9.2 Novel Contributions of this Work 155
9.3 Limitations and Future Work 156

References **161**

List of Symbols **175**

Acronyms **177**

Glossary **179**

Index **183**

List of Tables

2.1 Comparison of online storage types 34
2.2 Common RAID levels with strengths and weaknesses 35
2.3 Comparison among distribution frameworks : Key: S. cost, storage cost; N. cost, network cost (i.e. transaction cost); O. cost, operation cost (read, write, and delete). 39

3.1 Comparison among approaches to implementing artificial neural networks with reinforcement learning 76

5.1 Performance range of cloud storage services (measurements by cloudharmony.com) 91

6.1 Attributes of the synthetic datasets 101
6.2 Attribute variables from which the generator could select values 102
6.3 APPM, measurement evaluation for WS-dataset 1 105
6.4 APPM, measurement evaluation for WS-dataset 2 105
6.5 APPM, measurement evaluation for WS-dataset 3 105
6.6 APPM, measurement evaluation for R-dataset 1 105
6.7 APPM, measurement evaluation for R-dataset 2 105

8.1 Reinforcement learning parameter settings 120
8.2 Artificial neural network parameters settings 121
8.4 Summary statistics for uploading all datasets into all cloud storage using the SRD method. Latency_r, latency of reading; latency_w, latency of writing. 123
8.5 Summary statistics for uploading all datasets into all cloud storage using the heuristic method. Latency_r, latency of reading; latency_w, latency of writing 125

8.6 Summary statistics for uploading all datasets into all cloud storage using the proposed OFDAMCSS framework. Latency_r, latency of reading; latency_w, latency of writing 126
8.8 Test of parameters settings for generative parameters of OFDAMCSS . 135
8.7 A full statistical analysis of distributing all data sets using SRD, Heuristic code and OFDAMCSS frameworks. Where latency_r is latency of read and latency_w is latency of write 138
8.9 Percentage change in the storage cost for the writing group 139
8.10 Percentage change in network cost for each file 139
8.11 Percentage change in the storage cost for the reading group 140
8.12 Percentage change in the network cost for the reading group 140
8.13 Percentage change in the storage used for the writing group 141
8.14 Percentage change in the network used for the writing group 141
8.15 Percentage change in the storage used for the reading group 142
8.16 Percentage change in the network used for the reading group 142
8.17 Percentage change in the latency time for the writing group 143
8.18 Percentage change in the latency time for the reading group 143

List of Figures

1.1 Structure of the thesis 23

2.1 A mind-map for cloud computing characteristics, benefits and risks (from NIST definitions). 26

2.2 High-level architecture of cloud computing layers. All services are housed in a physical data centre and are typically accessed through Virtual Machine (VM) technology. 29

3.1 Block representation of process and components of machine learning . 47

3.2 Table matrix to represent data in machine learning 48

3.3 Representation Type 50

3.4 Block diagram for linear model 53

3.5 A fully connected feed-forward neural network structure consists of three layers of nodes: (1) input layer denoted as $x_i, i \in \mathbb{N}$, (2) hidden layer denoted as $h_j, j \in \mathbb{N}$ and (3) output layer denoted as $y_k, k \in \mathbb{N}$ 55

3.6 Transfer Function 56

3.7 Machine Learning Paradigms 61

3.8 The reinforcement learning paradigm consists of a machine learning system interacting with an environment. At each discrete time t, the machine learning system observes the state of the environment s_t and performs an action a_t in order to transition from its current state to a subsequent state s_{t+1}. It receives a reward $r_(t + 1)$ for the value of that transition. Over time, the machine learning system learns to improve the selection of actions that maximise the cumulative reward 64

3.9 Modelling $Q(s, a)$ with artificial neural network, Left panel: Naive formulation of Q-value, where the network takes state features and an action. Right panel: is more optimised formulation of Q-value, where the network only takes state features and input and produces multiple Q-values equal to a number of actions possible in the given state. 71

3.10 A mind-map of a reinforcement learning framework. Reinforcement learning has four main components that interact with each other to solve sequential decision problems: environment, action space, reward function, and action selection policy. 77

5.1 A high-level view of the Distribution Framework Structure, where APPM estimates how many times each file will be accessed in the future and its expected lifetime; the reinforcement learning system tunes the distribution parameters (i.e., the proportion of each file that will be located in each cloud); the distributor manages distribution, taking distribution policy from the reinforcement learning system ; C_1,C_2, and C_K cloud storage services where $k \in \mathbb{N}$. 88

5.2 Cloud storage emulator: the architecture of classes. The operation class is responsible for writing and reading from the storage. After each operation, the emulator calculates the latency time based on the speed of each cloud provider and the total cost based on each cloud provider's pricing scheme. The builder class is responsible for building the storage services based on the configuration file that contains all storage attributes 90

6.1 Usually in SFD, the user creates a vacation file, fills it, then sends it to the section manager. The manager annotates it and passes it to the department manager to sign. Finally, the file is passed to HR staff to process and save 96

6.2 The payment roll system is used to generate sheets of employees' salaries. Usually, sheets are generated as drafts at the start of a month. They then pass through various processes, administered by users in different departments, before being sent to the bank. 97

6.3 The annual budget system is used to create an annual report on expenses in the past year and expectations of expenses in the coming year 98

6.4 Research department system for generating research reports 100

7.1 Demonstration of how the reinforcement learning system works in the OFDAMCSS framework. At each discrete time t, the learning system receives file access pattern attributes (from APPM) that represent the state features X, which will be passed through an artificial neural network to produce different outputs. Each output will be transformed to an action a_k, which is the proportion of file size that will be located in each cloud. These actions will be given to the Distributor system to perform. The agent then receives rewards from each cloud r_k, which will be used by the TD-error (TD_K) to evaluate each action separately. The back-propagation function (BP) will then be triggered to tune the connection weights (w_{jk}, w_{ij}). 112

8.1 Average total cost and average latency time to send the entire data from all datasets into one cloud storage service 126
8.2 Distributing all files from all datasets across multi-cloud using standard RAID distribution SRD and OFDAMCSS 127
8.3 Distributing all files from all datasets across multi-clouds, using the heuristic approach and OFDAMCSS 128
8.4 Total cost and average latency time for each cloud provider using SRD . 129
8.5 Total cost and average latency time for each cloud provider using the heuristic approach 130
8.6 Total cost and average latency time for each cloud provider using OFDAMCSS framework 131
8.7 The reduction in total cost (in figure (a)) and latency time of reading and writing (in figure (b)) reduces over time , using OFDAMCSS 132
8.8 Change in the total cost (in figure (a)) and latency time of reading and writing (in figure (b)) change when a cloud stores was discontinued, (using OFDAMCSS) 133
8.9 Change in the cost (in figure (a)) and latency time of reading and writing (in figure (b)) change when a new cloud stores was added to the system , using OFDAMCSS layer 134
8.10 Generative features of OFDAMCSS framework by testing the ability of the framework to optimise cost and latency time at the same time were tested for different groups of cloud storage services 136
8.11 Nested meters for three artificial neural network parameters and three learning algorithms for reinforcement learning 145
8.12 Evaluate the impact of changing the number of hidden nodes on the hidden layer 146
8.13 The impact of α on the performance of learning in the OFDAMCSS . . 147
8.14 The impact of β on the performance of learning in the OFDAMCSS . . 148
8.15 The impact of λ on the performance of learning in the OFDAMCSS . . 149
8.16 The impact of γ on the performance of learning in the OFDAMCSS . . 150
8.17 The impact of η on the performance of learning in the OFDAMCSS . . 151
8.18 The impact of ϵ on the performance of learning in the OFDAMCSS . . 152

Acknowledgements

Bismillaah ar-Rahman ar-Raheem, In the name of God, the infinitely Compassionate and Merciful. All Praise and thanks goes to ALLAH, Lord of the Worlds, and may the peace and blessings be on his Prophets and Messenger Muhammad and on his family and all of his Companions.

First and foremost, I would like to acknowledge and thank my father Fayez Hassan Algarni and my mother Zarah Humood Algarni for their love, support, and encouragement. They made tremendous sacrifices to ensure that I had an excellent education and attained my current level of knowledge. For this and much more, I am forever in their debt. It is to them that I dedicate this thesis. I would also like to thank my wife Zainab for standing beside me throughout my studies. She has been my motivation to achieve my goals and improve my knowledge.

I would like to thank the vice President and Managing Director of the Saudi Fund for Development (SFD), H.E. Eng. Youssif Bin Ibrahim Albassam, for his kind support since I started my career in the SFD, and for allowing me the opportunity to study for Masters and PhD degrees. In addition, I am deeply grateful to Mr. Mohammed Alsabti and Mr. Saud Alfantoukh for their tremendous support and encouragement during my career and studies.

I would like to express special appreciation and sincere thanks to my supervisor Dr. Daniel Kudenko for his mentorship, understanding and patience over the past four years, and for allowing me to explore my research interests. I am also grateful to my assessor, Dr. Radu Calinescu, for his challenging questions and thoughtful discussions about my work during our scheduled meetings over the past four years.

I would like to thank Dr. Christopher Gatti from the US (http://www.chrisgatti.com/) for his kind responses to my emails and for his excellent advice. I would also like to thank all my friends and any individuals who have helped me over the period of my study, in all ways large and small.

Finally, I would like to thank all my brothers (Muhammed, Hassan, Yossef, Abdulrahman,and Sultan) and my sisters (Fatimah and Rana) for their love and support.

Declaration

This thesis has not previously been accepted in substance for any degree and is not being concurrently submitted in candidature for any degree other than Doctor of Philosophy of the University of York. This thesis is the result of my own investigations, except where otherwise stated. Other sources are acknowledged by explicit references.

I hereby give consent for my thesis, if accepted, to be made available for photocopying and for inter-library loan, and for the title and summary to be made available to outside organisations.

CHAPTER 1

Introduction and Motivation

1.1 Introduction

Cloud computing is a model for delivering computational resources through the Internet to users in a cost-effective way. Examples of such resources include computer networks, servers, storage, and applications. Interest in cloud computing continues to grow, and it offers many benefits – especially for businesses – compared with building and maintaining resources in-house. One of the main advantages of cloud computing is the scalability and elasticity of resources in response to load increases and decreases. In addition, it is easy to use and configure; users pay only for what they use. In general, cloud providers offer computational resources to users through a service model. The three most popular models are Infrastructure as a Service (IaaS), Platform as a Service (PaaS), and Software as a Service (SaaS), with each model consisting of various services.

This work only considers one service of IaaS service, namely a cloud storage service. Broadly speaking, a cloud storage service allows users to store data using remote storage, which can be accessed through the Internet. It offers many advantages for users, including increased work efficiency and reduced operational costs in the long term. Cloud

storage also allows organisations to improve the management of expanding storage capacity demands with regard to physical storage, which can increase dramatically over time [Linden 2012] [Rebello 2012].

Despite the advantages and benefits of using a cloud storage service, there are some underlying concerns about it. They are summarised in the following points:

- **Cloud performance:** in this thesis, 'cloud performance' refers to network latency time, which is the time a packet (a unit of data made into a single package) takes to travel from leaving the user-side to being completely stored on the cloud provider's servers. The reader should not confuse this term with the general meaning of the term 'latency', which usually refers to the delay that occurs before transfer of data begins, following an instruction for such data transfer [Oxf 2017]. The interested reader can access information elsewhere about the different meanings of 'latency' in various contexts [tec 2017]. Latency time is one of the big concerns to cloud users as it can affect the quality of their business or services. Typically, latency time is influenced by the distance between the locations of client and cloud servers, and by network throughput. Network throughput refers to the actual amount of data that can be transferred from one point to another across a network in timeframe [Solomon et al. 2014].

- **Cloud vendor lock-in:** Each cloud provider has a specific application programming interface (API) requirement. The API allows user applications to integrate and interact with cloud services, but means that cloud users must design and build their applications based on APIs. The lack of standardisation of APIs, and the divergence between API requirements from one cloud provider to another, incur the risk of locking a user's data to a cloud provider [Mu et al. 2012]. Several unified APIs have been designed and implemented to overcome this problem. Examples of unified APIs include Libcloud [Libclouds 2016] and Jcloud [Jclouds 2016]. ¥ These unified APIs are designed to allow users to move from one cloud to another without changing their applications, and enable users to adopt multiple cloud ser-

vices simultaneously. However, migrating from one cloud provider to another is still a challenge in terms of time, cost and effort.

- **Service Continuity:** Cloud services can go out of business with little warning. For example, Nirvanix Cloud Storage services shut down in October 2013 with only two weeks' notice for customers to withdraw their data[Marshall 2013]. Barracuda discontinued Copy Cloud Storage and CudaDrive services in May 2016 [Brinkmann 2016].

- **Service Availability:** Another issue in adopting cloud services is the number of cloud services that have suffered outages in recent years. For instance, Amazon Simple Storage Service (S3) suffered outages three times in 2008, for different reasons[Armbrust et al. 2010]. In 2008, Gmail was unavailable for about 1.5 hours due to a problem in the Contacts System [Armbrust et al. 2010]. More recently, Amazon EC2 and S3 suffered a rare outage in 2015 [Tsidulko 2015] and in 2016 Google Compute Engine cloud service went down for 18 minutes [Bort 2016].

Some researchers have addressed the above issues and suggested using the redundant array of independent disks (RAID) principle. This principle allows for distribution of data across multiple cloud storage services instead of relying on a single service (e.g., Abu-Libdeh et al. [2010] Papaioannou et al. [2012] Bowers et al. [2009]). Typically, RAID has been used in traditional storage to avoid problems with using a single hard disk, by distributing data on multiple hard disks. There are two main objectives in using RAID: (1) to improve the performance of reading from and writing to hard disks; and (2) to provide some level of fault tolerance in case one of the hard disks fails.

However, the case of multiple cloud storage is different. Each cloud service has unique characteristics that affect the connectivity between the client and the cloud service. Furthermore, file access patterns and file size play a fundamental role in the cost of cloud services. More specifically, the cost of cloud storage is based on several factors: (1) number of operations (typically: read, write and delete); (2) network usage (which is impacted by number of use and file size) and storage usage (which is affected by the

lifetime and by the size of files).

Optimising the cost and latency time of diverse cloud services is difficult because of differences in service performance and pricing schemes among cloud storage providers. Additionally, the price scheme and performance of any cloud provider are not stationary. The cost may change automatically based on how many data are stored or transferred through the network. Thus, the optimisation should ideally account for long-term cost and performance, rather than just optimising the current state. Accordingly, it is important to find a dynamic solution that is capable of optimising cost and latency time, and can adapt to changes in the states of the various cloud storage services.

This work demonstrates that reinforcement learning is a suitable technique to deal with these challenges because of its ability to maximise the expected long-term utilities. Furthermore, this work shows how supervised learning can be used to generate predictive models of file access patterns within large enterprise workflows. Unfortunately, no real data for file access log exists which could be used in supervised learning to predict the future. Thus, this work generated a synthetic dataset based on real workflow systems.

The overall contributions of this work are as follows:

- It provides an intelligent framework for optimising the cost and latency time (performance) when using multiple cloud storage services. In addition, the framework is designed to be flexible and easily configurable for varying numbers of cloud storage services, to mitigate the problems of service continuity, availability, and vendor lock-in. The development of this framework combined two machine learning methodologies:

    1. Reinforcement learning to determine the respective percentage of each file that would optimally be located in each cloud storage service, based on file access patterns.
    2. Supervised learning to predict the access patterns for each file.

- 2. It describes a novel method to turn multiple state values into actions. This method is a reinforcement learning approach designed specifically for this work.

It uses an artificial neural network to produce several continuous actions (concurrently) from different independent state values. This contribution can be divided into two aspects:

1. The approach can turn the output of multiple value functions into actions.
2. The approach can handle multiple continuous actions for distribution purposes.

Note : This approach can only work in a distribution domain, such as distributing data across several storage devices.

- It created synthetic datasets for trace log files. These datasets were generated based on an analysis of real workflow systems.
- It predicted numerical values of file access pattern attributes through a supervised learning algorithm. Two pieces of research in the literature predicate classification of file access patterns. To the best of our knowledge, our research is the first to predict (1) a numerical value for a file lifetime; (2) the number of times read access is used; (3) the number of times write access is used.
- It presents 'Hints and Tips' for avoiding many issues related to applying the artificial neural network as function approximator with reinforcement learning.

1.2 Thesis Structure

As shown in Figure 1.1, after the Introduction, the main body of the thesis begins with Chapter 2, which gives an overview of cloud computing. An examination of cloud storage services, which are the core of this work, is included. Chapter 2 also outlines the challenges of adopting cloud storage services, and provides a review of the literature related to the fundamental concepts in this work. Chapter 3 provides an overview of the field of machine learning, as the proposed solution is based on algorithms from that field. Chapter 3 includes a discussion of two subfields in machine learning: supervised learning and reinforcement learning. Chapter 4 provides a survey of work on machine

learning in cloud environments. Details of the architecture and design of the proposed framework are introduced in Chapter 5. The framework consists of two machine learning algorithms (supervised learning and reinforcement learning) that are discussed in Chapters 6 and 7. These chapters describe the methodology used in the experiment to empirically evaluate the framework. Chapter 8 presents the evaluation and a discussion of the results. In addition, the evaluation chapter explores the effect of parameters in reinforcement learning and artificial neural network learning on the performance of the framework. Finally, Chapter 9 concludes the thesis with a discussion of findings, innovations and possible directions for future work.

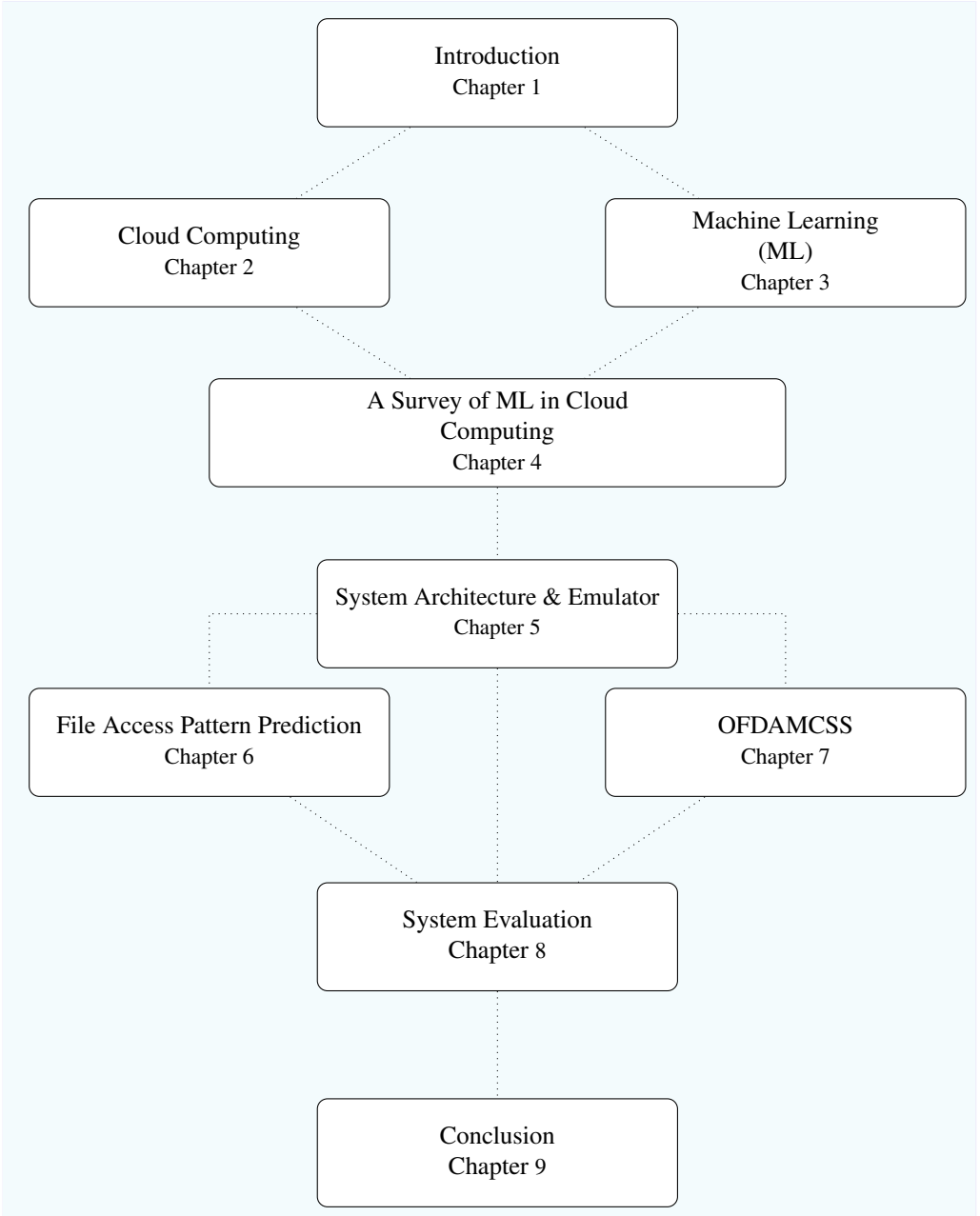


Figure 1.1: Structure of the thesis

CHAPTER 2

Cloud Computing: Background and Review of the Field

This chapter presents an overview of cloud computing including a discussion of relevant definitions and main service models. This research focuses mainly on cloud storage services, and most of this chapter examines that service, including its benefits and challenges. The chapter begins with a discussion of what cloud computing is, and presents the popular definitions of cloud computing – including its characteristics and services. Thereafter, the chapter presents the benefits of adopting cloud services (i.e. business drivers). The risks and concerns that hinder organisations from adopting cloud services are also discussed. Finally, an in-depth description of cloud storage services is provided, as is a review of relevant work related to the fundamental concepts on which this research is based.

2.1 Definitions of Cloud Computing

The term 'cloud computing' emerged in the commercial arena in 2006, as a new business model in the computing world, after Amazon started its Elastic Compute Cloud (EC2) service [Erl et al. 2013]. Since then, the topic has gained momentum and attention in

Figure 2.1: A mind-map for cloud computing characteristics, benefits and risks (from NIST definitions).

both academia and in the world of computing industry. However, around 2006 there was no widely accepted definition of cloud computing [Voas & Zhang 2009].

The need for a standard meaning for 'cloud computing' pushed researchers and organisations to find a comprehensive definition of the term. In 2008, The Gartner Inc. was one of first organisations to publish a definition of cloud computing:

> *A style of computing in which massively scalable IT-related capabilities are provided as a service using Internet technologies to multiple external customers. [Pettey & van der Meulen 2008]*

In 2009, they revised this definition and replaced 'massively scalable' with 'scalable and elastic'. This alteration highlighted the importance of cloud computing's scalability characteristic, in particular the ability to scale both up and down – not simply to a large size [Pettey & Goasduff 2008]. In addition, the new definition included a list of five attributes for cloud computing services:

1. They share a pool of resources
2. They scale up or down as the consumer demands.
3. They can be tracked with usage metrics to enable multiple payment models.
4. They are accessible through application programming interfaces (APIs).
5. 1. They are available online and can be accessed through the Internet .

Forrester Inc. also published a definition of cloud computing in 2008, which was widely accepted by the industry at that time. It was also used by several US Federal Government agencies, including the National Institute of Standards and Technology (NIST). They used it as a resource to create their own definition [Staten 2009] [Erl et al. 2013]. Forrester's definition described cloud computing as :

> *standardized IT capability (services, software, or infrastructure) delivered via Internet technologies in a pay-per-use, self-service way. [Staten et al. 2008]*

Starting in 2009 and continuing over several years, NIST also revised many draft versions of a cloud computing definition, reviewing them with government and industry. The goal was to create a robust, comprehensive cloud computing definition that supported interoperability, portability and security requirements [Mell & Grance 2011]. The

final version of that work was published in 2011. Since then, this definition has been widely used in industry and academia. The definition states as follows:

> *Cloud computing is a model for enabling ubiquitous, 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. [Mell & Grance 2011]*

The NIST lists five essential characteristics which are the key attributes for a service to be considered a cloud service:

1. **On-demand self-service** – the ability to provide computing capabilities automatically, without human intervention.

2. **Broad network access** – the availability of computing capabilities online on the Internet.

3. **Resource pooling** – the cloud provider's resources are pooled across multiple customers using a multi-tenant model.

4. **Rapid elasticity** –the capability of cloud infrastructure that can scale up or down dynamically at any time, to cope with demand peak on a near real-time basis.

5. **Measured service** – the ability to monitor the usage of cloud services by each consumer with usage metrics, to facilitate working as a utility service.

The NIST definition outlines four deployment models for cloud infrastructure. Each model can possess different characteristics to meet different business needs. The deployment models are:

- **Private Cloud** – Private cloud – a cloud infrastructure that can be provided by a single organisation for exclusive use.

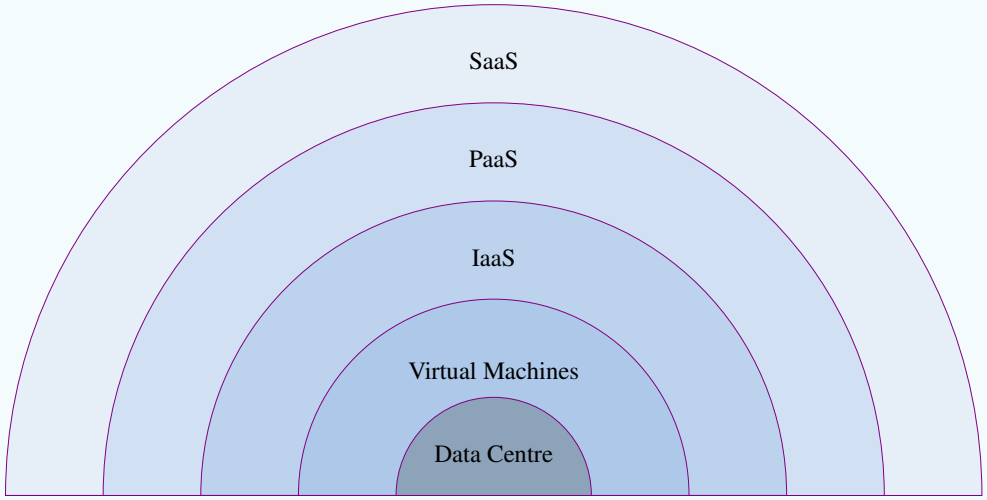


Figure 2.2: High-level architecture of cloud computing layers. All services are housed in a physical data centre and are typically accessed through Virtual Machine (VM) technology.

- **Community cloud** – a cloud infrastructure that can be provided by a group of organisations.
- **Public cloud** – a cloud infrastructure that can be provided by any cloud consumer, either an individual or an organisation.
- **Hybrid cloud** – a combination of one or more the above deployment models

Furthermore, NIST lists three fundamental service models (Figure 2.2), all of which should be designed to serve multiple clients simultaneously:

- **Infrastructure as a Service (IaaS)** – a virtual layer of the cloud's physical infrastructure (i.e. the hardware components) that can be delivered to cloud consumers through the Internet as virtual machines (VMs). Examples of this model's services include servers, networks, operating systems, storage and other raw computing resources. The goal of this model is to provide cloud consumers with a high level of control over configuration and utilisation. Hence, the computing resources that

are delivered to the consumer in this model are generally not preconfigured. [Erl et al. 2013].

- **Platform as a Service (PaaS)** – a configured (i.e. ready to use) environment built upon the IaaS model to provide a platform of computing resources with low control over the underlying resources that host the platform. This model allows consumers to develop and host their applications or to store their data in the cloud infrastructure without needing worry about infrastructure settings and maintenance. Providers of this model include Google App Engine and Amazon AWS.

- **Software as a Service (SaaS)** – various applications that are provided online to consumers by cloud providers. It is entirely run, maintained and managed by the cloud provider. An example of this model is web-based email.

Additionally, NIST defines five major actors involved in cloud computing: *cloud consumer*, *cloud provider*, *cloud auditor*, *cloud broker*, and *cloud carrier*. Each of these actors is an entity- that is,a person or an organisation that takes part in a transaction or process, or performs a task in cloud computing. In addition, each actor has a fundamental role in the cloud computing realm. For example:

- *Cloud consumers or cloud users* – these are the stakeholders who use cloud services provided by a cloud provider. They might be persons, enterprises or even governments.

- *Cloud providers* – they are responsible for providing one or more of the cloud service models to the cloud consumers, either directly or through a broker. In general, a cloud provider carries most of the responsibility for managing and controlling security, privacy, service configuration, service management and business support. However, many other responsibilities are assigned to them depending on which service model they provide.

- *Cloud auditor* – an independent party that examines and evaluates cloud services

according to various factors, and provides their assessment to either to cloud providers or cloud consumers.

- *Cloud broker* – an intermediary party that manages the relationship between cloud consumers and cloud providers. Brokers provide various services to consumers, including managing performance, managing security and combining multiple services into one or more new services, either (from one or many cloud providers).

- *Cloud carrier* – the connectivity and transport channel between the cloud provider and cloud consumer.

2.2 Business Drivers and Cloud Benefits

Several primary business drivers encourage cloud consumers, especially enterprises, to adopt cloud computing services. As an example, planning computational resources to determine and fulfil the future use of computational resources can be challenging for any organisation, as such planning requires a good estimation of fluctuation in usage load. A discrepancy between the resource capacity and usage requirements can result in a system either over-provisioning or under-provisioning. Over-provisioning mean a company has too many infrastructure resources, with a resulting unnecessary financial investment.

Under-provisioning refers to resources that cannot fulfil consumer demands at all times, leading to transaction losses and usage limitations or inefficiency at peak times. The elasticity characteristic of cloud computing offers a perfect solution for any organisation to cope with a capacity planning problem, and to fit its computational resources to its needs at all times. Merging the elasticity characteristic with a pay-per-use model helps organisations to eliminate the cost of acquiring new infrastructure resources and the cost of ongoing ownership of the infrastructure (operational expenses). Operational expenses can include hiring technical people to keep the infrastructural resources operational, paying utility bills, and paying for licenses and support arrangements. Moreover, any organisation may face a change in business requirements. Such changes might necessitate an IT response by scaling the computational resources of the organisation up

or down to meet the change. The changes to the business might be temporary or permanent; for example, reducing business expenses by closing branches or decreasing the number of employees, which requires shrinking computational resources to adapt to the new situation. By contrast, computational resources might need to expand if the business is growing.

The characteristics of cloud computing (like on-demand self-service) offer an organisational agility for organisations, allowing IT departments to respond to market changes by increasing or releasing their computational resources at any time with minimal managerial effort. Finally, data storage in any organisation can grow rapidly due to the increasing reliance on information, especially by businesses. Growth in data storage requires an increase in IT budgets to maximise computational resources, including storage, cooling systems, floor space, servers, networking and IT staff. Due to this growth, organisations are increasingly moving their data into cloud storage. Cloud storage services, which are the central focus of this research, is the strongest growth among all cloud computing services [Linden 2012]. Cloud storage is discussed further in Section 2.4.

2.3 Risks and Challenges

Despite the above-mentioned advantages and benefits, many concerns – pertaining mostly to public cloud services – constrain the movement of organisations towards this new business model. Moving to cloud computing services means renting off-premises computational resources that are managed by the cloud provider. This raises questions and concerns about the cloud provider's procedures to protect data from destructive physical forces (e.g. fire, flood or earthquake), and to secure the data from any unwanted actions by unauthorised users. In addition, guarantees of consistent online cloud service availability, without disruption from power outages, became a major concern after many cloud services suffered from outages. Examples include Telstra's cloud computing, which suffered a major outage of about 24 hours in April 2013, due to a failure in the data storage equipment [LeMay 2013]. Another example was an Amazon EC2 site that

went down in North Virginia in 2012 because of thunderstorms in the area. As a result, many websites and services, including Netflix, Instagram and Pinterest, were affected and were taken out of service during the outage [Smith 2012]. Additionally, serious concerns exist about cloud service continuity, especially after some cloud providers – such as Nirvanix Cloud Storage and Copy Cloud Storage [Marshall 2013] and Copy Cloud Storage [Brinkmann 2016]– discontinued all or some of their services. Finally, the risk of vendor lock-in is another challenge in cloud computing. Vendor lock-in means that cloud consumers are unable to move their data or services to another cloud provider without substantial switching costs. More specifically, each cloud provider requires a particular API, which means that cloud consumers must develop their applications to interact with their cloud service. Thus, moving to another provider means the consumer must redevelop their applications around the new cloud provider's API requirements. A lack of API standards in the cloud computing environment is becoming a major problem for consumers. There is some effort to create a unified API which can interact with different cloud providers (e.g. Jcloud [Jclouds 2016], Libcloud [Libclouds 2016]). However, these unified APIs cannot interact with all cloud providers. There have also been many attempts by non-profit organisations to develop standardised APIs, including OpenStack, Standards Acceleration to Jumpstart Ad-option of Cloud Computing, and The Open Group Cloud Computing Work Group. None of these is widely accepted by cloud providers [Lewis 2013] [Liang 2016].

The rest of this chapter focuses on the concept of cloud storage services. The discussion includes an examination of the major concerns and issues that are relevant to the research.

2.4 Cloud Storage Services

Cloud storage refers to virtual data storage devices that are used to store data online. Broadly, virtualisation uses tools to partition a physical computer (a server, database, hard disk, etc.) into multiple virtual machine images, so that each behaves like a separate machine.

Table 2.1: Comparison of online storage types

Features	Drive	Cloud Storage
Target consumer	Individual users	Enterprises
Manner of accessing	App or Internet Browser	API
Storage space	Limited free space	Unlimited
Payment mechanism	Monthly fee if more space required	Pay-per-use mechanism

Cloud storage is gaining considerable attention due to the dramatic increase in data volume within organisations. As shown in Table 2.1, two different types of online storage are provided by the cloud provider. The first type is personal online storage for ordinary cloud consumers, with the consumer accessing their data through an Internet browser or via an application (which is usually provided by the cloud provider or a third party). Usually, this type of service offers limited free space and a fixed price per month for extra space. This type is often called a 'drive'; examples include GoogleDrive, OneDrive, Amazon Cloud Drive and Dropbox.

The second type is designed to work as a utility service, mainly for enterprises. Often this type is called a 'cloud storage service'. It provides unlimited storage space and supports a pay-per-use mechanism. This mechanism is usually based on the amount of data stored and the amount of network bandwidth used, as well as the numbers of operations performed by the client – including add, update and delete. Providers do not provide an application for this type. Cloud consumers must integrate their system applications with these services through an API based on the cloud provider's requirements. This type of service is attractive to businesses as it offers benefits such as flexibility, scalability, and reduced expenditure on technology infrastructure ([Yang & Jia 2014; Furht 2010; Thakur & Lead 2010]). Examples include Google Cloud Storage, Amazon S3, Microsoft Azure Storage and RackSpace file cloud. The characteristics of this type are based on the NIST definition (discussed in Section 2.1.

This research concentrates on the second type, cloud storage services. From now on in this thesis, the term 'cloud storage' refers only to this type. However, before

Table 2.2: Common RAID levels with strengths and weaknesses

RAID Levels	Min.Num of Drives	Description	Strengths	Weaknesses
RAID 0	2	Data striping without redundancy	the highest performance	No reliability, if one hard-disk fails, all data are lost
RAID 1	2	Disk mirroring	Good performance and reliable	High cost overhead for redundancy , as all data are duplicated.
RAID 5	3	Data striping with distributed parity	Good balance between performance and reliability	Almost none
RAID 10	4	Combination of RAID 0 and RAID 1	High performance and high reliability	Same as for RAID 1 (redundancy cost overhead), as all data are duplicated. Requires minimum of four hard-drives

examining this type of storage in greater detail, it is important to understand how data storage strategies have developed, and how the issues for traditional hard disks that are relevant to cloud storage were solved.

2.5 Data Storage Strategies

'Data storage' refers to technology that allows computers to retain digital data (files, videos, images, etc.) on a magnetic hard disk. Before 1988, the strategy in most systems was to use a single large magnetic hard disk, known as a single large expensive disk (SLED), to store data. The primary concern with this strategy was the possibility of disk failure (i.e. reliability), which meant that data were susceptible to being lost if the SLED failed. Data could only be restored if they had been backed up onto another disk or tape before the disk failed [Johnson 2009] [Stone 1993]. In 1988, researchers at the University of California published A Case for Redundant Arrays of Inexpensive Disks (RAID) [Patterson et al. 1988], with the main aim of overcoming the problems of SLED.

The theory of RAID is to spread the data (by striping or mirroring) over an array of magnetic hard disks, which becomes an alternative to SLED. Broadly, RAID provides varying degrees of ability to avoid the potential reliability problems inherent in using

SLED, along with improving the speed of input / output (I/O) transactions between users and hard disks (I/O performance) [Buyya et al. 2001].

RAID uses several methods to spread data across multiple hard disks. These methods have been standardised into many levels. Some levels focus primarily on either reliability or performance, and some combine both aspects [Buyya et al. 2001]. The levels can be categorised according to three main methods:

- *Data striping*: dividing data into even segments (blocks) and then spreading them out over an array of hard disks so that more than one hard disk is read from and written to simultaneously.
- *Data mirroring*: replication of data on two or more hard-disks.
- *Data striping with parity bits*: data are striped into segments and distributed across an array of disks, with a bit-wise 'exclusive OR' (XOR) function to compute a parity bit value from the array data. This value is used to reassemble the data segments in case of hard disk failure.

Nowadays, cloud storage is becoming the new trend in data storage strategy development. However, adopting a single cloud storage service can lead to temporary issues of unavailability, which resemble SLED for the duration of unavailability. Hence, organisations are likely to face the same problems as with SLED, which are mainly reliability and performance issues, but in a different scenario. This raises the question whether the standard RAID technology can offer solutions for a concern that uses a single cloud storage service. Few researchers have addressed the issues of single cloud storage; most provide solutions on multiple storage services. A review of relevant literature is presented in Section 2.6.

In general, employing RAID for multiple cloud storage services can improve reliability. However, it is hard to optimise performance while reading from and writing to different cloud storage services concurrently, because many factors affect the performance of cloud services. Examples of these factors include the distance between the consumer and the cloud servers, and the cloud network structure (e.g.throughput, and access

policies). As a result, performance is restricted to the slower cloud services. This research considers these challenges and proposes an intelligent solution, namely adopting multiple cloud storage using the principle of RAID.

2.6 Multiple Cloud Storage Solutions

As mentioned in the previous section, several researchers have addressed the issue of dependence on a single cloud storage service. This section provides a review and discussion of that work. In general, the literature addresses the problems associated with dependence on a single cloud provider by adopting multiple cloud storage services. In brief, adopting multiple cloud storage uses a combination of several independent cloud services and considers them to be one cloud storage. Some of the advantages of distributing data over several cloud storage services are increased availability, increased performance, and reduced probability of losing data. However, this approach can increase the amount of storage and bandwidth used and therefore the cost goes up. This scenario is not at odds with the earlier statement that performance is restricted to the slower cloud services, because improvement in performance is indeed restricted to the slower cloud services. That is, the client application must wait for the slower cloud to respond each time, which means the fastest cloud will be not noticed because the user simply waits for the slower one.

Scalia [Papaioannou et al. 2012] introduced a cloud brokerage solution that continuously adapts the placement of data, based on file access statistics over several cloud storage services. This approach minimises the storage cost, improves the data availability, and eliminates the risk of vendor lock-in. However, the work of Scalia focused on the optimisation of cost and does not evaluate the impact of the solution on latency time (performance of transaction). The high-availability and integrity layer (HAIL) approach [Bowers et al. 2009] uses the principle of RAID to distribute files across a collection of cloud storage services. This approach enhances the availability of data and allows for the remote management of data security risks (i.e. data integrity) in the cloud, by employing the proof of retrievability (PORs) system. Although HAIL reduces the storage costs, it

does not consider the effect of access patterns on the network cost. In addition, the file is assumed to be static and the impact of the solution on performance is not considered. The multi-clouds database model (MCDB) [Alzain et al. 2011], is a framework that employs Shamer's secret sharing algorithm [Shamir 1979] to improve the data integrity, data intrusion, and service availability. The RAID system was also employed [Mu et al. 2012]. Their study describes a prototype system called μLibCloud, which leverages RAID to improve the availability, read-and-write performance and global access experience of clouds, along with fault tolerance of cloud storage services. However, this work did not consider optimising the cost and stated that costs would be subject to increases. [Bessani et al. 2011] presented a system called DepSky, which employs a cryptographic secret sharing scheme with erasure codes to avoid vendor lock-in, in addition to enhancing the availability and efficiency of distributed data. Although DepSky showed an improvement in performance, the authors state that the average cost increased by twice the cost of single cloud storage. [Paraiso et al. 2016] presented a service-oriented component-based Platform as a Service (PaaS) for managing elasticity, portability, provisioning, and availability across multiple cloud providers. High availability in their approach was achieved in two ways:

(1) The provision of a multiple cloud load balancer service, which fronts traffic for applications deployed over multiple clouds, and decides where to route traffic when a cloud suffers from outage. (2) Redundancy is used at all levels to ensure no single component failure in a cloud provider will affect the system's availability. This approach focuses mainly on availability and does not consider optimising the cost and performance of distribution across multiple cloud services.

[Kajiura et al. 2015] introduced an approach to dynamically determine optimal cloud storage services, for storing data in heterogeneous multiple cloud services. This approach is based on the request of the user; for example, the user can require more secure environment or a lower cost. The approach allows for trading off between cost, availability, and confidentiality. Their work considered only the the storage cost, and there was no consideration of network and operational costs. Moreover, Kajiura et al. took no ac-

Table 2.3: Comparison among distribution frameworks : Key: S. cost, storage cost; N. cost, network cost (i.e. transaction cost); O. cost, operation cost (read, write, and delete).

Framework	**the goal is to optimise:**				**Tackling problems of**		**# clouds**
	S. Cost	N. Cost	O. Cost	Latency time	Vendor lock-in	Availability	Multiple
HAIL	✓	x	x	x	✓	✓	✓
Scalia	✓	x	x	x	✓	✓	✓
MCDB	x	x	x	x	✓	✓	✓
μLibCloud	x	x	x	✓	✓	✓	✓
DepSky	x	x	x	✓	✓	✓	✓
Deco	✓	x	x	✓	x	x	x
soCloud	x	x	x	x	✓	✓	✓
Heterogeneous-multi-cloud	✓	x	x	x	✓	✓	✓
Data management approach	✓	x	x	x	✓	✓	✓
Proposed framework	✓	✓	✓	✓	✓	✓	✓

count of the effect of their approach on the performance of distribution. Similarly, [Kanai et al. 2014] proposed a secret distribution data management approach for multiple cloud storage services to maintain confidentiality. The motivation behind their approach was to improve the confidentiality of data. They assessed the storage cost and availability but not the network and operational costs, and they did not evaluate the performance.

Most of the above solutions neglect the differences in the network characteristics of cloud services, which affect the connectivity between client and cloud storage services. Each cloud storage possesses unique network architecture and access policies, which affect the performance (latency time) of reading and writing the data. Moreover, to our knowledge no-one has proposed a solution to optimise both cost and performance in the distribution of files across multiple cloud storage services. However, it is worth mentioning that [Zhou et al. 2015] propose a system called 'Deco' to optimise cost while keeping performance at reasonable levels. Even so, the Deco system is designed for dis-

tributing process tasks across multiple instances in a single cloud for workload purposes. Furthermore, it does not take account of network and operational costs, or of any change in the cloud pricing scheme. In addition, the efficiency of the Deco solution is restricted by the slowest cloud service used, if the approach is implemented across multiple cloud storage services.

Concerns have been expressed about hybrid cloud computing to solve some of the above-mentioned challenges, including security and availability [Tanimoto et al. 2013][Tanimoto et al. 2013]. Usually, the Hybrid solution is based on using private and public clouds. This solution is not advisable for small and medium-sized enterprises because of the problem of high cost to build a private cloud. In addition, researchers have compared the solution of adopting hybrid clouds and distributing data over multiple clouds, and concluded that distribution over multiple clouds is more advantageous and flexible than hybrid clouds [Kajiura et al. 2015][Kajiura et al. 2013]. The main limitation in all the works discussed above is the lack of a comprehensive solution to optimise both the cost and the performance of cloud storage concurrently. (Total cost includes storage cost, network cost, and operational cost.) These two factors improve data availability and abolish the risk of vendor lock-in. Hence, the results presented in this thesis provide an intelligent framework, built on a combination of two machine learning algorithms, to overcome the issues and limitations of previous solutions. Table (2.3) provides a summary of the differences between the various solutions mentioned above and the framework proposed in this thesis.

2.7 Summary

Cloud computing is a dynamic environment. This dynamism means that the pricing scheme, service performance and service continuity and availability are liable to change at any time. In addition, optimising both cost and performance in such an environment is a challenge, and optimising only one factor can unintentionally affect the other factor adversely. Hence, it is essential for any optimisation solution in the cloud computing environment to address all these difficulties. Due to their ability to adapt to changing

environments, machine learning algorithms provide a suitable approach to solving many problems in cloud computing. Hence, the next chapter gives an overview of machine learning, followed by a survey of the applications of machine learning in the cloud environment. Thereafter, the proposed framework is introduced.

CHAPTER 3

Machine Learning: Background

The previous chapter addressed the issues of optimising cost and performance (mainly latency time) for various cloud services. The difficulty of optimisation arises from the dissimilarity in service performance and pricing schemes among cloud storage providers. In addition, the price scheme and performance of any cloud provider can be affected or changed at any time.

The complexity of cloud structure and services, the dynamics of performance, and the changing prices in cloud computing all render the environment interesting and challenging for machine learning researchers. Any proposed solution for solving the cloud problem should ideally account for long-term cost and performance, rather than just optimising the current state. Machine learning algorithms have an excellent ability to learn to cope with such a dynamic environment. Hence, this chapter provides an overview of the field of machine learning, including the top branches of supervised learning, unsupervised learning, and reinforcement learning. In general, machine learning is a sub-field of Artificial Intelligence (AI), which aims to learn from data to solve problems such as prediction and estimation. Machine learning is discussed in this section, beginning with

several definitions of machine learning. This is followed by a review of the key components of algorithms of the black-box model in machine learning. Thereafter, an overview is provided of various learning paradigms in machine learning. Finally, the chapter concludes with a summary of machine learning studies that have examined the prediction of file access patterns. .

3.1 Definitions of Machine Learning

Machine learning allows a computer to extract knowledge from data automatically. This field integrates various disciplines, including (but not limited to) statistics and probability, psychology, computational complexity theory, control theory, information theory, and neurobiology [Mitchell 1997]. Learning in this context is a process of finding underlying patterns in the data and describing them in the form of a mathematical function.

The story of machine learning can be traced back to the 1950s, when Arthur Samuel created the first checkers program on IBM's first stored program computer (computers that can store and run a program electronically stored in an electronic memory) [McCarthy & Feigenbaum 1990]. Samuel's machine learning approach was developed to learn from experience how to win a checkers game using a heuristic search algorithm. Since then, the methodologies of this field have evolved, and many learning algorithms have been introduced in various applications to solve different problems. Such problems range from detecting email spam to performing face recognition, to developing an autonomous vehicle that can learn to drive on a public road. Generally, learning algorithms in this field are categorised into three learning paradigms: supervised learning, unsupervised learning, and reinforcement learning. Details of these paradigms are provided in Section 3.3.

The term 'machine learning' seems to have been coined by Samuel himself [Naqa & Murphy 2015]. A number of sources in the literature (e.g. [Khanna & Awad 2015], [Bell 2015] and [McClendon & Meghanathan 2015]) state that Samuel defined the field of machine learning in 1959 as follows:

A field of study that gives computers the ability to learn without being expli-

citly programmed

Although this definition is popular, unfortunately the main source of the definition could not be tracked down. However, the above definition provides a clear meaning for machine learning. The second formal definition of machine learning was published in 1997 in Mitchell's book, titled simply Machine Learning [Mitchell 1997]:

> *A computer program is said to learn from experience **E** with respect to some class of tasks **T** and performance measure **P**, if its performance at tasks in **T**, as measured by **P**, improves with the experience **E**.*

This means that in general, experience leads to improved performance when the computer runs a set of tasks. A further definition was given by Murphy [Murphy 2012]:

> *Machine learning is a set of methods that can automatically detect patterns in data, and then use the uncovered patterns to predict future data, or to perform other kinds of decision making under uncertainty.*

Another accurate definition was presented by Marsland [Marsland 2015]:

> *Machine learning is about making computers modify or adapt their actions (whether these actions are making predictions, or controlling a robot) so that these actions get more accurate, where accuracy is measured by how well the chosen actions reflect the correct ones.*

All these definitions share the idea of a computer being able to learn from data to perform specific tasks. In general, machine learning uses data-driven methods employing a large set of algorithms from another branch of knowledge, and combining them with the power of the computer to learn the relationships among certain data. These relationships are then represented in some form of model (hypothesis) function.

Machine learning algorithms use black-box and white-box models (the reader will find more details later in Figure 3.3). In general, a white-box model builds a hypothesis using readable and interpretable algorithms. Examples of white-box models include

decision tree learning [Rokach & Maimon 2014], rule learning [Frnkranz et al. 2012] and nearest neighbour learning [Duda et al. 2000]et al. 2000]. These models are often considered to be nonparametric learning algorithms. By contrast, , the black-box model does not require any prior learning about the data. Usually, a black-box model uses adjustable parameters to describe a pattern in the data. Examples of black-box modelling including artificial neural networks and linear regression. All algorithms of machine learning discussed in this thesis refer only to black-box models; also, the reader should note that some of the terminology used in this work differs slightly from other literature in this field. The reasons for that is to keep terminologies consistent with the different branches of machine learning discussed in this thesis, namely supervised learning and reinforcement learning.

3.2 Machine Learning Components

As mentioned, this thesis focuses on machine learning algorithms that output a parametrised hypothesis – which are mainly black-box models. In general, most machine learning algorithms that belong to a black-box model structure are based on the interaction between two components, to build an accurately parametrised model that represents the pattern of the data. This pattern is depicted in Figure 3.1, but it should be noted that Figure 3.1 is not comprehensive and provides only a high-level overview of machine learning methods. The parameters of this model are unknown. Hence, the goal of machine learning is to search through the space of possible hypotheses (parameter values) to fit the model to the data. The first component is 'environment', which consists of data that need to be learned. The second component is the machine learning system, which is computer software or tools that are capable of learning from the environment; these are often called 'agents'. In general, especially in black-box modelling, the machine learning system consists of two sub-components:

- *Hypothesis representation* – a model function that consists of parameters (θ) ¥ which represent a pattern in the data of the environment.
- *Optimisation algorithms* – these are used to analyse the data and tune the paramet-

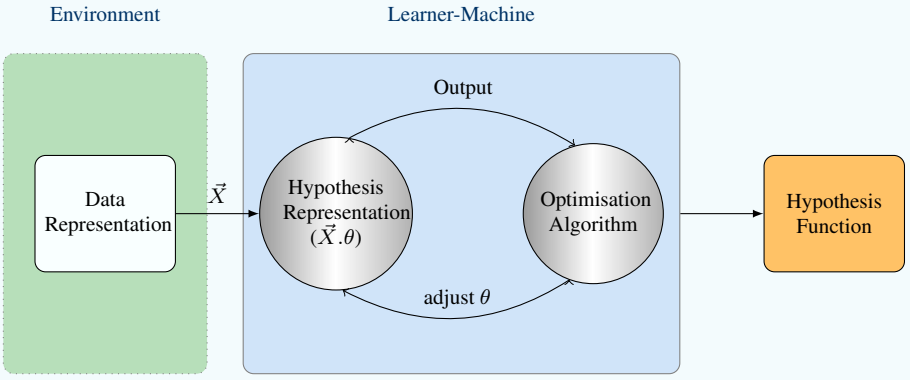


Figure 3.1: Block representation of process and components of machine learning

ers (θ) .

Evidently, the key design of a machine learning component is based on the choice of the hypothesis representation function [Mitchell 1997]. Thus, it is important to determine how the machine will represent its parameters in a mathematical model and the best algorithm to optimise the model. More information about formulations of hypothesis representation and optimisation algorithms is presented in Sections 3.2.2 and 3.2.3. However, first it is necessary to understand the environment and data that machine learning can learn from; these topics are discussed below in Section 3.2.1. Please note, Figure 3.1 is not comprehensive and provides only a high-level overview of machine learning methods.

3.2.1 Environment and Data Representation

During the learning process, the machine learning system receives or obtains a dataset (D) that describes the state of its environment. D can be collected and made availableto the machine learning system after being prepared and cleaned by a human expert, or the machine learning system can acquire D through iterative interaction with the environment. The dataset in the broadest sense is represented in the form of a table matrix (Figure 3.2), where each row is an input vector $\vec{X}$, called an instance. Instances are characterised by independence of features (denoted $x_i \in \vec{X}, i \in \mathbb{N}$) that measure different

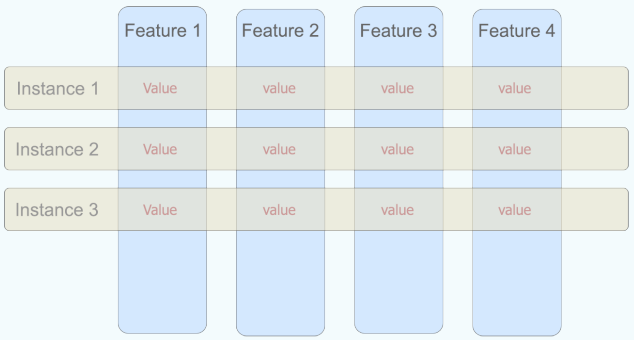


Figure 3.2: Table matrix to represent data in machine learning

aspects of the instance [Witten & Frank 2011].

Understanding the effects of feature characteristics in machine learning is a fundamental step in choosing the machine learning components that fit the hypothesis function. These characteristics are summarised in the following points:

- **Feature type:** this term refers to whether the feature value is numeric or nominal. A numeric value means the data take numerical values, either real numbers or integers. By contrast, nominal (or categorical) data refer to a string of characters. Nominal features require encoding schemes to represent the values numerically before the learning process can be started.

- **Feature continuity:** this term refers to whether the feature value is discrete or continuous. 'Continuous' here typically refers to an infinite or very large numeric feature space [DBL 2005] [Mitchell 1997]. By contrast, discrete values indicate a small set of values, either numeric or nominal. .

- **Feature space dimensionality:** this term refers to how many elements are used in the feature vector to represent the environment. This characteristic can be thought of as the number of features in D that the machine learning system receives. A high degree of dimensionality might slow the learning process. Thus it is im-

portant to select and extract only the features that are correlated with the problem being learned.

The fundamental task of machine learning is to learn or discover the pattern of D. This goal can be accomplished through three learning paradigms:

- **Unsupervised learning**: In this paradigm, D consists only of input vector ($D(x_i), x_i \in \vec{X}$). Usually, the goal of this paradigm is to search in the data for similarities and to generate classes of patterns [Bishop 2006]. This paradigm falls outside the scope of this work, but a brief description is provided in Section 3.3.1.

- **Supervised learning**: in this paradigm, D consists of two vectors (1) an input vector $x_i \in \vec{X}$ and (2) a corresponding supervisory single $y_k \in \vec{Y} : k \in \mathbb{N}$. The vector $\vec{Y}$ is the desired output , which is either a discrete-valued quantity or a continuous-valued quantity. In this paradigm, the input vector is an element of a fixed training dataset of D, and the output values y_k are then assigned to the dataset , in the form $D(x_i, y_k)$. The goal is to map the input vector to its corresponding desired output, $\vec{X} \rightarrow \vec{Y}$. More details about this paradigm are discussed in Section 3.3.2.

- **Reinforcement learning**: This paradigm is similar to supervised learning. However, instead of the model aiming for an explicit corresponding desired output, the environment is designed with a reward signal r that tells the machine how appropriate its decision or action is. This takes the form $D(x_i, r_p) : p \in \mathbb{N}$. Based on this reward signal, the machine learning system searches for the best decision or action (i.e the action that maximises the cumulative reward) that maps the input vector to an action denoted as $\vec{X} \rightarrow a_j$. Here, a_j refers to the range or list of actions ($a_j \in A$) that are available to the machine learning system in each state. Another difference between supervised learning and reinforcement learning is that reinforcement learning does not require a fixed training set from which the input vectors are taken. Additionally, the reinforcement learning system is embedded

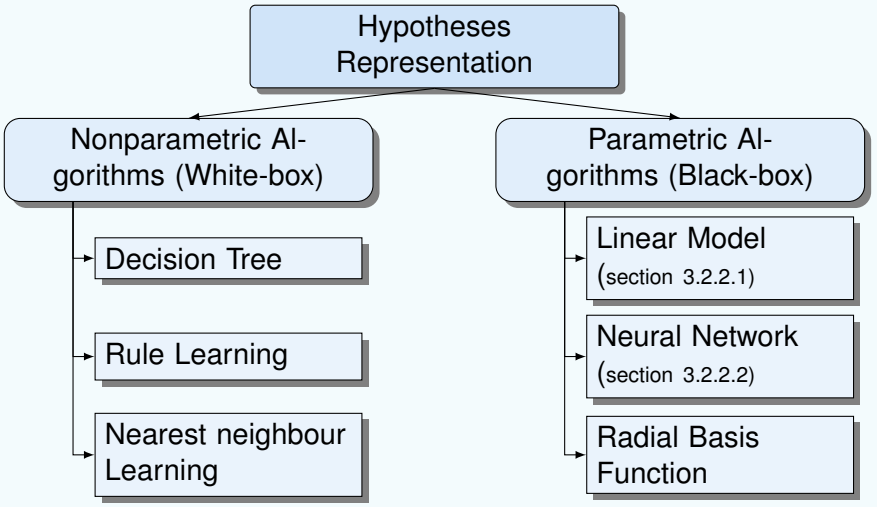


Figure 3.3: Representation Type

into an environment in which each action affects the state of the environment, and is followed by a numerical reward that tells the learning system how good or bad the action is (i.e. the learning system learns from a trial-and-error technique). By contrast, supervised learning requires guidance on what to do during the learning process. The details are discussed further in Section 3.3.3.

In short, machine learning can learn the pattern of a dataset in different ways, based on the characteristic of the features, the availability of corresponding desired output results, and the learning target. In general, the machine learning system requires a target function to determine how the pattern will be represented [Mitchell 1997]. In addition, the machine learning system requires an optimisation algorithm to tune the representation function parameter. These components, as they occurred in this research, are discussed individually in the following sub-sections.

3.2.2 Hypothesis Representation

Hypothesis representation is a model to underline a structural pattern in a dataset; it allows a machine learning system to make a decision or to perform a prediction for the future. Mostly, the model develops of mathematical expressions that consists of a set of

coefficient parameters (denoted here θ) that represents a knowledge (pattern) in a given D. Many formulations of representation model have been used in machine learning to represent the hypothesis based on the data structure and the target of learning. The simplest and most basic form of representation is a look-up table [Witten & Frank 2011], which in some literature is called a tabular representation [Sammut & Webb 2011]. However, this type of representation is usually limited to relatively small and discrete feature spaces, due to memory constraints [Sammut & Webb 2011][Marsland 2015]. Furthermore, this kind of representation is used mainly with reinforcement learning problems rather than with supervised learning problems. However, when the feature space is continuous or grows to a larger size, some form of function approximator with parameters may be used instead. Indeed, the function approximator outperforms the look-up table because of its ability to generalise and estimate between feature values [Marsland 2015]. In addition, it is more practical to represent continuous feature space. The linear model is one of the most widely used as a function approximator in machine learning. It is a powerful and simple mathematical model that can be used to make an estimation based on a linear combination of the input and output variables. Another approximator is an artificial neural network model, which has been used in various of capacities for machine learning. Although many other function approximators have been used to solve different types of problems, as shown in Figure 3.3, the scope of this work concerns only linear function and artificial neural network models.

The learning process of most machine learning algorithms is implemented through two passes. The first is called the *forward pass*, when the input vector is applied, and the output is calculated using a function approximation. The second pass is called the *backward pass*, which in general means adjusting or tuning the parameters of the function approximation to optimise the accuracy of the output, using some form of optimisation method. In this thesis, the two passes are discussed separately. For the forward pass,two of the most popular function approximations are considered, namely the linear model (Section 3.2.2.1) and an artificial neural network (Section 3.2.2.2). For the backward pass, *regression* and *back propagation* are discussed in Section 3.2.3 as optimisation of

algorithms.

3.2.2.1 Linear Model

A linear model is a black-box model. More specifically, it is a simple mathematical equation that represents a relationship between one or more independent variables $x_i, i \in \mathbb{N}$, and one dependent variable y. This relationship is shown by a straight line when plotted on a $k \in \mathbb{N}$-dimension graph. There are main two types of linear equation. The first is a simple linear model in which the linear equation involves one independent variable $x_i, i = 1$. The second type is called a multiple linear model, , which involves than one independent variable, $x_i, i > 1$. In general, machine learning applications consider the multiple linear model, which takes the general form:

$$\hat{y} = b + \sum_{i=1}^{n} w_i \times x_i \tag{3.1}$$

where b is the intercept (In machine learning it is known as *bias* [Hastie et al. 2009]) which is the value of $\hat{y}$ when $\vec{X}$ = 0; $w_i \in \theta$ is the slope of the line. This form of the linear model has been widely used in machine learning and statistical learning to define the pattern of a dataset in linear algebraic relation [Witten & Frank 2011]. Figure 3.4 is a a high-level block diagram that illustrates the relationship between the input vector and the output variables.

The values of parameters b and w are unknown. Hence, at the beginning of the learning process, these values are set arbitrarily. The machine learning system then optimises them using optimisation algorithms to fit the data points to a straight line. The most widely used optimisation algorithms with the linear mode is regression analysis. More information about this kind of optimisation algorithm is presented in Section 3.2.3.1 . However, often a linear function is not appropriate for modelling a non-linear relationship between the output and input vectors. In addition, if the dimensionality of the feature space is very high, representing the output using a linear model becomes intractable.

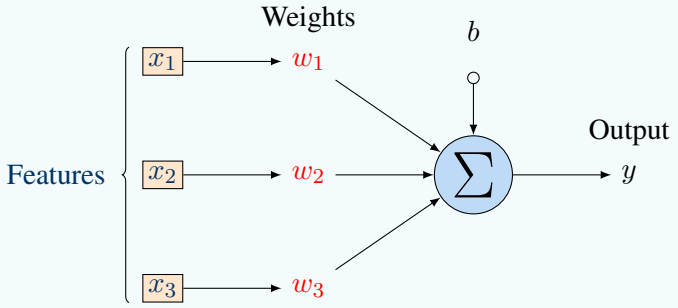


Figure 3.4: Block diagram for linear model

3.2.2.2 Artificial Neural Network Model

An Artificial Neural Network is an alternative black-box model approach to approximate an output (y) from a given dataset D. It is a powerful and complicated mathematical model that can represent complex linear and non-linear functions [Mitchell 1997]. Figure 3.5 depicts the basic implementation uses of an artificial neural network, within a feed-forward neural network. It consists of three layers of computational neuron units called *nodes*. Typically, the first layer- called the *input layer*- takes the input features $x_i \in X$, where $i \in \mathbb{N}$ is the index of the input feature. The second layer is called the *hidden layer* (denoted h_j), where $j \in \mathbb{N}$ is the index of hidden nodes. The last layer is called the *output layer* (denoted y_k) where $k \in \mathbb{N}$ is the index of output nodes. Each node in each layer is fully connected with all nodes in the next layer and each link is associated with a weight value, denoted here as $w \in \theta$. The value of the nodes in the hidden layer is computed by the following equation:

$$h_j = b_j + \sum_{i=1}^{n} x_i w_{ij} \tag{3.2}$$

where n is the number of inputs and b_j is the bias for hidden nodes, similar to the method described above for linear models (Section 3.2.2.1). The term w_{ij} refers to the connection weight between input node i and hidden node j. In addition, the output nodes

is computed as follows:

$$y_k = b_k + \sum_{j=1}^{m} h_j w_{jk} \tag{3.3}$$

where m is the number of all hidden nodes, b_k is another bias for output nodes, and w_{jk} is the connection weight between hidden node j and output node k.

One of the main characteristic elements of an artificial neural network is the transfer function [Haykin 2007]. This function restricts the output range of each node in the hidden and the output layer between two small values. One of the benefits of using a transfer function is to allow the artificial neural network to learn non-linear relationships. Many mathematical functions can be used to transfer values in an artificial neural network. These include linear function (for the output nodes only), sigmoid function, step function, ramp function, gaussian function, and hyperbolic tangent function. Two types of transfer function (Figure 3.6) are discussed here. The most common form is the *sigmoid function* [Haykin 2007], which can be stated as:

$$\varphi(z) = \frac{1}{1 + e^{-z}} \tag{3.4}$$

where z is any output variable from hidden or output nodes. This function graph can be thought of as an S-shaped curve that bounds the output values between [0,1]. The second function is the *hyperbolic tangent function* (*tanh* for short), which is defined as the ratio between the value of hyperbolic sine and cosine functions [-1,1]. It takes the following form:

$$\varphi(z) = tanh(z) = \frac{sinh(z)}{consh(z)} = \frac{e^{(z)} - e^{-(z)}}{e^{(z)} + e^{-(z)}} \tag{3.5}$$

The block diagram (Figure 3.5) shows a basic artificial neural network structure with three input nodes, three hidden nodes, and one output node. This structure can be exten-

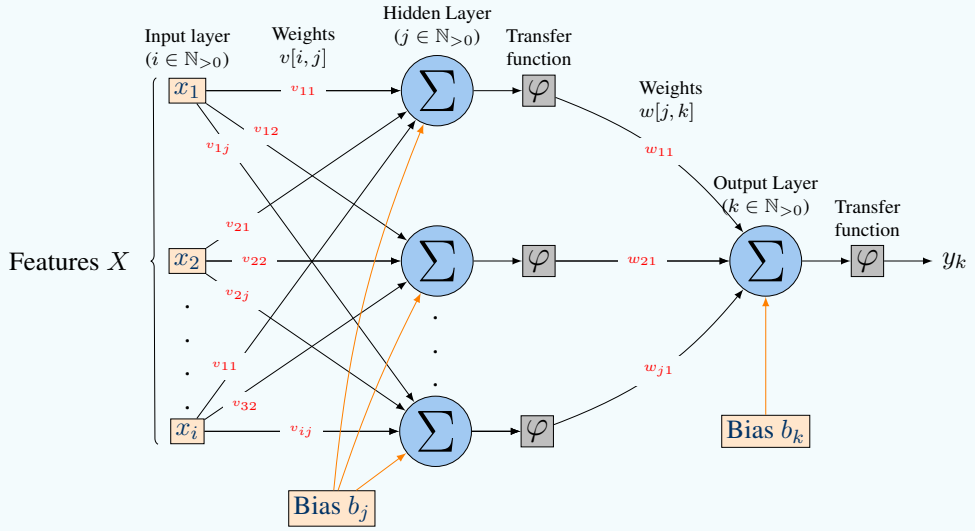


Figure 3.5: A fully connected feed-forward neural network structure consists of three layers of nodes: (1) input layer denoted as $x_i, i \in \mathbb{N}$, (2) hidden layer denoted as $h_j, j \in \mathbb{N}$ and (3) output layer denoted as $y_k, k \in \mathbb{N}$.

ded to include any number of hidden nodes or multiple hidden layers, and may also have several output nodes.

The use of an artificial neural network for machine learning is attractive for many reasons. First, unlike the linear model, it provides a robust approach to approximate different output values, including continuous output from a complex dataset structure [Mitchell 1997]. This power comes from the ability of the hidden layer(s) to extract implicit features or information that cannot be defined in the input vectors [Haykin 2007].This feature is also, one of the benefits of using the transfer function. Artificial neural networks usually use a back-propagation algorithm as an optimisation function to adjust the connection weights via the gradient descent method. More information on this topic appears in Section 3.2.3.2

As state earlier, in this research the forward pass algorithm is separate from the optimisation algorithm (backward pass). Algorithm 1 is the forward pass algorithm for the artificial neural network.

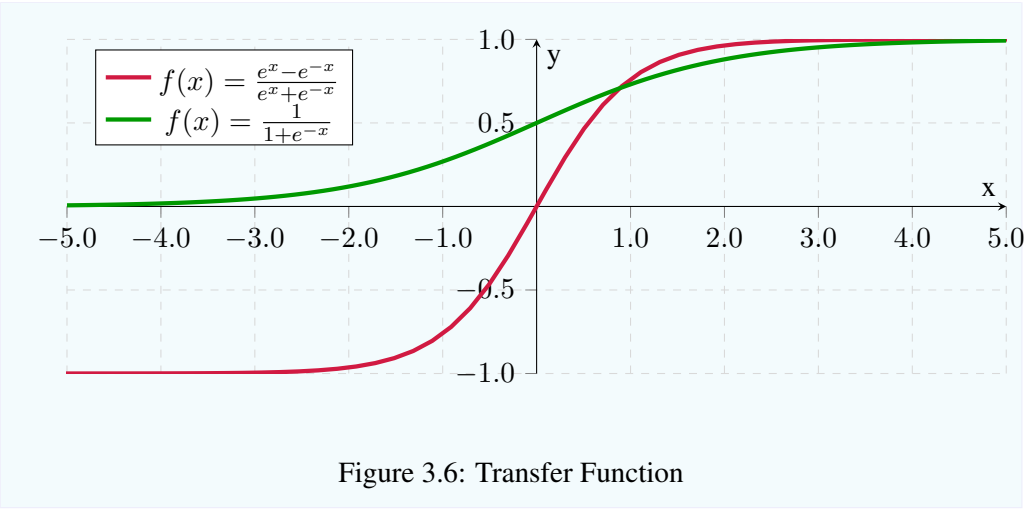


Figure 3.6: Transfer Function

Algorithm 1 Artificial neural network framework, forward action algorithm

Require: : Initialise array weight w_{ij}, w_{jk} with random values
Require: : Initialise bias $b_j = 1, b_k = 1$
1: **for** <each hidden node j> **do**
2: compute h_j ▷ according to Eq. 3.2
3: $h_j = \varphi(h_j)$ ▷ according to Eq. 3.4 or 3.5
4: **end for**
5: **for** <each output node k> **do**
6: compute y_k ▷ according to 3.3
7: $y_k = \varphi(y_k)$ ▷ according to Eq. 3.4 or 3.5
8: **end for**
9: **Return** y

3.2.3 Optimisation Algorithms

As shown in previous sections, the coefficient parameters (θ) of the function approximator are unknown. Hence, the machine learning system requires a method to analyse the given dataset D and search through the hypotheses space to find θ that best fit the D. These methods are called *optimisation algorithms*. The optimisation algorithm is a fundamental element in each function approximation model; it aims to select the best parameters from the set of available alternatives. In the previous section, the forward pass function for computing the machine learning system output was discussed. In this section, the backwards pass to adjust the value of the hypothesis parameters θ is dis-

cussed.

Several optimisation algorithms have been used in machine learning. This research considers two algorithms: regression analysis and back propagation. These are the most common algorithms to tune the hypothesis-parameter for hypothesis representation, especially in the linear and artificial neural network models. The reason for this suitability is that these two algorithms offer good convergence that they offer to different hypotheses [Witten & Frank 2011][Mitchell 1997]. Both algorithms adjust the model's parameters by specifying a measure of the error E between the predicted output $\hat{y}$ and the desired output y (often called the residual). This measure can be stated as:

$$E_k = (y_k - \hat{y}_k) \tag{3.6}$$

where k is the index number of predicted output that has been produced by the representation. Indeed, the desired output can be pre-defined and provided to the machine learning system, if the problem is one of supervised learning, or if it is computed based on a reward signal from the environment (i.e. reinforcement learning problem). Further details are presented in Section 3.3.2 and Section 3.3.3.

3.2.3.1 Regression Analysis

Regression analysis is one of the optimisation algorithms that are used to tune parameters, especially in a linear model. Specifically, it is a statistical method for modelling relationships between variables [Montgomery et al. 2015] and it is extremely widely used statistical techniques [Montgomery et al. 2015]. In general, many types of regression analysis can be used to fit a given dataset into different formulations. Types of regression analysis include logistic regression [Hosmer Jr & Lemeshow 2004], polynomial regression [Fan & Gijbels 1996], stepwise regression [Cohen et al. 2013], ridge regression [Birkes & Dodge 2011], lasso regression [Li et al. 2005], elastic net regression [Hans 2011] , and linear regression. Linear regression is the most popular method and is used to modify the bias (intercept) and slopes so that the data points fall on a straight line.

Linear equation 3.1 stated that b and $\vec{w_i}$ are unknown constants, which represent the intercept and slopes respectively. The goal of the regression function is to find the best values of these parameters to fit a line to a given D. The first step is to measure the error E (Equation 3.6), so that Equation 3.1 can be rewritten as follows:

$$\hat{y} = b + \big(\sum_{i=1}^{n} w_i \times x_i\big) + E \tag{3.7}$$

where $n = |D|$.

Equation 3.7 is a *multiple linear regression* model. The second step is to employ a method to reduce the value of E; the most common approach in this regard is to use the least mean squares (LMS) method. The LMS method is a mathematical procedure for estimating the b and w_i, so that the sum of squares (SS) of the residuals E is minimised.

$$SS = \sum_{i=1}^{n} E^2 \tag{3.8}$$

The estimators for b and w_i (denoted here as $\hat{b}$ and $\hat{w}$, respectively) are derived through calculus to find the values that minimise SS. The final estimators of $\hat{b}$ and $\hat{w_i}$ are stated as follows:

$$\hat{b} = \bar{y} - \hat{w}\bar{x} \tag{3.9}$$

and

$$\hat{w} = \frac{\sum_{i=1}^{n} y_i(x_i - \bar{x})}{\sum_{i=1} n(x_i - \bar{x})^2} \tag{3.10}$$

where $\bar{y}$ and $\bar{x}$ refer to the means of y and x, respectively:

$$\bar{y} = \frac{\sum_{i=1}^{n} y_i}{n} \qquad \bar{x} = \frac{\sum_{i=1}^{n} x_i}{n} \tag{3.11}$$

The analysis of residuals then plays a crucial role in checking the adequacy of the fitted regression model. The process might iterate several times before an adequate model is obtained [Montgomery et al. 2015].

3.2.3.2 Back Propagation

Another approach to adjusting the representation parameters θ, which is widely used with the artificial neural network model, is back propagation. This method employs *gradient descent* (denoted δ) to adjust θ (θ also called the connection weights w), in order to minimise the error (E) between the network predicted output $\hat{y_k}$ and the desired output y_k.

In general, the weight update equation in the back-propagation algorithm can be stated as follows:

$$w = w + \triangle w \tag{3.12}$$

where $\triangle w$ is the weight correction, for weight $\triangle w_{jk}$ connection from nodes in the hidden layer h_j to nodes in the output layer y_k, which takes the form:

$$\triangle\, w_{jk} = \alpha \times E \times f'(y_k) \times h_j \tag{3.13}$$

where α is the learning rate parameter that modulates the magnitude of the weight adjustment. The term $f'(y_k)$ is the derivative of the transfer function $\varphi(y_k)$ evaluated at the induced local field h_j, which takes the general form:

$$f'(z) = z \times (1 - z) \tag{3.14}$$

where z can be replaced with the induced local field at the hidden or output nodes. In

addition, the weight update equation can be extended to update the weights $\triangle w_{ij}$ connecting nodes from the input layer x_i to nodes in the hidden layer h_j. This is expressed as follows:

$$\triangle w_{ij} = \beta \times E \times \left(\sum_j (f'(h_j) \times w_{jk}) \right) \times f'(y_k) \times x_i \qquad (3.15)$$

where $f'(h_j)$ is the derivative of the transfer function $\varphi(h_j)$ evaluated for the induced local field h_j, as expressed in Equation 3.14. The parameters β are another learning rate to modulate the weight adjustment equation for the input-hidden connection weight. Generally, these two learning rates (α and β) take different values in the interval [0,1].

Algorithm 2 describes the backward pass used to optimise the parameters θ in the artificial neural network model.

Algorithm 2 Back propagation algorithm

Require: : Initialise α and β
for <each output node : k > **do**
  compute $E \leftarrow y_k - \hat{y_k}$ ▷ according to equation 3.6
  compute $f'(y_k) \leftarrow y_k \times (1 - y_k)$
  for <each hidden node : j > **do**
    compute $\triangle w_{jk}$ ▷ according to equation 3.13
    Update network weight $w_{jk} = w_{jk} + \triangle w_{jk}$
    compute $f'(h_j) \leftarrow h_j \times (1 - h_j)$

    for <each input node : i > **do**
      compute $\triangle w_{ij}$ ▷ according to equation 3.15
      Update network weight $w_{ij} = w_{ij} + \triangle w_{ij}$
    end for
  end for
end for
Return $\vec{w}$

3.3 Learning Paradigms

In the past few decades, a vast set of learning algorithms has been introduced and developed to learn from different types of data and to solve a wide variety of tasks. These

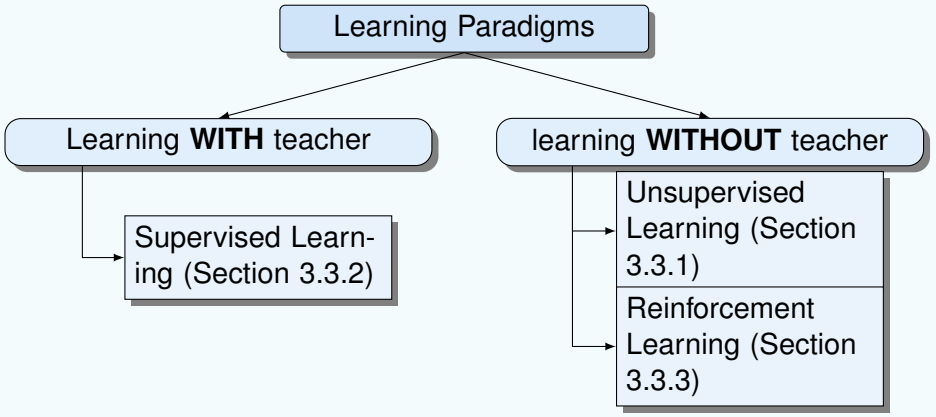


Figure 3.7: Machine Learning Paradigms

algorithms are classified into two broad categories, as shown in Figure 3.7. The first covers learning with a 'teacher', also called *supervised learning* (Section 3.3.2). The term 'teacher' refers here to a dataset, represented by a set of input-output examples, $D(x_i, y_k)$. The second category of algorithms involves learning without a teacher, which has two subdivisions: *unsupervised learning* (Section 3.3.1) and *reinforcement learning* (Section 3.3.3). This research focuses primarily on reinforcement learning and supervised learning. Therefore, unsupervised learning is discussed at a high level without going into the details.

3.3.1 Unsupervised Learning

The unsupervised learning paradigm is a data analysis technique that aims to find meaningful information from given data without prior knowledge of that data and without any guidance. More specifically, in contrast to supervised learning and reinforcement learning, there are no corresponding desired outputs or environmental feedback associated with each input.

One of the sub-fields of unsupervised learning is clustering. In general, clustering aims to organise the data into meaningful subgroups. Each subgroup that emerges throughout data analysis defines a smaller set of data that shares a degree of similarity. Clustering is a useful method for structuring information and deriving significant rela-

tionships between data [Raschka 2015]. Examples of clustering algorithms include hierarchical clustering [Flach 2012:p.253] and the k-means algorithm [Flach 2012:p.247].

Another important sub-field in unsupervised learning is dimension reduction (or dimensionality reduction). This sub-field is useful with data of high dimensionality of feature space. In a broad sense, dimension reduction is a process of reducing the number of features by removing noise from the data. Information about this sub-field can be found in [Sugiyama 2016] and [Kramer 2013]. The high dimensionality of feature space can present a challenge for the performance of machine learning, and can also affect storage space. Because unsupervised learning lies beyond the scope of this work, this paradigm and its algorithms are not discussed further here. More information can be found in [Bishop 2006].

3.3.2 Supervised Learning

Supervised learning is a type of machine learning algorithms, which use labelled datasets to learn how to perform actions or predictions about the future [Hastie et al. 2009][Mitchell 1997]. The approach relies on a "teacher" to gain knowledge from the data. The teacher in this context is a labelled dataset of input-output pairs $Ta = (x_i, y_i)_{i=1}^{N}$. From this teacher, the machine learning system learns the parameters underlying the relationship between the input vector x and the desired output value y, in order to predict an efficient decision in the future. Here Ta is called the *training set*, where $Ta \in D$, and N is the number of training sets used to train the model.

Supervised learning problems can be divided into two broad categories:

- **Classification**: In this category, the goal is to assign a label from a discrete class to the observations. That is, the output of the model is nominal, such as 'right' or 'left'; 'white' or 'black'. Applications include medical disease diagnosis, document categorisation, and social network analysis [Aggarwal 2015].

- **Regression**: In this category, the goal is to predict numerical values that are either continuous or discrete, such as 'prices', 'ages', or 'percentages'. Examples of

regression include forecasting stock prices or variations of economic variables [Mohri et al. 2012].

To illustrate this learning paradigm, a machine learning system learning might learn how to predict the weather for the the immediate future based on historical weather records. There are two types of prediction in weather forecasting that supervised learning can be deployed to solve. The first is to predict the temperature, which is a continuous variable. This kind of prediction is called a *regression problem*. The second is to predict the class of weather forecasts, such as sunny, cloudy, rainy, or snowy and so on, where the output of this prediction type is a discrete (nominal) value. This second kind of prediction is known as a *classification problem*.

3.3.3 Reinforcement Learning

In contrast to supervised learning, the reinforcement learning paradigm uses numerical feedback to evaluate the machine learning system's decisions. Reinforcement learning is a machine learning method used to tackle sequential decision-making problems through a trial-and-error technique to search for effective actions [Sutton & Barto 1998]. It is defined as a way of instructing the machine learning system by using a reward and punishment (feedback) signal without needing to specify how the task is to be achieved [Kaelbling et al. 1996]. In the broadest sense, a machine learning system, using the reinforcement learning paradigm, interacts with a single environment ; it observers the state of that environment, selects an action, and receives a scalar reward or feedback for the action. The process is depicted in Figure 3.8.

The environment, in this paradigm, is characterised by a set of states, S, in which every state is constructed from a vector of features (called state features). The machine learning system consists of a set of actions, A, that are applicable to perform on the environment (Figure 3.8). A machine learning system[1] interacts with its environment at each time of a sequence of discrete or continuous *time steps*, $t = 0, 1, 2, 3 \cdots$ The interaction takes place through a repeated cycle of three steps:

[1] In most literature on reinforcement learning, machine learning systems are called the "*agent*". However, in this thesis the term *machine learning system* is consistently used, to avoid confusion

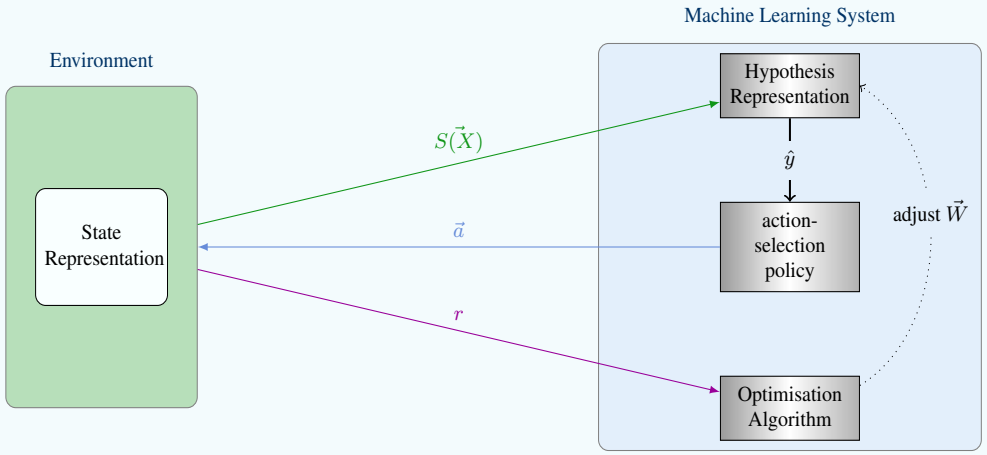


Figure 3.8: The reinforcement learning paradigm consists of a machine learning system interacting with an environment. At each discrete time t, the machine learning system observes the state of the environment s_t and performs an action a_t in order to transition from its current state to a subsequent state s_{t+1}. It receives a reward $r_(t+1)$ for the value of that transition. Over time, the machine learning system learns to improve the selection of actions that maximise the cumulative reward

(1) sensing the state of the environment at t $s_t \in S$;

(2) performing an action $a_t \in A(s_t)$, where $A(s_t)$ is a set of actions that are admissible for the state s_t; that is, $A(s_t) \subset A$;

(3) receiving a scalar reward, which in general cases is defined as $R : S \to \mathbb{R}$, which specifies the reward obtained in each state.

For the reward function, there are two other definitions. The first is that $R : S \times A \to \mathbb{R}$, which means the reward is given for performing an action in a state. The second is $R : S \times A \times S \to \mathbb{R}$, which refers to the reward obtained for a transition from one particular state to another, after performing an action. These two definitions are interchangeable, but the second one is more convenient in model-free algorithms because they require both the starting state and the resulting states to obtain the value [van Otterlo & Wiering 2012]. Throughout this work ,the definition $R(s_t, a_t, s_{t+1})$ is used primarily,

where s_t is the given state, s_{t+1} is the next state and t is the time steps. In general, the reward function indicates the utility of the action taken in the given state without specifying what the best or the worst possible action is.

At each time step t, the machine learning system interacts with its environment with the goal of building an action selection policy (denoted π_t) , which maps the states to the actions $\pi : S \rightarrow A$, where $\pi_t(a|s)$ is the probability of $A_t = a$ if $S_t = s$. The goal of this policy is to maximise the reward signal that represents a long-term objective. Thus, the policy is a fundamental step in understanding the characteristics of the reinforcement learning components before building any reinforcement learning application; the components are the environment state space, the machine learning system action space, and the reward space. These components of reinforcement learning can be formalised using a Markov decision process (MDPs) framework, especially if the state and action space are discrete. The MDPs are a standard formalism for learning sequential decision making [Wooldridge & Jennings 1995]. They are tuples (S, A, Tr, R) where:

- **S** is the set of environment states, which can take a broad range of forms. For instance, state spaces can be defined by continuous variables such as velocity, price, performance, torque etc., called *continuous state-spaces* ($|S| \in \mathbb{N}$); Alternatively, they can be defined by a *discrete state-space* if the number of states is discrete.

- **A** is the set of possible actions available to the machine learning system.

- **Tr** is the state transition function. It is defined as $Tr(s_t, a_t, s_{t+1}) \rightarrow [0, 1]$, where Tr represents the probability of reaching state $s_{t+1} \in S$ by applying action $a \in A(s_t)$ in state $s_t \in S$. A characteristics of this function is that it is deterministic. This refers to the probability of the learning system being in some state s_{t+1} after taking action a_t from state s_t, or $\rho^{a_t}_{s_t s_{t+1}}$. State transition determinism occurs when $\rho^{a_t}_{s_t s_{t+1}} = 1$. By contrast, if $\rho^{a_t}_{s_t s_{t+1}} < 1$ the transition is non-deterministic or stochastic.

- **R** is the reward function: $R(s_t, a_t, s_{t+1}) \rightarrow \mathbb{R}$. It provides an immediate indication when an action $a_t \in A(s_t)$ is taken in state s_t and moves the machine learning

system into a subsequent state $s_{t+1} \in S$.

Figure 3.10 shows a mind-map of the reinforcement learning components and their features. The following sections present a formal description of the characteristics of these components, followed by a discussion of how to build an action policy, how to represent it and how to optimise $\vec{w} = \theta$ (from Section 3.2.2).

3.3.3.1 State Space Dimensions

This section describes the basic characteristics that are associated with the state space of the environment. Each state is characterised by features that can be represented in the form of a table matrix, as described in Section 3.2.1. Through interaction with the environment, a machine learning system observes features of s_{t+1} in the form of input vector $\vec{X}$. In Section 3.2.1 it was explained that the state feature vector has several of characteristics such as being *continuous* or *discrete*. Although many problems (especially benchmark problems) of reinforcement learning have discrete feature spaces, such as Grid-world [van Haaelt 2012], many real-world problems have continuous states, such as helicopter controls [van Haaelt 2012]. In the most challenging cases, it can be very complex for a machine learning system to learn whether the environment is characterised by an infinite or a very large number of states [Gatti 2015]. In this research the number of states is referred to as *complexity* (Section 3.2.1).

3.3.3.2 Action Space Dimensions

This section describes the basic characteristics associated with the action space of the machine learning system. Actions are used by the machine learning system to control its states. In most reinforcement learning problems, the machine learning system consists of a set of actions A, where the size of the action space is *discrete*. However, there are many real-world problems in which the machine learning system includes a large or infinite action space. In such cases, the action space is called *continuous*. Often there is some relation between the action space and the state space, such that continuous state spaces have continuous action spaces and vice versa [Gatti 2015]. The final characteristic of action space is the *branching factor* which refers to the number of actions that can be

taken in any given state. More specifically, some domains may have a constant number of actions that can be performed from all states. By contrast, other domains may have a different number of possible actions in each state [Sutton & Barto 2012] [Gatti 2015].

3.3.3.3 Reward Properties

This section describes the basic characteristics that are associated with the reward space of the environment. Generally, the reward signal guides the machine learning system to achieve a goal that is not explicitly defined. The term *reward* describes any numerical feedback, whether positive or negative, which the machine learning system obtains after visiting a state or performing actions in a state. Although broadly the reward provides an indication of the utility of actions in the immediate sense, it is also used to build a value for each state; rewards are accumulated to show the longer-term benefit, denoted as $V(s)$ where $s \in S$. This *state value* allows the machine learning system to plan for the future before performing any action. The value of the state can be denoted as:

$$V^{\pi}(s_t) = \sum_{i=0}^{\infty} \gamma^i r_{t+i} \tag{3.16}$$

where r_{t+i} is the reward values received after the $i-th$ transition starting from s_t at time t, and $\gamma \in [0, 1]$ is the discount factor, which is a constant that determines the importance of future rewards. Meaning, if γ value is set to 0 then only the immediate reward is considered; but as if the value is set closer to 1, future rewards become more important for the machine learning system. A reinforcement learning system actually requires a policy (π) to use equation 3.16. Further discussion the policy appears in Section 3.3.3.4.

The reward can be *deterministic*, which means each state $s \in S$ has a fixed reward value provided to the machine learning system when it visits the same state at each distinct time t. However, the reward value can be *stochastic* (non-deterministic) when it is a fraction of the time, which is relatively more challenging for the machine learning system. Furthermore, the reward signal can be *distributed* over state space in a different manner. For instance, the mountain car problem has a single reward state which is the

state when the car reaches its goal. By contrast, Tic-tac-toe has multiple states that all provide a reward signal of equal magnitude, because there are many ways to win the game. However, in Backgammon the reward signals are of equal magnitude because there are different types of wins. [Gatti 2015] defined several cases that represent the most common reward distributions and magnanimities:

- Single reward state with a positively valued reward only.
- Two reward states, one state with a positive reward value and the other with a negative reward value, where each has the same magnitude.
- Multiple reward states. each state has either positive or negative reward values, and all have the same magnitude
- Multiple reward states, each state has either positive or negative reward values, but rewards do not have the same magnitude.

Moreover, in most applications, the reward function is *stationary*. This means the rewards distribution does not change over time.

The vast majority of reinforcement learning problems have a stationary reward function that does not change over time with different types of distribution characteristics. However, in this research, the reward function is non-stationary. In addition, it is constructed from multiple values with different magnitudes, which makes this research more challenging.

3.3.3.4 Building Action Selection Policy

The action policy (for short, 'policy', denoted by π) for reinforcement learning defines how the machine learning system behaves in each state $s \in S$ at each discrete time t. More specifically, a policy is a computable function for selecting the best action $a_t \in A$ to be taken by the machine learning system when it is in a state $s_t \in S$, at t; that is, $\pi(s_t) = a_t$. This process is known as mapping each state of the environment to an action. However, often each state might be assigned to a set of admissible actions; thus, the goal of reinforcement learning algorithms is to find the best action for each state, to

maximises the cumulative reward of a machine learning system over several time steps. The result is called the 'optimal policy' (π^*).

Almost all reinforcement learning applications learn the optimal policy by computing the *state value* (Equation 3.16). The state value can be achieved by following a policy π starting from a state s_t as follows:

$$V^{\pi}(s_t) = E_{\pi}\left[\sum_{t=0}^{\infty} \gamma r_t | s_t = s\right] \tag{3.17}$$

This function is called the expected value of state s under policy π, where $0 \leq \gamma < 1$ is a factor used to discount the future reward, and E_{π} is the expectation assuming the machine learning system follows policy π. The goal of the machine learning system is to find the optimal policy that returns the greatest $V(s)$ for a given state s, which is called the 'optimal value function' and is denoted by $V^*(s)$. The best way to do this is by employing the *Bellman optimality equation* [Bertsekas 1987] for each state, which takes the following form:

$$V^*(s_t) = \max_{a_t \in A(s_t)} \sum_{s_t \in S} Tr(s_t, a_t, s_{t+1})\big(R(s_t, a_t, s_{t+1}) + \gamma V^*(s_{t+1})\big) \tag{3.18}$$

The Bellman equation indicates that the value of being in a state (s_t) and performing an action (a_t) under an optimal policy (denoted $\pi^*(s)$) will be equal to the expected return for the best action in that state. The optimal policy function for selecting the best action in a given state takes the following form:

$$\pi^*(s_t) = \arg\max_{a_t \in A(s_t)} \sum_{s_t \in S} Tr(s_t, a_t, s_{t+1})\big(R(s_t, a_t, s_{t+1}) + \gamma V^*(s_{t+1})\big) \tag{3.19}$$

In other words, the optimal value function for each state s ia associated with any

optimal policy π^*, such that $V^{\pi^*}(s) \geq V^{\pi}(s)\ \forall s \in S$ and for all policies π. Keeping track of all state values, as well as finding the optimal policy, therefore requires memory to retain and update each state value, which is called *Representation* (as discussed in Section 3.2.2).

A basic method to perform representation is to explicitly store each state value in a look-up table consisting of the values of each state under different policies. The table can be extended to add actions $a \in A$ to the table columns, so that the value of executing each legal action in each state can be computed. The result is called a state-action pair value $Q(s_t, a_t)$.

$$Q^{\pi}(s_t, a_t) = E_{\pi}\left[\sum_{t=0}^{\infty} \gamma r_t | s_t = s, a_t = a\right] \tag{3.20}$$

However, the look-up table approach is computationally infeasible for a problem with continuous state or/and continuous action. In such cases, a better method is to approximate the state value or the state-action pair value using a parametrised function approximation such as an artificial neural network (Section 3.2.2.2). This research followed that approach. Moreover, the machine learning system requires algorithms to evaluate and update the value of each state. For this purpose, a broad set of algorithms has been introduced to update and learn the $V(s)$ or $Q(s, a)$ within the representation, called here the *optimisation algorithm*. However, since reinforcement learning problems have no corresponding target output (as is the case for supervised learning problems), most learning algorithms in reinforcement learning are built on the basis of the *Temporal Difference error* (TD-error) function. This function predicts the error and allows the machine learning system to evaluate the action and learn the value of the state. The TD-error can be expressed as follows:

$$TE = \left[r_t + \gamma V(s_{t+1})\right] - V(s_t) \tag{3.21}$$

where the quantity inside the brackets can be thought of as the target value; r_t is the

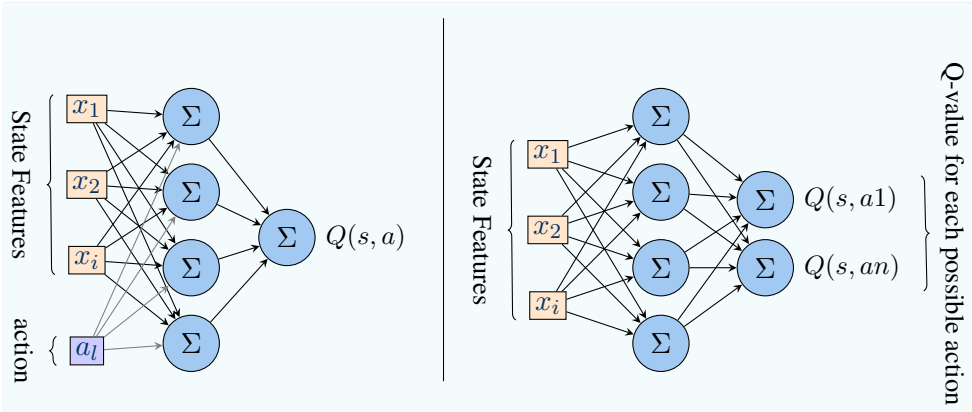


Figure 3.9: Modelling $Q(s,a)$ with artificial neural network, Left panel: Naive formulation of Q-value, where the network takes state features and an action. Right panel: is more optimised formulation of Q-value, where the network only takes state features and input and produces multiple Q-values equal to a number of actions possible in the given state.

reward signal; $V(s_{t+1})$ is the value of the subsequent state and $V(s_t)$ is the value of the current state. Based on this prediction error function, the update function for the $V(s)$ and $Q(s,a)$ can take the following general forms:

$$V(s_t) = V(s_t) + (\alpha \times TE) \tag{3.22}$$

and

$$Q(s_t, a_t) = Q(s_t, a_t) + (\alpha \times TE) \tag{3.23}$$

where α is the learning rate.

There are two approaches to model $Q(s,a)$ through an artificial neural network, as illustrated in Figure 3.9. Firstly, the artificial neural network takes as input the features of a state s_t along with action $a \in A(s)$ available for this state and produces a single output that represents the value of $Q(s,a)$. This process is shown in the left panel of Figure 3.9. However, this approach leads to a practical issue because the policy of the machine learning system is to take an action that maximises the Q-value; that is:

$\pi^*(a|s) = \arg\max Q(s,a)$. Therefore, with this approach, when the machine learning system wants to perform an action it would have to iterate over all actions, then evaluate Q for each one, and take the action that gave the highest Q [Karpathy 2014] [Matiisen 2015]. Alternatively the network could take only the state features as input and produce several Q-values for each possible action, where each was interpreted as the Q-value of taking that action in the given state. This approach has the advantage of choosing the action that fulfils the policy $\pi = \arg\max$ (the action with the highest Q-value). It requires performing only one forward pass through the network and all Q-values for all actions are available immediately. This second approach is the more optimised architecture for formulating Q-values using an artificial neural network model.

Thus far, the thesis has provided a discussion of the most popular methodology to approximate the optimal policy π^*, through the estimation of the optimal value function (V^* and Q^*). This means that rather than estimating π^* directly, V^* and Q^* are estimated. This methodology is often called '*value approximation*'. However, in some of the literature it is called the '*critic-only*' algorithm, where the critic is the approximate value function [Grondman et al. 2012][Heidrich-Meisner et al. 2007][Grondman 2015]. Evidently, the value function (critic-only) methodology assumes that the action space is independent of state space and also consists of a fixed number of discrete actions. Therefore, using this approach in continuous state and action spaces can be non-trivial and time consuming [van Haaelt 2012]

Two other methodologies can be used to estimate the policy. The first estimates the policy directly without estimating the critic. This method is often called the direct policy-search [Ng et al. 1999] or actor-only algorithm [Konda & Tsitsiklis 1999]. Usually, algorithms in this method rely on optimising parametrised policies, usually by gradient descent [Grondman et al. 2012] [Konda & Tsitsiklis 1999]. The advantage of this method is the ability to generate actions in the complete continuous action space [Grondman et al. 2012] [Grondman 2015]. However, a conceivable drawback of this methods is that it might suffer from a large variance in the estimates of the gradient, leading to slow learning [Grondman 2015] [Konda & Tsitsiklis 1999]. Furthermore, the method

does not use temporal-difference learning algorithms, which means that a new gradient is estimated independently of past estimates. For this reason, there is no learning in the sense of accumulation and consolidation of older estimations [Konda & Tsitsiklis 1999].

The second methodology combines the advantages of the critic-only and actor-only methods. It is called the *actor-critic*. The critic uses an approximation architecture to estimate a value function, which is then used to update the actor's policy parameters. Hence, this method can deal with continuous state and action space problems more successfully than the previous methods [van Haaelt 2012]. As the name implies, actor-critic methods rely on two functions: (1) the actor function which is used to implement a stochastic policy that maps state to action, and (2) the value function (described above, Equation 3.22 and Equation 3.23); this function is used to estimate and evaluate the value of each state [Crites & Barto 1994]. Evaluations and updates are based on the TD-error function (Equation 3.21). More specifically, at each time t, the reinforcement learning machine learning system approximates the value of the current state $V(s_t)$ and selects an action a_t based on the actor function. It then computes the TD-error based on the $V(s_{t+1})$ and $V(s_t)$. Finally, the machine learning system updates the actor by adjusting the action probabilities using the TD-error value, and by improving the state-value function using the same TD-error value. The problem in updating the actor function is that if the TD-error value is negative, the action performs relatively poorly and its probability will be decreased. Therefore, actor-critic algorithms usually update their actor only if the TD-error is positive [van Hasselt & Wiering 2007] [Wiering & van Hasselt 2007]. Additionally, actor-critic methods assume the actor function is independent of the critic function and select a single action at each time t step.

3.4 RL with Artificial Neural Network: a Survey

Artificial neural network is a popular and attractive candidate as a function representation in reinforcement learning (RL) applications. It has the capacity to represent complex functions and is able to generalise effectively from a few training examples [Lange et al. 2012] [Whiteson 2012].

As mentioned previously in this chapter (Section 3.3.3.4), actions of machine learning systems can be represented by the artificial neural network in two ways (Figure 3.9). Moreover, the input vector in the artificial neural network corresponds to a state feature, such that the value of these features together describes the state of the machine learning system. In general, traditional reinforcement learning systems assume discrete state and action spaces [Smart & Kaelbling 2000]. However, many real-world problems have continuous states and action spaces; the current study is a good example of that [van Haaelt 2012][Gatti 2015]. This section introduces several approaches to implementing the artificial neural network with reinforcement learning, and demonstrates their action policies. Table 3.1 shows the differences between the approaches (mentioned in this section) of using the artificial neural network to model the $Q - value$ and build the action policy. All these approaches mainly interact with a single environment and perform one action at each time step, selecting from action spaces. The approach in this work differs in that it interacts with multiple environments and therefore performs multiple actions simultaneously.

3.4.1 TD-Gammon

Perhaps one of the oldest and best-known successful implementations of reinforcement learning with an artificial neural network is TD-Gammon [Tesauro 1992]. TD-gammon used an artificial neural network as a position evaluator for a Backgammon game. The output of the artificial network was an evaluation of the game position (based on an encoding of the game position), which was fed to the network as an input vector. However, the artificial network did not consider the possibility of doubling; that was handled by a separate heuristic code. After each forward pass step, the weight connection of the artificial network was updated with a back-propagation algorithm after computing the TD error of the output. The reinforcement learning system was trained by self-play, without explicit exploration [Szita 2012].

The artificial neural network was used to approximate the state value function $V(s)$ only, rather than the action-value function $Q(s, a)$. Also, it used a direct policy research (i.e. actor only) to learn the policy of action from the self-play games [Mnih et al.

2013]. By contrast, our approach applies reinforcement learning system requirements to perform multiple continuous actions at the same time, with each action conflicting with all other actions.

3.4.2 Neural Fitted Q Iteration (NFQ)

Neural fitted Q (NFQ) iteration is an algorithm for efficient and effective training of a $Q - value$ function, represented by a multiple layer perceptron [Riedmiller 2005]. The NFQ is a batch reinforcement learning method [Lange et al. 2012]. This approach overcomes the problem of needing many training samples, collected and stored as a sequence of samples from the environment, for every learning iteration in order to re-use them to update the value function simultaneously at all transitions. This method affects the other state value function examined so far.

However, NFQ uses a batch update that has a computational cost per iteration, which corresponds in size to the size of the dataset [Mnih et al. 2013]. Moreover, NFQ assumes that the action space is finite or restricted within a limited interval. Also, it was designed to perform one action at a time.

3.4.3 Deep Reinforcement Learning

Deep reinforcement learning combines deep neural networks with reinforcement learning. Perhaps the first work that showed impressive results with this approach was that of [Mnih et al. 2013] and more recently [Mnih et al. 2015]; both studies were performed by a team of DeepMind Technologies – which became Google DeepMind in 2014. Their work was focused on learning control policies (actions) from the heigh-dimensional sensor, using a combination of a Q-learning algorithm with a deep neural network. Their approach is distinguished from NFQ by its use of stochastic gradient updates, which are not considered in NFQ. The advantage of using stochastic gradient updates is they have a low constant cost per iteration and can scale to large datasets [Mnih et al. 2013] . The researchers then used this approach to train two artificial neural networks, one for policy and anther for value function, to allow the reinforcement learning system to learn a GO game, which they called AlphaGO [Silver et al. 2016]. However, this approach suffers

Table 3.1: Comparison among approaches to implementing artificial neural networks with reinforcement learning

Framework	Number of environments		State space		Action space		Action policy	Number of actions to be performed at once
	Single	Multiple	Discrete	Continuous	Discrete	Continuous		
NFQ [Tesauro 1992]	✓	x	✓	✓	✓	–	$\pi = argmax(Q(s,a))$	1
Playing Atari [Mnih et al. 2013]	✓	x	✓	✓	✓	✓	$\pi = argmax(Q(s,a))$	1
AlphaGO [Silver et al. 2016]	✓	x	✓	✓	✓	✓	$\pi = p(a\|s)$	1
Our Approach	x	✓	✓	✓	x	✓	$\pi = \frac{(V(s))_k)_t}{\sum_{\forall V(s)}(V(s))}$	multiple

from substantial overestimations in some games in the Atari 2600 domain [Van Hasselt et al. 2016]. To overcome this problem, [Van Hasselt et al. 2016] introduced the Double Q-learning algorithm, which can be generalised to work with large-scale function approximation. Although the deep reinforcement learning approach is fairly similar to our approach and it can work with continuous state and action spaces, our approach is designed to deal with multiple continuous actions spaces.

3.5 Summary

Machine learning algorithms are suitable for solving problems in a complex and changing environment, such as cloud computing. This chapter provided an overview of machine learning algorithms, including two main branches: supervised learning and reinforcement learning. In addition, this chapter reviewed the use of machine learning to predict an access pattern as it relates to this work. In the next chapter, a survey of different results using machine learning algorithms to solve problems in a cloud computing environment is presented.

Figure 3.10: A mind-map of a reinforcement learning framework. Reinforcement learning has four main components that interact with each other to solve sequential decision problems: environment, action space, reward function, and action selection policy.

CHAPTER 4

Machine Learning Applications in Cloud Computing

The second chapter in this thesis discussed cloud computing services, including the benefits and risks of using these services. That chapter was focused on cloud storage services and provided a detailed discussion about the issues of storing data using a single cloud storage service. The chapter then provided a review of research that has aimed to overcome the problems related to single cloud storage services.

Due to the complexity and dynamism of the cloud computing environment, machine learning algorithms offer a solution for many optimisation problems in the cloud. The previous chapter discussed machine learning algorithms, in particular the supervised learning and reinforcement learning methods.

This chapter surveys the researcher's proposed method to solve problems in the cloud computing environment. Since cloud computing emerged in the market, researchers have been addressing problems related mainly to resource management and energy efficiency. The main goal of this chapter is to provide insight into various problems for which the literature proposes solutions, based on machine learning algorithms, and to distinguish this work from other research.

4.1 Resource Allocation Management

'Computational resources' refers to all hardware, whether physical or virtual, and software connected to a computer system. This is different from the objective of this work, which focuses on allocating files to remote storage based on file access patterns.

Cloud providers, especially those that provide IaaS, use virtualisation technologies to partition physical computational resources into autonomous virtual resources. The physical resources include for example servers, central processing units (CPUs), memories, storage, and network band-width. The autonomous virtual resources are often called virtual machines (VMs). This technology enables cloud applications to scale resources dynamically. Management of virtual resources in a large environment – such as the cloud – presents challenges for resource planning and application management. These challenges are associated with co-located virtual machines. Considerable effort has been made by researchers to develop solutions to tackle these issues and control the scaling of virtual resources. Since this research only considers solutions that include machine learning techniques, other solutions used to allocate computational resources are not discussed. The following list present works the provided solutions for resource allocation management based on machine learning algorithms.

- Choi et al. [2008] • addressed the problem of imbalance in the migration of VMs across all physical machines, which sometimes causes an overload in machines. To solve this problem, the authors presented a learning framework that autonomously adapts to changes in VM migration and therefore uses resources effectively and efficiently. However, they did not mention precisely what type of learning algorithm was used.

- Rao et al. [2009] presented a reinforcement learning based algorithm, namely VCONF. This algorithm is capable of auto-configuration of VMs to adapt to changes in demand for applications. In general, the solution aims only to control performance without considering the influences on the cloud cost.

- Dutreilh et al. [2011] proposed a reinforcement learning approach to automatically add and remove resources, based on the variable workload model. The aim of this work

was to self-adapt several resources allocated to applications in cloud environments. The rewards were formed from the cost and the penalties imposed when the target performance was violated. However, there was no balance between cost and performance in this work. Furthermore, the authors did not consider the growth of cost over the billing time. In addition, the cost function was based on the use of VMs, without thinking about network bandwidth.

- Bu et al. [2011] proposed a coordinated auto-configuration framework called CoTuner, to automatically adjust virtual resource allocation and application parameters to optimise application performance. The heart of this framework is a combination of Simplex and reinforcement learning methods. The Simplex method was used to reduce the search space and avoid performance degradation caused by random exploration. The aim of this framework is to adjust the configurations of the VMs dynamically, in response to changes in workload. The authors claimed the proposed framework does not require pre-learned performance models, and is suitable for highly complex and dynamic systems. The reward function in the reinforcement learning system is designed to reflect the overall system performance of VM applications. More specifically, the reward is constructed from the throughput, the response time, and the penalty for a service level agreement (SLA [1]) violation. However, like many others, this work lacks embedding of cloud cost optimisation.

- [Kundu et al. 2012] studied the impact of configurations of VM resource allocations on the application performance. The authors identified the three main parameters that exert the most influence on the performance of virtual applications. Based on the results, they introduced a solution that relies on an artificial neural network and support VM, to model the performance of a VM-hosted application to improve resource allocation management. They claimed this solution helps both users and cloud service providers. However, the study and the proposed solution focus mainly on performance and do not consider the effect of the solution on the cloud cost.

- [Vasić et al. 2012] proposed a framework called DejaVu, which uses supervised learn-

[1] SLA is an agreement or contract between a service provider (here cloud provider) and the cloud consumer that defines the level of service expected from the cloud provider.

ing techniques to learn from experience how to automatically react to workload changes in virtual resource allocation. The authors used an off-the-shelf classifiers technique that operates on workload clusters, which are determined after an initial learning phase. They claimed that this technique achieved a positive result in reducing the overall resource management effort and overheads.

- [Xu et al. 2012] introduced a unified reinforcement learning (URL) approach to auto-configure VM processes and appliances running on virtual machines. The approach comprised two reinforcement-learning learner machines. The first machine, App-Agent, tuned the application parameter settings; the second machine, VM-Agent, adjusted the VM configurations online. This work focused only on enhancing the response time of applications in the cloud, without paying any attention to the cloud cost.

- [Chen & Bahsoon 2013] developed a self-adaptive and sensitivity-aware Quality of Service (QoS) modelling approach. The approach consists of symmetric uncertainty, with two machine learning techniques: auto-regressive moving average with eXogenous inputs model (ARMAX), and a neural network. The approach yielded two formulations of the QoS model. Mainly, the aims for this approach were to identify the primitives correlated with QoS at run-time to capture the dynamics of QoS sensitivity.

- [Barrett et al. 2013] - introduced a parallel reinforcement learning agent approach to optimise resource allocation in a cloud environment. Each agent observes an insular environment state space (from the entire environment), learns an individually optimal policy, performs a different action, and obtains a different reward. The agents then communicate with each other and exchange information regarding their observations while operating in the environment. Each agent can choose an action from a discrete action space (add, remove or maintain the numbers of VMs allocated to the application). Rewards are determined based on a combination of the cost of resources and any associated penalties that might apply as a result of violating the specified SLA, which is related to the response time (measured in milliseconds). Although this approach minimised cost and response time, the researchers did not consider some of the factors that affect the cost, such as network bandwidth. Furthermore, the cost of cloud computing is a cu-

mulative number within the billing period; this means that the cost value of the reward function increases each time the user uses the same VM. Moreover, this work implicitly assumes that the reward parameters (the cost of acquiring a resource and the response time) have equal importance at all times. This means there is no trade-off between cost and response time based on differences in demands or tasks.

- [Jamshidi et al. 2015] developed an online learning mechanism called FQL4KE, which is a combination of fuzzy control and Fuzzy Q-Learning algorithms. Fuzzy control facilitates logic at a higher level of abstraction (human logic), and the Q-learning allows the application to be adjusted according to demand by automatically scaling the computer resources at run-time. The objective of this combination is to connect human expertise to a continuous evolution mechanism. The reward function for Q-learning was defined based on SLA violations, the amount of resource acquired (VMs), and throughput. Although the reward parameters have corresponding weights that determine their relative importance in the reward function, it requires a user to determine the value of these weights based on their goal. Additionally, the authors implicitly assumed that the pricing scheme was fixed, because their solution was designed for only one cloud service.

4.2 Energy Efficiency

Many interconnected factors at different levels of a computing system influence energy consumption in a data centre. These factors include hardware efficiency, the resource management system, the effectiveness of the application running on the system, power distribution, and thermal load and cooling systems.[Beloglazov et al. 2011] [Chen et al. 2011].

Cloud computing resources are generally driven by demand application of cloud users, whose numbers are rapidly growing. This growth has led to an overwhelming increase in the energy bill and carbon dioxide footprint [Beloglazov et al. 2011] [Salimian & Safi 2013]. However, limiting the energy use may affect users in term of service performance and quality of service (QoS). These issues have received considerable research attention and many solutions have been proposed to optimise the consumption of energy

within the cloud computing infrastructure [Demirci 2015]. Cloud computing sometimes uses machine learning, and this topic is discussed next. The literature in this field is reviewed below.

- [Cioara et al. 2011] addressed the problem of dynamic server consolidation in virtual service centres, such as cloud infrastructures, by presenting an energy aware run-time consolidation algorithm based on reinforcement learning. The aim of that research was to minimise energy consumption in such an environment. However, the researchers did not consider the effect of the research on the performance of services.

- [Prevost et al. 2011] introduced a framework for the prediction of future load demand in cloud resources. The aim of this framework was to optimise cloud resources to reduce energy consumption without adversely affecting all existing SLAs. In other words, this work did not consider any enhancement of the performance of services.

- Dabbagh et al. [2014] introduced a method to control energy consumption. The authors developed a framework that used stochastic theory to predict future VM requests, and an unsupervised learning algorithm to cluster the VM requests into categories. Based on this, the framework could decide when to switch the physical machines into sleep mode to save energy. However, the researchers did not study the effect of their framework on the performance of services.

4.3 Summary

This chapter summarised the considerable research in the field of machine learning algorithms in the cloud computing environment. Most solutions that have been proposed to solve various problems with cloud infrastructure have focused on the VM layer, either to improve performance or to minimise costs and reduce energy consumption. However, none of the studies reviewed in this chapter have incorporated file usage predictions (file access patterns). Moreover, all have dealt only with a single cloud environment that has a sole pricing scheme. To our knowledge, no studies have used machine learning algorithms to distribute data across both single and multiple cloud storage. Therefore,

this study might be the first to introduce machine learning techniques to overcome issues related to distributing data across multiple clouds.

CHAPTER 5

System Architecture and Emulator

The previous chapters provided an overview of cloud computing and addressed concerns about storing data in a single cloud storage service. A review of the literature related to these concerns was provided, as was an overview of machine learning paradigms and the application of machine learning to cloud computing. This chapter describes the architecture of the framework for file distribution proposed in this thesis. A description of the cloud storage emulator used to evaluate this framework is also provided.

5.1 OFDAMCSS Framework Architecture

The aim of this work is to use the power of machine learning algorithms to distribute and optimise files across multiple cloud services. This chapter and those that follow propose a novel framework, named Optimisation of File Distribution Across Multiple Cloud Storage Services (OFDAMCSS). The framework allows for distribution of data across multiple cloud storage services from the consumer side, based on file access patterns. The main objective of the framework is to optimise both cost and performance factors. The total cost, as considered in this research, includes storage, network band-

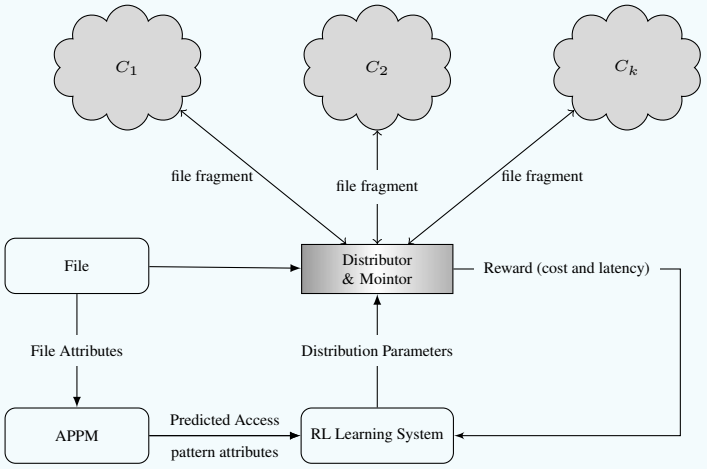


Figure 5.1: A high-level view of the Distribution Framework Structure, where APPM estimates how many times each file will be accessed in the future and its expected lifetime; the reinforcement learning system tunes the distribution parameters (i.e., the proportion of each file that will be located in each cloud); the distributor manages distribution, taking distribution policy from the reinforcement learning system ; C_1,C_2, and C_K cloud storage services where $k \in \mathbb{N}$.

width, and operations. The performance factor relates to transferring files between the cloud consumer side and the cloud provider side (latency time). The distribution of data across multiple cloud storage services will improve data availability and service continuity, while avoiding the risk of vendor lock-in. The OFDAMCSS, illustrated in Figure 5.1, comprises two machine learning methods:

1. **Supervised learning** Supervised learning, which predicts the access pattern for each file, is here called the access pattern prediction model (APPM). It uses linear regression to predict the access pattern for each file and is discussed in detail in Chapter 6.

2. **Reinforcement learning** 1. which decides how to distribute files across multiple clouds. This decision is based on the variance in cloud storage services' pricing schemes and performance, as well as file access patterns. The reinforcement learning system was trained with an artificial neural network to optimise cost and per-

formance of cloud storage services over the long term. This learning system has a new method to transfer the value of each state into an action. The method was specifically designed for this research and is suitable for distribution tasks only. Further details are provided in Chapter 7.

The APPM reads the file attributes and predicts the file access behaviour as well as the file's lifetime. The APPM then passes the file access pattern attributes to the reinforcement learning system, which uses an artificial neural network to manage the distribution policies based on the principles of RAID technology (Section 2.5). The distributor uses the outputs of reinforcement learning to distribute each file (each file has different size) over multiple cloud storage services. Here, the distributor aims to locate a different proportion of each file on each cloud storage service.

Because of the high cost and time that would have been required to examine this framework on real cloud storage services, a cloud storage emulator was built to evaluate the framework. In the next section, the emulator is discussed in detail.

5.2 Cloud Storage Emulator

The emulator was built in JAVA and was designed to emulate the performance and costs of cloud storage services. It was a block box emulator that received the file size and returned the latency time. The latency time is the time required to upload (write) and download (read) the file to or from each cloud service. In addition, the emulator calculated the total cost of using the storage and the network bandwidth of each cloud service. It was flexible and capable of emulating any number of cloud storage services simultaneously. The performance of the providers in this research was set up to simulate the real performance of Google Cloud Storage, Amazon S3, Microsoft Azure Storage and RackSpace Cloud File; these services were measured using the performance analysis services of cloudharmony.com. For each cloud storage service, the fastest and slowest performance speeds were measured. The mean value was calculated and was assigned to the cloud in the emulator. Table 5.1 shows the highest and lowest values measured for each cloud storage service by cloudharmony.com.

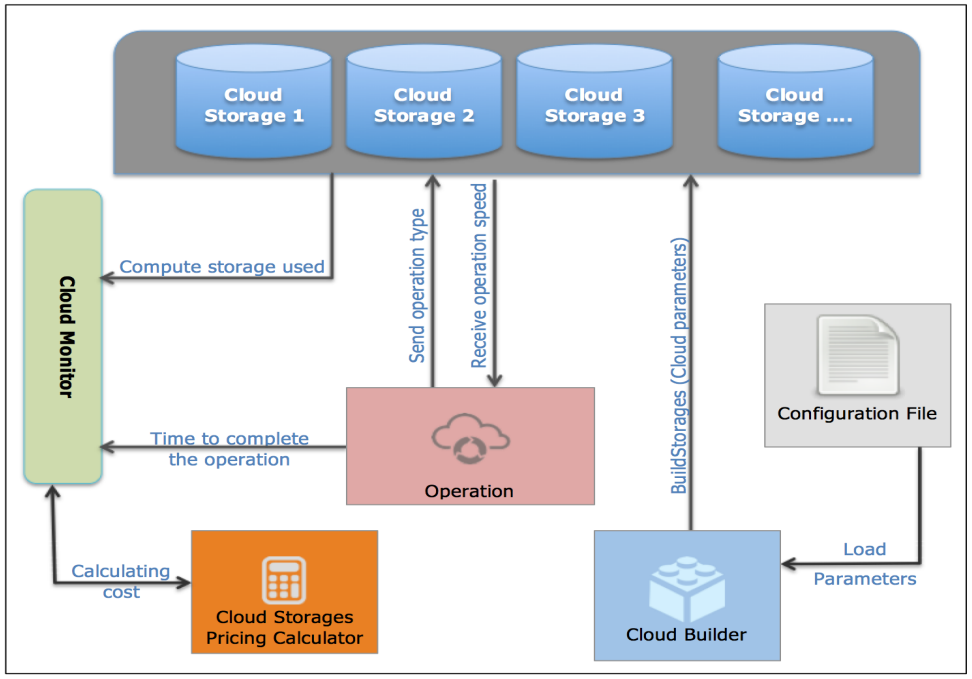


Figure 5.2: Cloud storage emulator: the architecture of classes. The operation class is responsible for writing and reading from the storage. After each operation, the emulator calculates the latency time based on the speed of each cloud provider and the total cost based on each cloud provider's pricing scheme. The builder class is responsible for building the storage services based on the configuration file that contains all storage attributes

As shown in Figure 5.2, the cloud storage emulator consists of five classes: cloud storage containers, cloud builder, operation, cloud storage pricing calculator, and cloud monitor. First, the cloud builder class loads the cloud parameters from a configuration-file, which consists of numbers of cloud storage services required to be built and the minimum and maximum transfer speed for download and upload into each cloud. The operations (read, write and delete) are managed by the operation class. The operation class receives the order of upload (write), download (read), or delete from the distributor system and then computes the time needed to complete the operation. The cloud monitor class will observe the speed and cost for each cloud and send the average result to the reinforcement learning system (agent).

Table 5.1: Performance range of cloud storage services (measurements by cloudharmony.com)

Cloud Provider	Fastest speed MB/s	Slowest speed MB/s	Average MB/s
Google Cloud Storage	20.59	17.55	19.07
Amazon S3	25.29	6.56	15.92
MS Azure Storage	21.08	16.04	18.56
RackSpace Cloud File	19.55	13.62	16.58

5.3 Summary

This chapter focused on the architecture of the framework and how it works. It also included a description of the emulator system used to perform experiments to evaluate the framework. The following chapters cover these components in detail, as well as their methodologies and how they work and cooperate in the framework.

CHAPTER 6

File Access Pattern Prediction

File access pattern attributes (read, write, lifespan) are the main factors that influence the cost of cloud storage services. In Chapter 2 it was explained that the cost of cloud storage is calculated using three factors: (1) storage cost, which is affected by the lifespan of the file; (2) network cost, which is affected by the read and write behaviours of each file; and (3) the number of operations (e.g. read, write, delete), which are influenced by access behaviours. Therefore, if the access pattern can be predicted, the cost in the future can also be predicted.

This chapter begins with a brief review of the use of machine learning to predict access patterns. This is followed by details of how the training data were built and how the prediction model was built. Finally, an evaluation of the prediction model is presented.

6.1 Prediction of File Access Pattern: A Review

This research is based on the pattern of file access. The literature on using machine learning to predict file access patterns is reviewed here; however, few works have been

published in this field. Only a few researchers have used machine learning algorithms to predict files access patterns. For instance, [Vengerov 2008] presented a reinforcement-learning-based framework for redistributing files over multi-tier storage systems, based on their recent classification of access pattern. They classified file access patterns into two groups: (1) hot, when the file is accessed frequently; and (2) the opposite group called cold. In other research, [Mesnier et al. 2004] used decision trees to predict the access pattern and lifespan of new files and classified them into read-only and write-only. Based on that classification the system can automatically assign storage policies for each individual file, instead of assigning one policy for all files. For instance, read-only files will be aggressively replicated onto several hard disks to provide opportunities for load spreading. By contrast, write-only files will be stored in an LFS partition. The goal of the work was to adapt to the overall workload and performance changes in the storage system in one data centre. Moreover, demonstrated a strong association between a file's access pattern and its names and attributes, such as date created, file's owner, and file extension type.

Generally, the above studies were based on the classification of access patterns. However, this kind of classification is not suitable for cloud computing storage services, because the price of such services depends on the extent of file access. Hence, this work employs a numerical prediction of how many times each file will be read and written to (updated) during its lifespan.

6.2 Trace History Files: Collections and Structure

This section provides details of how the dataset used to evaluate the approach taken in this research was obtained. Initially, several organisations such as the University of York, the Saudi Fund for Development (SFD), RackSpace, IBM, Microsoft, and Lepide were contacted to request real-world data on file access logs. Unfortunately, none of those organisations had activated file trace systems because of the load that would create on the file-store system. Several researchers were also contacted about the data used in their research, including [Mesnier et al. 2004], [Liu et al. 2013] and [Agrawal et al. 2007]. I

found some datasets, but it was for a short period (2 to 5 days) and was used for studying the behaviour of each user for different files. However, what this research required was data on the behaviour of all users for each single file. In addition, the data did not contain clear information about the number of reads, updates, and lifetimes.

Due to the lack of available real-world log files, an access file log generator was created to generate a synthetic dataset of access pattern log files, based on a workflow of real systems in SFD. In fact, in this research, generating data synthetically proved to be more useful than real-world data for several reasons. For example, because the parameters that control the generation process were controlled, it was possible to generate a wide range of datasets to demonstrate the generality of the system. Furthermore, synthetic data can be a playground for the parameter space to meet some special condition that cannot be found in real data. (More information about the benefits of using synthetic data, especially with machine learning algorithms, is available in [Nonnemaker 2008] and [Eggert et al. 2015]). In addition, our synthetic data were realistic because they were based on the workflows of real systems. Examples of these systems included vacation, payroll, research, and annual budget systems. This research focused only on how long the file would be active and how many times it would be accessed (number of reads and writes).

This section provides use-case diagrams for all systems mentioned above, to give an overview of how these systems work and how the access pattern logs were generated. Below is a selection of examples of the workflows for these systems.

- **Vacation Application System:** This system is one of the core elements in any human resources (HR) system. Figure 6.1 shows a general use-case diagram of this system, including users and their tasks, which affected the access pattern determined in this research. There are several types of vacation, but all of them have the same workflow:

  1. *User:* creates a vacation application, then submits it to the section manager.
  2. *Section manager:* receives applications from the team, then annotates them before sending them to the department manager.

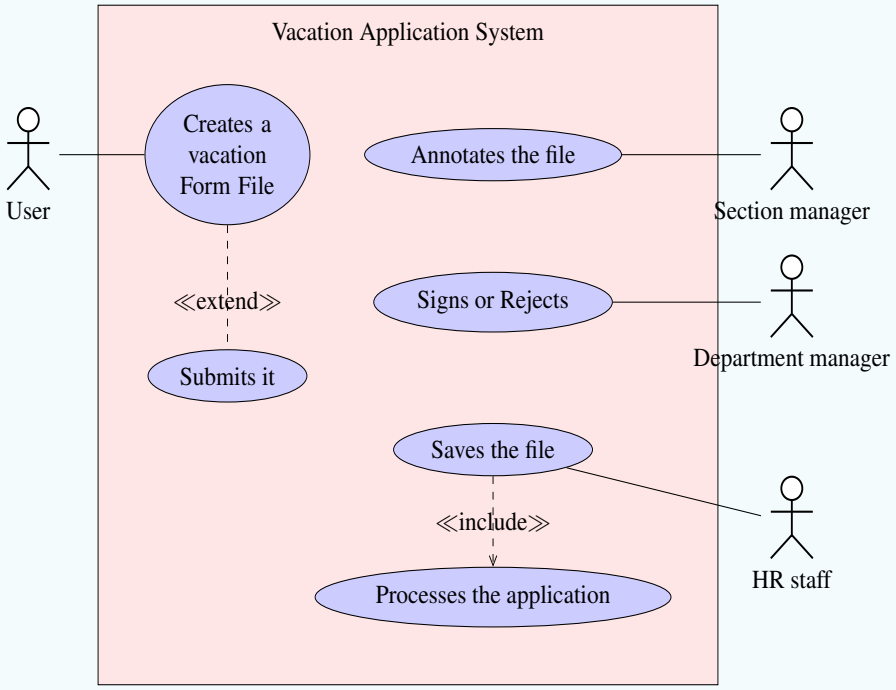


Figure 6.1: Usually in SFD, the user creates a vacation file, fills it, then sends it to the section manager. The manager annotates it and passes it to the department manager to sign. Finally, the file is passed to HR staff to process and save

3. *Department manager:* He or she decides about the application and either approves and signs it or rejects it. If the application has been approved, it is sent directly to the HR department.

4. *HR staff:* They check the regulations and the process for the application before saving the file.

Here, the vacation system is not only related to annual leave but refers to all types of leave, such as sick leave, study leave, and exceptional leave. In general, the time taken to process an application is two to four days. The file size is usually between 50 KB and 70 KB.

- **Payroll system workflow:** This is the process by which the organisation prints sheets of employees' salaries, with each department on a different sheet, to be

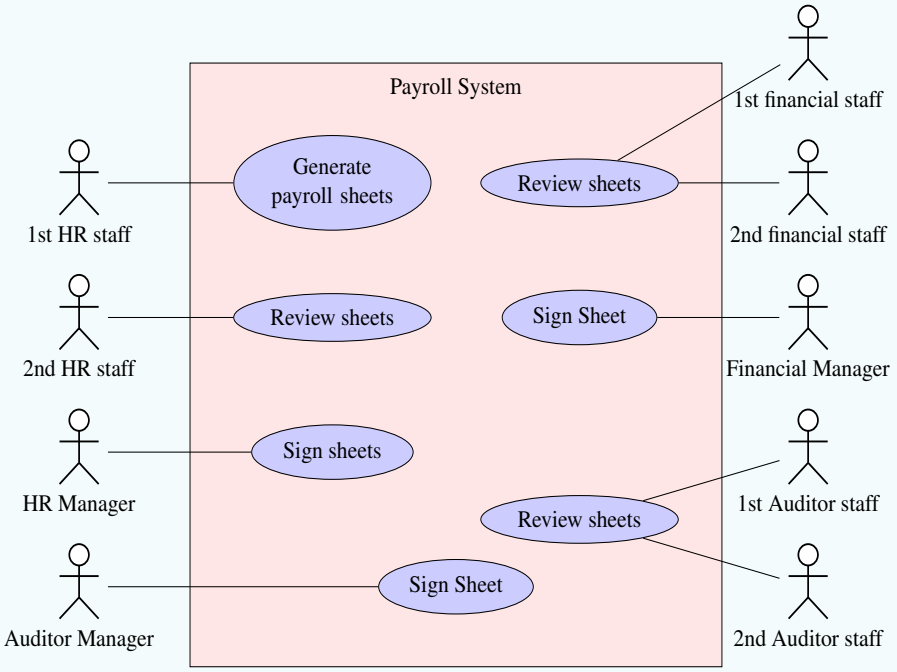


Figure 6.2: The payment roll system is used to generate sheets of employees' salaries. Usually, sheets are generated as drafts at the start of a month. They then pass through various processes, administered by users in different departments, before being sent to the bank.

sent to the bank. As illustrated in 6.2, the process of generating payrolls is fairly lengthy because it involves three departments, each of which has different reviewers. The following is an outline of this system's workflow:

1. At the beginning of each month, first HR staff create a payroll sheet (first HR staff refers to any employee in HR who has access to this system).
2. The second HR staff review the sheets and sign them (second HR staff refers to any HR employee).
3. HR department manager: He or she signs the sheets before sending them to the finance department.
4. In finance department: Two staff members review the sheets and check if the total amount of all salaries is available in the account, before sending the

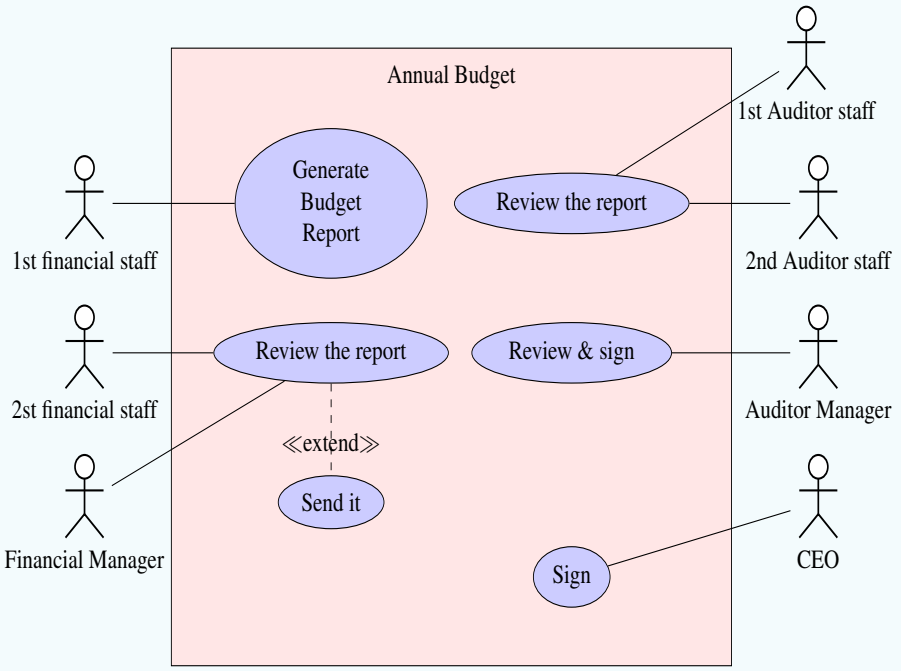


Figure 6.3: The annual budget system is used to create an annual report on expenses in the past year and expectations of expenses in the coming year

sheets to the auditing department.

5. In auditing department: Two reviews take place before the manager of the department signs the sheets and sends them to the bank; the bank then transfers the salaries to employees' accounts.

Monthly, a different sheet for each department is generated, containing all employees' names and salaries. The process of this system starts at the beginning of each month and usually ends on the 15th of each month. The file size of each sheet is about 105 KB to 145 KB.

- **Annual budget system workflow:**Based on this system, staff from the finance department generate an annual report of all expenses in the past year, including salaries. This means all payroll sheets for the last 12 months will be re-opened. Figure 6.3 gives an overview of the use-case diagram that shows a number of users

of this system and their effect on the documents. This system's workflow can be outlined as follows.

1. At the end of each year, first financial staff create a comprehensive report on all expenses. (First financial staff refers to any employee in the finance department who has access to this system).
2. Second financial staff: This person reviews the sheet and signs it. (Second financial staff refers to any employee in the finance department.
3. Finance department manager: This manager also reviews the sheet and signs it, then sends it to the auditing department.
4. At the auditing department, two staff must review the report before the manager of the auditing department reviews it and signs it. The report will then be sent to the CEO for approval.

The file size of this report is about 85 KB to 90 KB, and the process takes about six to nine days from being generated until the CEO approves it.

- **Research department system workflow:** This system helps in creating research documents about developing countries, to be used for studies. It is a small but active system. Figure 6.4 gives an overview of the use-case diagram for the process of creating and submitting research files. The following is the outline of this system's workflow:

  1. At any time of the year, a researcher creates a report about a certain country.
  2. The research manager reviews the report and sends it to the board of directors.
  3. The board of directors may read this report several times.

  The file size of this report is about 455 KB to 490 KB, and the lifetime of each file is between five and 17 days.

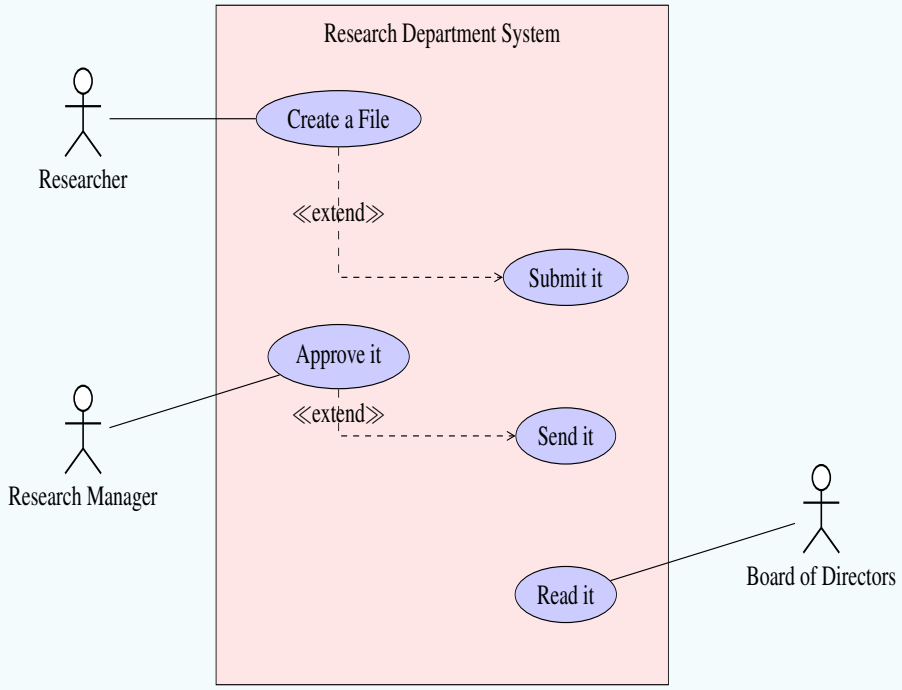


Figure 6.4: Research department system for generating research reports

6.3 Synthetic Data Generator

The generator system was written in JAVA to generate the datasets that emulate the behaviour of users on each file (e.g. read, update). A class was created for each of the above systems to generate several file attributes: file size, number of times read, number of times written to, the user, the creator of the file (file owner), date created, user department, and user level (ordinary user, head of section, manager of department, or general manager). Each class generated random variables for each attribute, uniformly distributed over the intervals shown in Table 6.2. For example, the payroll class generated one sheet for each department monthly for one year. The size of each sheet (file size) was randomly generated – with uniform distribution – from the range for each sheet, based on how many employees work in each department. The departments were IT, HR, finance, auditing, communications, engineering, and research. Similarly, the other attributes of the file (number of reads and writes and its lifetime) were uniformly distributed in a

Table 6.1: Attributes of the synthetic datasets

	Num. of Instances	Min. file size	Max. file size	Toltal size in GB
WS-dataset 1	947	42 KB	490 KB	267.43
WS-dataset 2	2092	35 KB	490 KB	193.21
WS-dataset 3	2075	36 KB	490 KB	221.33
R-dataset 1	2116	16 KB	515 KB	172554

range [vmin,vmax], as shown in Table6.2. All other system files were generated in the same manner. For the annual budget and payroll systems, the generator allowed users to add a cover page, created as separate file with a size of between 35 KB and 50 KB.

Three synthetic datasets were generated based on the workflow systems mentioned in the preceding section (denoted here as WS-dataset). One synthetic dataset (denoted here as R-dataset) was generated randomly for unspecified workflow systems, based on the random system row in Table 6.2;this dataset was used to test the generality of the data. Table 6.1 shows the number of instances in each dataset. The total files in all datasets was more than 9154 files, with a total size of more than 854 GB.

Table 6.2: Attribute variables from which the generator could select values

SYSTEMS	# File range	File Size range	NR range	NW range	Lifetime range	File Types	Owner Department	Owner Position level
Vacation applications	[300,450]	[50,70] KB	NW + [1,2]	3	[2,4] days	”docx”	HR	O.user , H.section , D.manager
Research Documents	[3,15]	[485,490] kb	NW + [5,7]	[1,3]	[5,17] days	”docx”	Research	O.user , H.section , D.manager
Annual Budget	1	[120,145] KB	NW + [5,7]	[6,9]	[10,20] days	”xslx”	Financial	O.user , H.section , D.manager
PayRoll sheets	111	[220,500] kb	NW+[2,4]	9	[6,10] days	”pdf”	HR	O.user , H.section , D.manager
Random System	[500,2500]	[15,600] kb	NW+[1,10]	[1,10]	[1,20] days	”pdf”	HR	O.user , H.section , D.manager

* # File range, number of files to be generated; NR range, number of access readings; NW, number of access writings; owner department, department of user who created the file; owner position level, pay grade of user who created file (O.user = ordinary user; H.Section = head of section; D.manager = department manager or director)

6.4 Access Pattern Predictive Model

The file access pattern describes the file's behaviour throughout its lifetime. More specifically, the pattern describes how many times the file will be read or updated throughout its lifespan. Understanding the behaviour of a file is vital for optimising the cost in cloud storage services.

Unfortunately, few researchers use machine learning algorithms to predict file access patterns. As summarised in Section 6.1, [Vengerov 2008] used a reinforcement learning algorithm for redistributing files over a multi-tier storage system, based on the recent classification of the access pattern (i.e. hot or cold files). [Mesnier et al. 2004] used decision trees to predict the access pattern and lifespan of new files and to classify them into read-only or write-only. However, using a classification mechanism to predict the access pattern of files in cloud computing storage is infeasible. As discussed in Section 2.4, the cost of storage services for most cloud computing is calculated based on (1) the amount of data stored, (2) the amount of network bandwidth used, and (3) the number of operations (e.g. read, update, delete). Therefore, optimising the cost of cloud storage services necessitates a prediction of 'numerical' access patterns of files, together with their lifetime.

The goal of the access pattern predictive model (APPM) is to find correlations between the proprieties of a file and its access pattern values. The values are as follows:

1. **Lifetime**: refers to the number of days the file is kept active. Although the general meaning of lifetime or lifespan is the period between creating and deleting a file, in this work only the active-lifetime of each file was of interest.

2. **Number of read**: refers to the number of times the file will be read throughout its lifetime.

3. **Number of write**: refers to the number of times the file will be updated during its lifetime.

The APPM was trained on the synthetic datasets for several department systems in a

large organisation (SFD, www.sfd.gov.sa). The datasets were generated based on the file workflow of vacation, payroll, financial reporting, research and budgeting systems, as described in the preceding section. The log file consisted of the following file attributes: the departmental owner of the file, the file owner's position, the date on which the file was created, the file size, the file extension (docx, xlsx, or pdf), and the file category (general letter, finance, vacation etc.).

6.5 Prediction Model Evaluation

To evaluate the quality of the APPM predictions, three utility functions were employed to measure the regression performance. These functions were:

- Correlation coefficient, denoted by r (Pearson's r).
- Root mean squared error, denoted by $RMSE$.
- Mean absolute error, denoted by MAE.

Tables 6.3, 6.4 and 6.5 show the evaluation results for WS-Dataset1, WS-Dataset2 and WS-Dataset3 respectively. The datasets in all tables display a correlation coefficient close to +1, which indicates a very strong linear pattern in the datasets. In addition, the results for MAE and RMSE imply a high accuracy of future prediction.

The other two datasets (R-Dataset1 and R-Dataset2) did not differ substantially from the above three datasets. The results for these two datasets for the undefined workflow system are shown in 6.6 and 6.7. Correlation coefficients values within the two datasets were all above 0.85, which again indicates strong linear relationship. were higher than the means of the WS-datasets. The differences between the R-datasets and WS-datasets might be explained by the fact that the R-datasets were built randomly and not on certain and clear patterns.

6.6 APPM System Overheads

The APPM experiments were conducted on MacBook Pro with processor 2.4 GHz Intel Core i7, memory 8 GB, and 1600 MHz DDR3. The generation code of the synthetic

Table 6.3: APPM, measurement evaluation for WS-dataset 1

	Lifetime	Number of read	Number of write
r	0.9761	0.9228	0.9889
MAE	0.9621	0.5648	0.0418
$RMSE$	1.3025	1.143	0.2119

Table 6.4: APPM, measurement evaluation for WS-dataset 2

	Lifetime	Number of read	Number of write
r	0.9758	0.9085	0.9787
MAE	0.9843	0.6107	0.0774
$RMSE$	1.335	1.2154	0.3024

Table 6.5: APPM, measurement evaluation for WS-dataset 3

	Lifetime	Number of read	Number of write
r	0.9752	0.9155	0.9852
MAE	1.0133	0.6323	0.0617
$RMSE$	1.3674	1.2245	0.2526

Table 6.6: APPM, measurement evaluation for R-dataset 1

	Lifetime	Number of read	Number of write
r	0.8517	0.8726	0.8644
MAE	2.0049	0.9689	0.5084
$RMSE$	2.7346	1.8352	1.3956

Table 6.7: APPM, measurement evaluation for R-dataset 2

	Lifetime	Number of read	Number of write
r	0.8728	0.8726	0.8525
MAE	1.9521	0.966	0.4082
$RMSE$	2.6531	1.835	1.3479

datasets was written in Java using IntelliJ IDEA CE. The generation of the datasets took a few minutes. Thereafter, the datasets were fed to the WEKA system, which is a collection of machine learning algorithms for data mining tasks. The task for WEKA was to generate three regression models: one for reading, one for writing, and one for the lifetime. The WEKA system took a few minutes to complete this task.

6.7 Summary

This work has shown that linear regression models are an accurate and adaptable mechanism for capturing associations between file attributes and file access pattern values. The access pattern values can be used by a reinforcement learning system (accompanied by a variation in cloud performance) to learn how to optimise cost and latency time in different cloud storage services. Before discussing how reinforcement learning works, the role of artificial neural networks in reinforcement learning system is explained. This is a complex topic that requires introduction; in addition, this research provides a new algorithm based on an artificial neural network. The following chapter therefore provides details about how reinforcement learning systems learn to distribute data and optimise cost and latency time. The next chapter also provides details about the hypothesis representation (artificial neural networks).

CHAPTER 7

Intelligent Framework for Optimising File Distribution Across Multiple Cloud Storage Services

The core of this research entailed learning how to balance differences in performance and cost across multiple cloud storage services, based on predictions of file access patterns. Learning in this context relies on the reinforcement learning paradigm. The reason for choosing this paradigm to optimise the cost and performance of cloud storage was its ability to maximise the expected long-term utilities in a large and dynamic environment, such as cloud computing. Furthermore, reinforcement learning is a powerful approach to deal with sequential decision tasks. This feature was apt for solving the problem of optimising costs and latency times over the long term on several cloud services. In addition, the best way to collect information about the behaviour of the cloud and its real cost was by interacting with it.

The framework for optimising file distribution across multiple cloud storage services, summarised as OFDAMCSS, has two learning components. These are supervised learning, described in Chapter 6 and reinforcement learning, which is covered in this chapter. In this study reinforcement learning involved the use of an artificial neural network as

a function approximator. This chapter starts by providing technical tips and discussing some of the issues in implementing the artificial neural network with the refinement learning algorithm. A description of how reinforcement learning works in this framework, including a novel idea for transferring the value state of each cloud to an action, is then described.

7.1 Characteristics of the Reinforcement Learning System

The reinforcement learning framework, as described in Section 3.3.3, interacts with its environment through (1) sensing the state of the environment, (2) performing an action, and receiving a reward for that action. In this section of the thesis, the state space, action space and reward function used to implement the reinforcement learning system within the proposed framework are defined.

- **State space:** in this work, the state space consisted of four continuous features. These features represented the attributes of the access pattern for each file. The attributes were: (1) size of the file, (2) number of times the file was predicted to be accessed in the future for reading only, (3) number of times the file was predicted to be accessed in the future for writing only, and (4) the file's lifespan. These features were fed to the artificial neural network as the input vector.

- Action space: this feature was also continuous. It represented the proportion of each file that would be allocated to each cloud storage service (i.e. a number between 0 and 100 for each cloud).

- Reward function: this feature was constructed based on the latency time and total cost, as described 4.

The system had one feed-forward artificial neural network, which used the state features as input and then produced several actions. The number of actions was equal to the number of clouds available at each time point. The feed-forward network in this study consisted of three layers of nodes (input, hidden and output), with the input layer having four nodes and the hidden layer 16. The output layer nodes were designed to expand and

shrink according to how many cloud storage services were available to the framework at each time step.

Using an artificial neural network with reinforcement learning requires certain engineering processes to build the network. Thus, it is worth discussing some tips and techniques that would usefully apply to any artificial neural network with any reinforcement learning application. The following section presents tips and steps that are helpful in building an artificial neural network as a function approximation for a reinforcement learning algorithm.

7.2 Artificial Neural Network with Reinforcement Learning

In any reinforcement learning application, the use of an artificial neural network with back-propagation to approximate the true value for each state can be a complicated process [Sutton 2004]. Unlike supervised learning, reinforcement learning has several parameters that require careful setting. Choosing one parameter value inaccurately can affect the quality of learning or cause a computer error. Unfortunately, little work has been published on the setting of parameters for artificial neural networks in reinforcement learning applications. The few examples include [Gatti & Embrechts 2013] and [Konen & Bartz-Beielstein 2008]. This section provides hints for training the artificial neural network using reinforcement learning algorithms to avoid any major issues.

As shown in Section 3.3.3.3, the reward value plays a crucial role in allowing the reinforcement learning system to learn, by guiding the state values towards their optimal values. There are two scenarios in which rewards are provided to the learning system. The first occurs at the end of a learning episode or when the agent reaches the desired goals. Examples include the games of chess and Tic-tac-toe and the car mountain benchmark. For this scenario, [Gatti & Embrechts 2013] provides basic techniques for implementing the artificial neural network in the reinforcement learning problem. The second scenario occurs when the reward is offered at each time step throughout each episode; an example of this scenario is the current work.

This section describes techniques and provides tips to build an efficient artificial

neural network for the reinforcement learning application in which a reward is provided at the end of each step. A rule of thumb is to determine the reward dimensions, as discussed in Section 3.3.3.3. Thereafter, the selection of the transfer function, at the output layer, must correspond to the diversity of the reward value. For example, if the reward range is [0,1], the transfer function at the output layer should operate over 0 and 1, including sigmoid function. Another example shows if the reward range is [-1,1], the hyperbolic tangent function or similar is a good choice, as it produces values between -1 and 1. The procedure is more delicate if the reward range space is infinite or large (continuous reward), which was the case in this work. Generally, the linear transfer function suits a continuous reward space. However, the main problem with continuous reward applications is that if the differences between the network output values and the reward values are vast, the training process might take a long time or might fail. In this research, this was indeed a problem. The best solution was to normalise the reward value to lie between two small values (upper and lower bounds), then choose the transfer function compatible with it.

On the other side of the network, for the hidden layer, the choice of the transfer function is less important than the transfer function at the output layer. However, [Gatti & Embrechts 2013] recommend using a hyperbolic tangent function or a modified version, such as $f(x) = 1.7159 \times tanh(\frac{2}{3}x)$. I did not find any significant effect of the choice of transfer function for this layer, as shown in the following chapter (Section 8.3.5). Thus, the recommendations of [Gatti & Embrechts 2013] were followed in this study.

Another important step is the choice of learning-rate values in each connection layer. Many studies indicate that network learning is improved if there are different learning rates for each connection layer[Nikolaev & Iba 2006] [Gatti & Embrechts 2013]. The main reason is that when a single learning rate exists for all layers, the connection weights between the hidden and output layers can change substantially, whereas those of the earlier connection layers might change very little, leading to inefficient training [Gatti & Embrechts 2013]. The learning rate has a considerable effect on the updating of the connection weight values, and affects the convergence and performance of the network.

Moreover, it can cause extremely small or large values of nodes or connection weights, which can result in an error on the computer. There are no constant rules for choosing to learn rate values. The best method might entail tuning the learning rate parameters after each training time until the best result is achieved. However, it is worth mentioning that two approaches were presented by [Embrechts et al. 2010] and [Gatti & Embrechts 2013] for tuning learning rate parameters on each of the connection layers, to avoid such issues in artificial neural networks. Due to time limitations these two approaches were not tried in this work; instead, the earlier approach was used (which is based on adjusting the learning rate randomly until the best result is reached). The best value was found relatively quickly.

In addition to the above settings, the temporal discount factor λ has an effect on connection weight updates. If the λ is zero or close to zero, the influence on the weight update is small. By contrast, if the λ is close to 1 (but $\lambda \neq 1$) the impact on the weight update is large, Finally, if $\lambda = 1$ is considered to follow Monte Carlo behaviour, but in a superior manner that can apply to continuous tasks and works incrementally instead of waiting until the end of the episode. [Gatti & Embrechts 2013] suggest choosing a $\lambda \approx 0.7$ which was found to be suitable starting value for this work.

7.3 Reinforcement Learning Model

In most reinforcement learning applications that use an artificial neural network as the function approximator, the artificial neural network is fed with the state vector s as input. The network produces multiple outputs, each of which represents $Q(s, a)$, as shown in Section 3.3.3.4. Thereafter the action can be chosen as either critic-only or actor-critic (Section 3.3.3.4). Critic-only is limited to a discrete action space, whereas actor-critic is suitable for continuous state and action spaces. Recent and popular examples of these two methods are provided by [Mnih et al. 2013] [Mnih et al. 2015] and [Silver et al. 2016]. These methods, especially the critic-only approach, usually require a fixed output neurone node, which did not suit the requirements of this work. [Silver et al. 2016] used two separate artificial neural networks, one for evaluation of the state of

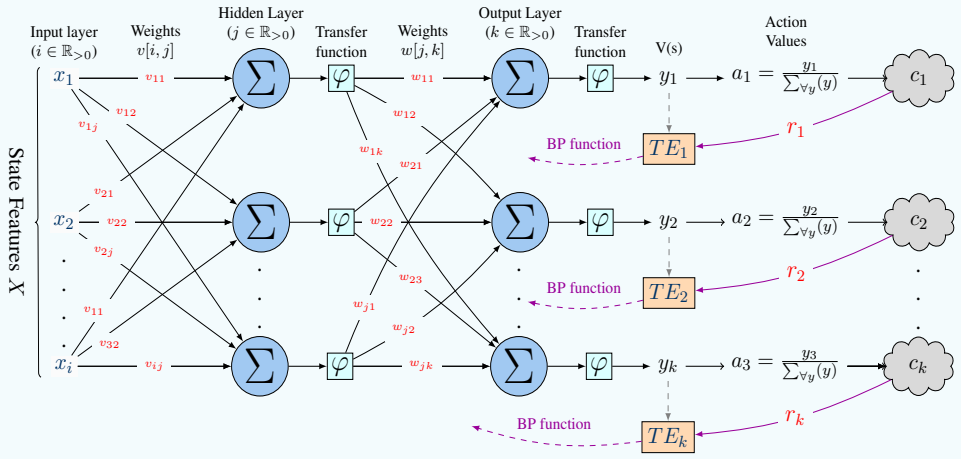


Figure 7.1: Demonstration of how the reinforcement learning system works in the OFDAMCSS framework. At each discrete time t, the learning system receives file access pattern attributes (from APPM) that represent the state features X, which will be passed through an artificial neural network to produce different outputs. Each output will be transformed to an action a_k, which is the proportion of file size that will be located in each cloud. These actions will be given to the Distributor system to perform. The agent then receives rewards from each cloud r_k, which will be used by the TD-error (TD_K) to evaluate each action separately. The back-propagation function (BP) will then be triggered to tune the connection weights (w_{jk}, w_{ij}).

the environment, and the other to choose an action that had the highest probability of winning. The difference between these approaches and that of this work is that the machine learning system in this work interacted with multiple cloud storage services (multiple environments). Each cloud storage service was susceptible to suffering from outages at any time. Thus, a relatively flexible method that could adapt to changes in the number of cloud storage services was needed for this study .

This work proposes a new approach for an artificial neural network with reinforcement learning that fulfils the target of this work, as illustrated in Figure 7.1. The goal of this new approach is to allow reinforcement learning to interact with each cloud independently to produce multiple output values, each of which represents the value of the state in different environments (i.e. different cloud services). For example, suppose

there are three cloud storage services, each of which has its own state space S. The artificial neural network takes all the state features from the APPM as the input vector and produces three output values, each of which corresponds to a single cloud. Each one of these values can be interpreted as $Q(s, a)$ for the state of the file on that cloud. Thereafter, as shown in Figure 7.1, each output value is transformed to a single numerical action, considering the value of other actions. The procedure is shown in Equation 7.1 below. That is, each action is the percentage of the value of node k (y_K) to the total value of all nodes value ($\sum_{\forall y}(y)$)).

$$(a_k)_t = \frac{(y_k)_t}{\sum_{\forall y}(y)} \tag{7.1}$$

where k represents the index number of each cloud provider, and y refers to the state value of cloud provider number k.

This method allows the reinforcement learning system to allocate a larger portion of each file to the cloud that has the largest state value. After executing all actions, the reinforcement learning system receives a different reward from each environment. These rewards are used to compute the multiple TD-error functions, then the $\vec{W}$ is adjusted using the back-propagation (BP) function.

One of the most difficult problems in this research was the process of computing the reward function. The process is outlined in Algorithm 4. The reward value relies on two unbounded values: cost and latency. To optimise the latency time, the ratio of the current latency time to the maximum latency time (observed by the framework during the interaction with all environments) was computed. The maximum latency time is susceptible to change when the system finds a new maximum. Thereafter, the amount obtained was multiplied by -1 to reduce the latency time.

However, the process used with latency time cannot work with cost value. The cost of cloud storage services is cumulative, which means it rises over time within the billing period. Hence, optimising the cost of cloud storage after each action is rather challenging. Therefore, to optimise the cost, first the cost was computed with respect to the

Algorithm 3 Distribution framework outline

Require: : Initialise artificial neural network weight w_{ij} and w_{jk}

for <episode =1, M> **do**
  for <t=1, N> **do**
    $c\_num \leftarrow$ check how many cloud storage services are available
    adjusts numbers of nodes in the output layer to be equal to c_num
    Get file size and file access pattern attributes and pass them to the artificial neural network.
    for <k=1, c_num> **do**
      $(y_k)_t \leftarrow$ output value of node k
    end for
    for <k=1, c_num> **do**
      generate action $(a_k)_t = \frac{(y_k)_t}{\sum_{\forall y}(y)}$
    end for
    Execute all actions $(a)_t$
    Observe rewards $(r)_t$ from all clouds ▷ based on Algorithm.4
    Compute TD-error for all clouds
    Update network weight
    ▷ according to Eq. 3.13 and Eq. 3.15
  end for
end for

amount of data used, as expressed below:

$$cost = \sum_{i=1}^{n}(\frac{Scost_i}{Sused_i} + \frac{Ncost_i}{Nused_i} + \frac{OTcost_i}{OTused_i}) \tag{7.2}$$

where $Scost$ is the total cost of total storage of data into cloud i , $Sused$ is the amount of data that has been stored so far in cloud storage i, $Ncost$ is the total network cost, $Nused$ is the amount of transferring data to and from the cloud storage services, $OTcost$ is the total cost of performing all operations in cloud i, ($OTused$) is the total number of operations that have been performed on the cloud provider i, and n denotes the number of cloud providers available. The aim beyond this equation is to compute the change of price with respect to the amount used.

Typically, the above equation yields a value less than 1. Computing the ratio of the cost in a similar way to that of the latency time did not work in this study. Thus,

a slightly different method was used. The convergence of learning was improved by adding the following condition: any value of the above equation that exceeded 1 would be used to compute the ratio of all subsequent steps, until a new value exceeded 1; then the system overrides this value with the new value. This technique enabled the system to learn how to optimise the cost perfectly and rapidly. Finally, to make the framework operate more efficiently to optimise both latency and cost concurrently, different weights for cost and latency were added to the reward function. The weights were based on the importance of the file, which was extracted from the file access pattern attributes. If the file was estimated to be very active in the future, the framework would reduce the importance of cost and focus on improving latency time. The same was true in reverse: for inactive files, the importance of latency time was reduced and cost was optimised. The entire reward function is outlined in Algorithm 4.

In short, by giving the network the predicted attributes of each file (Section 6.4) along with the file size, the network could produce multiple output values that corresponded to the number of cloud storage services available at a time t. The output values represent the value of storing the file on each cloud. Based on these results, the reinforcement learning agent decided what proportion of the file size would be sent to each cloud. The decisions were then evaluated and tuned using the TD-error and back-propagation functions.

One of the strengths and novel methods of the framework is that the number of outputs of the network is changeable, corresponding to the number of cloud storage providers available at each time step t. This feature gives the framework the ability to overcome the issue of service availability and continuity. For example, if one of the cloud providers is not available for any reason, or the data owner has added new cloud storage, the number of output nodes can be changed accordingly and automatically – without human intervention or the need to reset the network. Specifically, when a cloud storage service cannot be reached for any reason, whether permanently or temporarily, the system eliminates from the output layer the node that holds the value of that cloud's state. In so doing, the relevant connection weights are also removed. Similarly, when a new cloud storage service appears or an old one comes back into service again, the

Algorithm 4 Reward function based on latency time and total cost of distributing each file across multiple cloud storage

Require: : Initialise number of cloud storage services available $numCloud$
Require: : Initialise variables for the most important file , and maximum latency and cost for each cloud ($maxWeight$,$maxLatencyArry[]$, $maxCostArry[]$)
Require: : Initialise three variables ($latencyReward[]$,costReward[],cost[]) in order to compute the total reward

$w \leftarrow$ calculate the important of the file from the prediction of access pattern ($numReard + numWrite + lifeTime$).

if $w > maxWeight$ **then**
    $maxWeight \leftarrow w$; ▷ find the most important file
end if
$w = w/maxWeight$ ▷ compute the ratio of the current latency time to the maximum weight yet.

for <i =1, numCloud> **do**
    if $fileLatencyTime[i] > maxLatencyArry[i]$ **then**
        $maxLatencyArry[i] \leftarrow fileLatencyTime[i]$ ▷ find the slowest file
    end if
    $latencyReward[i] \leftarrow -1 * (fileLatencyTime[i]/maxLatencyArry[i])$, ▷ compute ratio of the current latency time to the slowest file latency yet, for each cloud.
    compute $cost[i]$ according to Eq 7.2
    if $cost[i] > 1$ **then**
        $maxCostArry[i] \leftarrow cost[i]$
    end if
    $costReward[i] \leftarrow -1 * (cost[i]/maxCostArry[i])$ ▷ ratio of the current cost to the maximum cost yet, for each cloud
    $totalReward[i] \leftarrow (1 - w) * costReward[i] + w * latencyReward[i]$ ▷ compute the total reward for distributing one file
end for
return $totalReward$

system establishes a new node in the output layer and creates all its connection weights. This process of learning is outlined in Algorithm 3 for a given number of episodes M; 'episode' here means a single learning step in reinforcement learning. .

7.4 Summary

In this chapter, the main components of this research were described. The research presents a reinforcement learning model that uses an artificial neural network. In addition, this chapter presented one a novel algorithm that transfers the value of a state into an action for each cloud storage service. The following chapters present material on the experiments that were conducted to evaluate the proposed framework (OFDAMCSS).

CHAPTER 8

System Evaluation

In the previous chapter, the architecture and design of the OFDAMCSS framework components were defined and described. These components were designed to distribute files across multiple cloud storage services. In this chapter, the experiments conducted to evaluate the framework are discussed. The evaluation was aimed at ensuring the system could effectively optimise the cost and latency time over multiple cloud storage services, based on the prediction of file access patterns. The chapter starts with a description of the experimental methodology. The next section provides the results of the evaluation. The method of testing the generative ability of the framework through experiments with several cloud storage services is also described. This is followed by an analysis of all the experimental results, and a discussion of the effects of experimental parameters on the framework. The chapter concludes with a summary.

8.1 Experiment Settings

The experiments were conducted using the cloud storage emulator (Section 5.2). The emulator was used to emulate the performance (latency time) of several cloud storage

services and to calculate the total cost of using the services in each cloud, based on storage, bandwidth, and operations. As mentioned in Chapter 5 5, the performance of the providers in these experiments was set up to resemble Google Cloud Storage, Amazon S3, Microsoft Azure Storage, and RackSpace Cloud File. The performance of these providers was measured by the performance analysis service offered by cloudharmony.com.

As described in Section 6.4the first step in running the framework is to predicate the access patterns for each file using APPM. The file size accompanied by access pattern attributes is then fed to the reinforcement learning system, which is trained with the artificial neural network. The implementation of the learning algorithm and the artificial neural network requires setting several parameters. These parameters are likely to influence the ability of the framework to learn. Initially, the parameters were set according to the recommendations of other researchers, including [Gatti 2015], [Gatti & Embrechts 2013] and [Tesauro 2002]. The settings are shown in Table 8.1 and Table 8.2.

Table 8.1: Reinforcement learning parameter settings

parameters	values
learning rate	$\alpha = 0.001$
Temporal discount factor	$\lambda = 0.7$
Next state decay parameter	$\gamma = 0.75$
Number of training times (episode)	200000

The initial settings of the artificial neural network consisted of a fully connected three-layer network with four input nodes, 16 hidden nodes, and a flexible number of output nodes. As described above, the number of output nodes was based on the available cloud storage services at each time step. The input and hidden layer had a bias node with a constant value of +1. All hidden nodes used a hyperbolic tangent transfer function and the output node used a sigmoid transfer function. The learning algorithm parameters were initially set as follows: $\epsilon = 0.1$; $\lambda = 0.7$;$\gamma = 0.9$.

Subsequently, individual experiments were performed to assess the effect of changing parameters and settings on the quality of learning. Further descriptions of these

experiments are presented found in Section 8.3.5.

Table 8.2: Artificial neural network parameters settings

parameters	values
# input nodes	4
# hidden nodes	16
# output nodes	= # cloud storage services
Weight init. method	randomly $[-0.2, 0.2]$
Transfer function	$f(x) = \frac{1}{1+e^{-x}}$
Network learning rate	$\beta = 0.0007$
Network Momentum	$\eta = 0.5$

The framework was evaluated using four datasets with a total of more than 9154 files and a total size of more than 854 GB. The files were distributed over four cloud storage services. The reason for choosing four cloud storage services was that RAID Level 5 requires at least three storage spaces to distribute files. However, if there were only three cloud stores and one suffered from outages for any reason, the whole system would not work; hence, four cloud storage services was the best choice.

At the start of the experiment, the latency time and the total cost for each cloud provider were measured individually by sending whole files to each provider (i.e. without splitting the distribution). These measurements provided the baseline. The latency and total cost were then measured again after distributing the same files uniformly among the cloud providers, using the principle of RAID with a standard configuration.

Finally, to evaluate and test the flexibility of the OFDAMCSS framework, three scenarios were defined. ('Flexibility' here means the manner in which reinforcement learning adapts to a change in the number of cloud services.) The three scenarios were as follows:

- **Scenario 1:** the number of cloud providers is fixed and does not change during the learning process.
- **Scenario 2:** one of the cloud providers is removed from the cloud storage array in

the middle of the learning process.

- **Scenario 3:** a new cloud provider is added to the storage array in the middle of the learning process.

The goal of these scenarios was to test the robustness of the proposed framework, especially regarding the continuity and availability issues mentioned in Chapter 2.

8.2 Evaluation Methods

Three methods were used in this work as a benchmark to evaluate the proposed framework. In short, the first method was to send all files into each single cloud storage service without any distribution; this was the 'mirroring method' (details in Section 8.2.1. The second method was to distribute all files into all available cloud stores, using standard RAID distribution (SRD) (details in Section 8.2.2). The third method was to distribute all files into all available cloud stores using a heuristic distribution code (details in Section 8.2.2). The datasets described in Section 6.3 were used to evaluate the proposed system by comparing it with these three methods.

8.2.1 Mirroring Approach

This method required testing each cloud provider independently, without any split distribution of data. Table **??** shows the results of uploading the four datasets to each cloud services without distribution. The results are given as the average total cost per year, which was $210 for Google (standard deviation (SD) = 35.1; 95% confidence intervals (CI=1.1)); $172 for Amazon S3 (SD = 28.8; the CI=0.9) ;$104 for MS Azure storage (SD = 17.46 ; CI=0.54); and $223 for RackSpace (SD = 37.56; CI=1.18) . The results also show that the average latency times was 39.2 seconds to read, and about 9.8 seconds to write in Google ; in Amazon, 40.9 seconds to read and 10.6 seconds to write; in MS Azure , 40.2 seconds to read and 10.6 seconds to write; and in Rackspace 44.98 seconds to read, and 11.25 seconds to write. Table **??** shows the complete analysis of these results for each dataset using this benchmark method .

8.2.2 Principle of Standard RAID Distribution Approach

This benchmark method uses the principle of the standard RAID distribution (SRD) method; more specifically, RAID level 5 was used only because it is the most popular method. Using this method, all datasets were distributed individually across all the cloud providers uniformly. Figure 8.2a and Figure 8.2b show the total cost and average latency time of distributing files uniformly by SRD. The average costs of distributing the four datasets into all clouds was about $209 per year (SD=35; CI=1.09) per year. The average latency time was 15.3 seconds for reading (SD=1.6; CI=0.052) and about 3.8 seconds for writing (SD=0.42; CI=0.01). Table 8.6 shows a summary statistics for the datasets after distributing using the SRD method.

Table 8.4: Summary statistics for uploading all datasets into all cloud storage using the SRD method. Latency_r, latency of reading; latency_w, latency of writing.

Dataset Name	Distribute all datasets into all clouds using ***SRD*** framework		
	Total Cost	Avg latency_r	Avg. Latency_w
dataset_1	249.67	15.26	3.81
dataset_random	186.58	13.37	3.34
dataset_2	175.02	15.14	3.79
dataset_3	228.22	17.49	4.37
Average	**209.8725**	**15.315**	**3.8275**
Standard Deviation (STDEV)	35.012898	1.687927724	0.421772055
CONFIDENCE INTERVAL (CI95)	1.097773	0.052922254	0.013223983
confidence range at 95%	210.97027	15.36792225	3.840723983

8.2.3 Heuristic Distribution Approach

This method extends the evaluation of the proposed framework by comparing it with a basic heuristic approach. The parameters used were related to total cost and average latency times for reading and writing. Algorithm 5 shows the pseudo-code of the heuristic approach. Basically, this approach searches in portion parameter space for the best tuning parameters to optimise the cost and latency time on multiple cloud storage services. As shown in Algorithm 5 , each cloud's portion parameters were set to $\frac{1}{numberofcloud}$. For example, if four cloud storage services were used, the portion parameter setting was 0.25 for each cloud. The portion settings were used as the first threshold, and based on that the algorithm searched in the parameter space for the best

portion parameter settings. After each iteration the algorithm compared the latency time and the cost that resulted from distributing a file using the current portion parameter with the previous result (using the previous portion parameters). If any improvement was noted in either the latency or cost, a new threshold was set and the algorithm would continuously search for better distribution parameters. For example, after setting a portion parameter of 0.25 for each cloud (if there were only four clouds), the tuning process started by choosing the first cloud portion parameter and increasing it by 0.01; the second cloud would be decreased by 0.01. If this new setting did not improve either the cost or latency, the algorithm would increase the portion for one cloud and would decrease the portion in another cloud. .

Algorithm 5 Pseudo-code of Heuristic Distribution Approach

Require: : Initialise : $old_T totalCost$, $minimum_L W$, $minimum_L R$, $bestCost$

Require: : Initialise portion parameters, each cloud has $\frac{1}{number of cloud}$ portion

1: **while** dataset file not finished **do**
2: perform a portion parameters and distribute files based on number of cloud available
3: $LR \leftarrow$ compute the latency time for READING from all clouds
4: $LW \leftarrow$ compute the latency time for WRITING from all clouds
5: $totalCost \leftarrow$ compute the total cost from all clouds
6: $changedCost \leftarrow old_T totalCost - totalCost$ ▷ compare the old total cost a new cost
7: **if** $LR < minimum_L R or LW < minimum_L W or (changedCost > 0 and changedCost < bestCost)$ **then**
8: $minimum_L R \leftarrow LR$
9: $minimum_L W \leftarrow LW$
10: $bestCost \leftarrow totalCost$
11:
12: **else** ▷ search for new portion parameters
13: increase one portion that assigned to a cloud by 0.01, and decreases another portion by 0.01 (i.e tuning portion parameters).
14: **end if**
15: **end while**

Table 8.5 shows the summary statistics for uploading all datasets into all cloud storage services, using the heuristic code described above. These results were used as a benchmark together with the SRD results to evaluate the proposed framework..

Table 8.5: Summary statistics for uploading all datasets into all cloud storage using the heuristic method. Latency_r, latency of reading; latency_w, latency of writing

Dataset Name	Distributing all datasets into all clouds using ***Heuristic Distribution*** code		
	Total Cost	Avg latency_r	Avg. Latency_w
dataset_1	188.8	8.08	2.02
dataset_random	141.27	7.08	1.77
dataset_2	132.3	8.02	2
dataset_3	172.47	9.26	2.31
Average	**158.71**	**8.11**	**2.025**
Standard Deviation (STDEV)	26.434746	0.893009892	0.221284131
CONFIDENCE INTERVAL (CI95)	0.8288189	0.027998886	0.006938007
confidence range at 95%	159.53882	8.137998886	2.031938007

8.3 OFDAMCSS: Experiments and Analysis

This section provides details for the experiments using the OFDAMCSS framework to distribute all datasets (Section 6.3). The results for all experiments were evaluated with the three benchmarks discussed in the previous section.

8.3.1 Overall result

As described earlier, the latency times were tested and the total cost for each cloud provider was calculated individually. This procedure was followed for emulations of Google Cloud Storage, Amazon S3, MS Azure, and RackSpace File cloud, using three approaches as a benchmark to evaluate the proposed framework.

The bar chart in Figure 8.1 illustrates the differences in average total cost and average latency times across four different cloud providers, using the mirroring approach. The main purpose of mirroring data across all clouds (without any split distribution) was to gain insight into the real cost and latency time of using a single cloud storage, and the disparity among different cloud service providers.

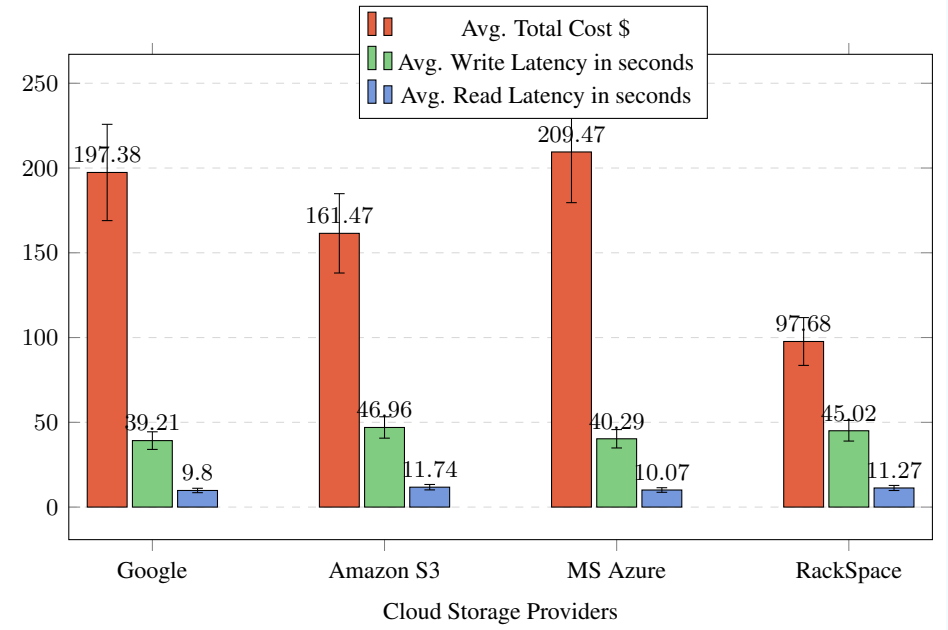
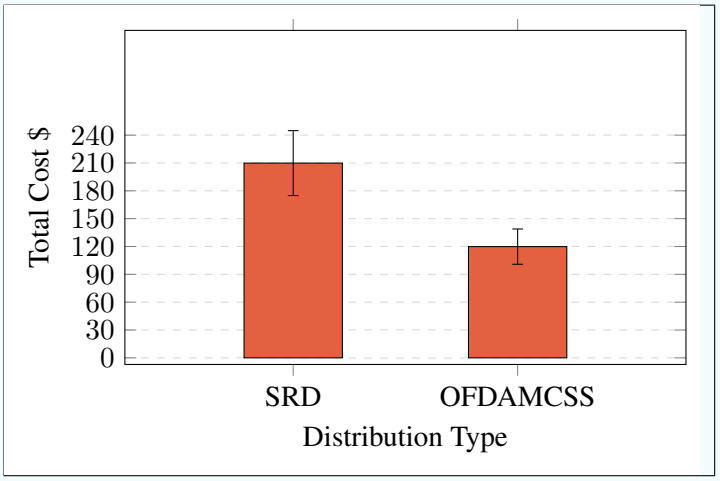


Figure 8.1: Average total cost and average latency time to send the entire data from all datasets into one cloud storage service

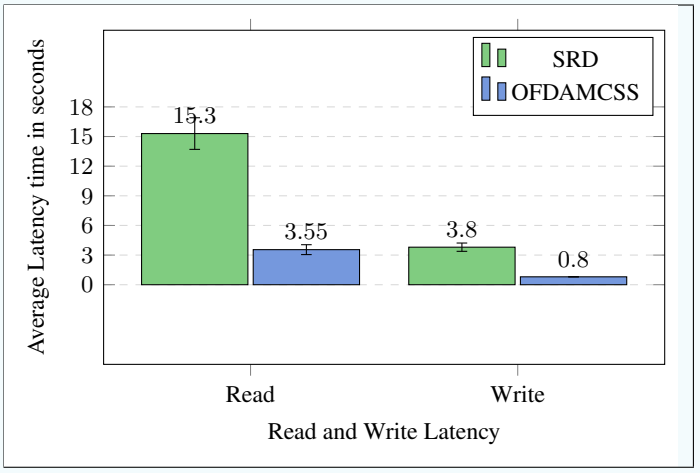
Table 8.6: Summary statistics for uploading all datasets into all cloud storage using the proposed OFDAMCSS framework. Latency_r, latency of reading; latency_w, latency of writing

Dataset Name	Distribute all datasets into all clouds using *DFAMCSS* framework		
	Total Cost	Avg latency_r	Avg. Latency_w
dataset_1	143.11	3.71	0.93
dataset_random	109.66	2.81	0.7
dataset_2	98.29	3.71	0.93
dataset_3	128.41	3.97	0.99
Average	**119.8675**	**3.55**	**0.8875**
Standard Deviation (STDEV)	19.857573	0.508330601	0.128160056
CONFIDENCE INTERVAL (CI95)	0.6226022	0.015937887	0.004018252
confidence range at 95%	120.4901	3.565937887	0.891518252

Thereafter, the same datasets were distributed using the OFDAMCSS framework over the same cloud storage services. As shown in Figure 8.2 and Figure 8.3a, provided a better result than either the SRD or the heuristic approach, with an average total cost of $119.86 (SD = 19.85; CI = 0.62). These results showed that the proposed model



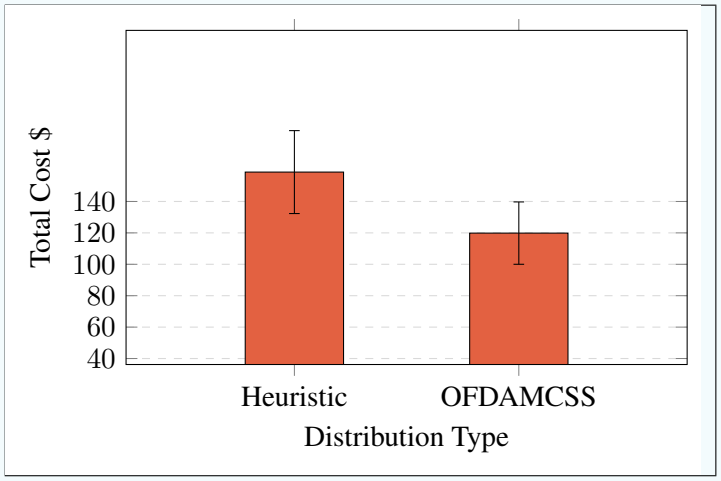
(a) (a) The difference in average cost after distributing all files by SRD and OFDAMCSS.



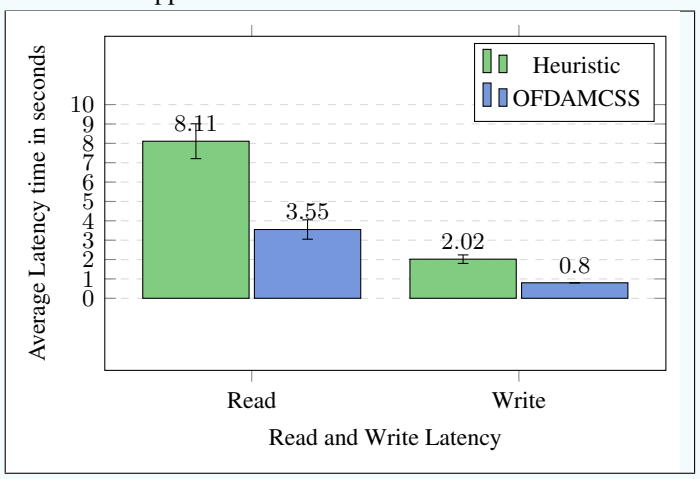
(b) (b) The difference in latency time (read and write) after distributing all files by SRD and OFDAMCSS.

Figure 8.2: Distributing all files from all datasets across multi-cloud using standard RAID distribution SRD and OFDAMCSS

outperformed SRD by about 42% and the heuristic approach by about 24%. The average latency times for reading and writing were significantly reduced, by more than 76% compared with SRD and by about 56% compared with the heuristic approach. The latency time for reading, using the OFDAMCSS framework, was 3.55 seconds (SD=0.5; CI=0.015); and the latency time for writing was 0.88 seconds(SD=0.12; CI=0.004).



(a) The difference in average cost after distributing all files by the heuristic approach and OFDAMCSS.



(b) The difference in latency time (read and write) after distributing all files by heuristic approach and OFDAMCSS.

Figure 8.3: Distributing all files from all datasets across multi-clouds, using the heuristic approach and OFDAMCSS

These results show that distributing data based on file access patterns, together with the use of reinforcement learning, offers a promising solution and can save money and time for enterprises.

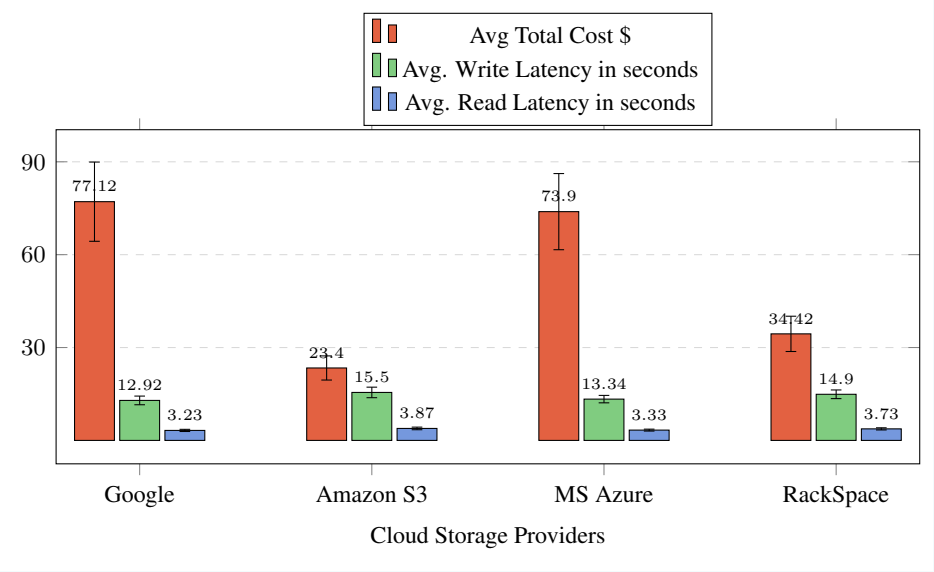


Figure 8.4: Total cost and average latency time for each cloud provider using SRD

8.3.2 Analysis of Results

An overall reduction in total cost and average latency times does not indicate that the cost and average latency were reduced for all cloud providers. The OFDAMCSS framework did reduce file sizes for some cloud providers, whereas others received an augmentation in file sizes, which resulted in increased cost and average latency time.

Figure 8.6 illustrates the changes in cost and average latency time when using OFDAMCSS framework, compared with SRD (Fig 8.4) and Heuristic approach (Fig 8.5). Figure 8.1 latency time and cost refer to the average values for each cloud storage services after uploading all four datasets without any split distribution (mirroring). This differs from the latency time and cost shown in Figures 8.4, 8.5, and 8.6, which relate to individual cloud storage services with split distribution. Figure 8.4,8.5, and 8.6 show that the average cost for Google was reduced from $77.12 (SRD) and $74.8 (heuristic) to $35.54 when the OFDAMCSS was used (SD=10.7; CI=0.33). This result translates to -54% of the SRD values and -52% of the heuristic approach values. The average latency time dropped by -55% compared with SRD and by about -54% compared with

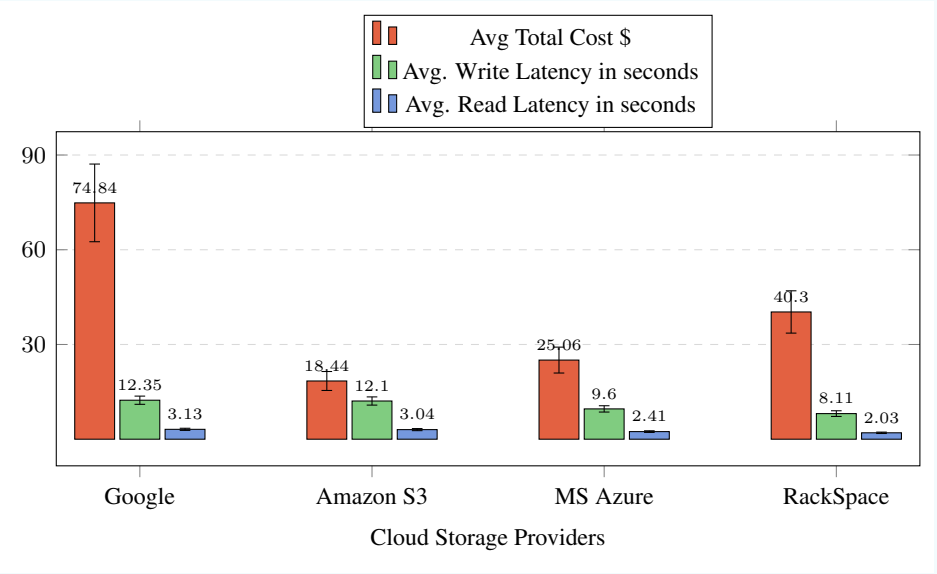


Figure 8.5: Total cost and average latency time for each cloud provider using the heuristic approach

the heuristic approach. For the OFDAMCSS model, the latency time for reading was 5.69 seconds (SD=1.0; CI=0.03) and 1.42 seconds for writing (SD=0.2; CI=0.0079) (Figure 8.6). The average cost and average latency time went down for RackSpace to $9.0 (SD=4.4 and CI=0.13) for the cost. This result translates to -67% compared with SRD and -55% compared with Heuristic approach. The latency time for RackSpace was reduced to 3.8 seconds (SD=1.6 ; CI=0.05) for average reading and to less than 1 second (SD=0.3 ; CI=0.012) for writing. This result translates to -67% compared with using SRD approach and -55% compared with Heuristic approach.

However, distributing files using OFDAMCSS framework increased the average cost and average latency time for Amazon S3 by around 18% compared with SRD and 5% only compared with Heuristic approach(to $27.67 (SD=4.9; CI=0.15)). In addition the average latency time for Amazon S3 went up by more than 16% compared to SRD and 47% compared to Heuristic approach (to 18.0 seconds (SD=2.1; CI=0.06) for reading, and to 4.5 seconds (SD=0.5; CI=0.01) for writing. The average cost in MS Azure storage increased,also, by about 55% compared to SRD and 62% compared to Heuristic

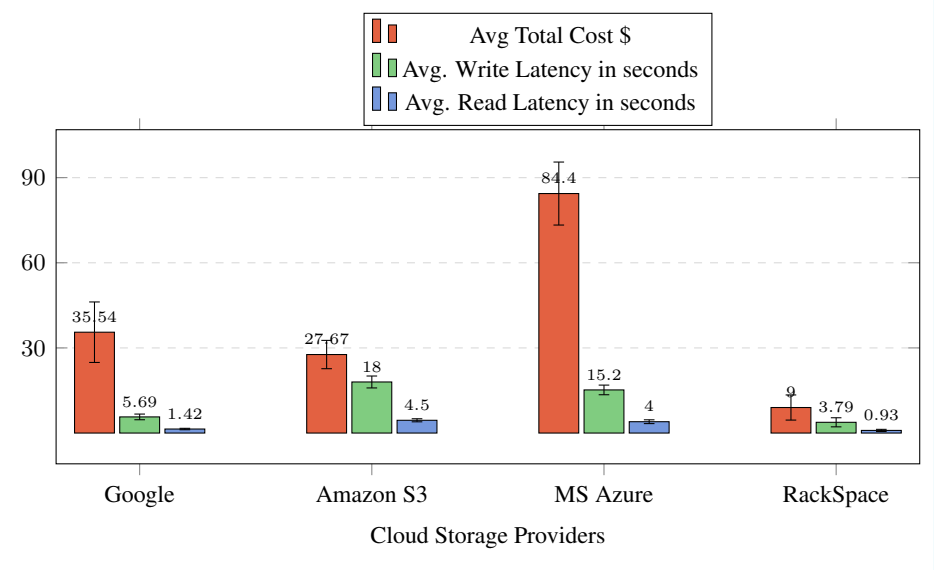
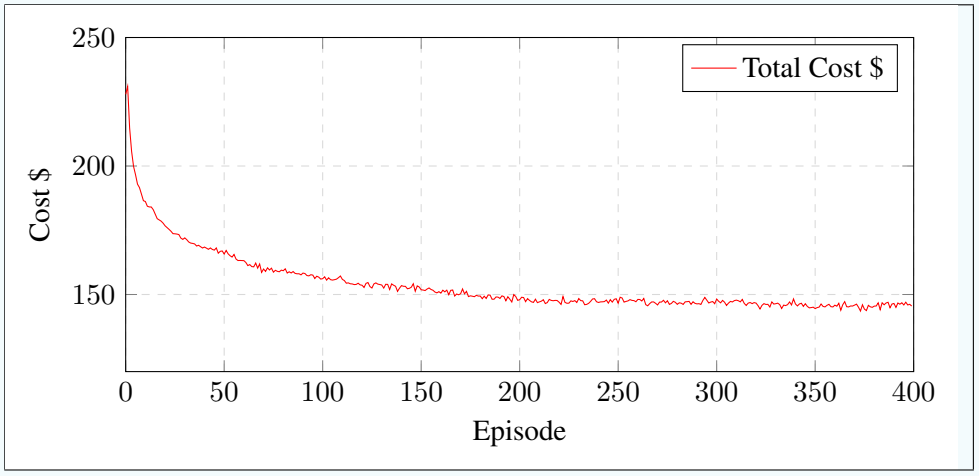


Figure 8.6: Total cost and average latency time for each cloud provider using OFDAMCSS framework

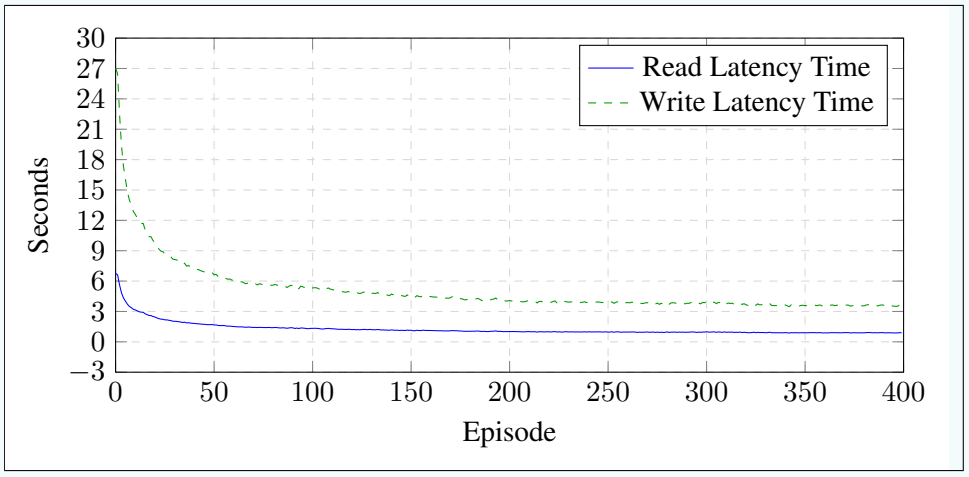
approach. The average cost rose to to $84.4 (SD=11.1; CI=0.34). The average latency time also rose by more than 56% compared to SRD and 62% compared to Heuristic approach. This results translates to 15.2 seconds (SD=1.7;CI=0.05) for reading, and to 4.0 seconds (SD=0.67; CI=0.02) for writing. Table 8.7 shows a graph of the statistical results for the differences between distributing all datasets using SRD, heuristic approaches and the proposed framework in this thesis, OFDAMCSS.

The graph in Figure 8.7a shows how the total cost (a combination of network, storage and operational costs) became smaller over the learning time. Similarly, Figure 8.7b shows how the average latency time was reduced over the learning episodes.

Finally, the ability to adapt to changes in cloud provider availability and continuity was tested in the OFDAMCSS framework by performing several experiments, based on the scenarios mentioned in Section 8.1. The aim of the experiment was to examine changing the number of cloud providers during the learning time, by adding or removing a cloud provider on the list. The experiments showed that the proposed framework was flexible and adaptable when the number of cloud services changed. Figures 8.8a and



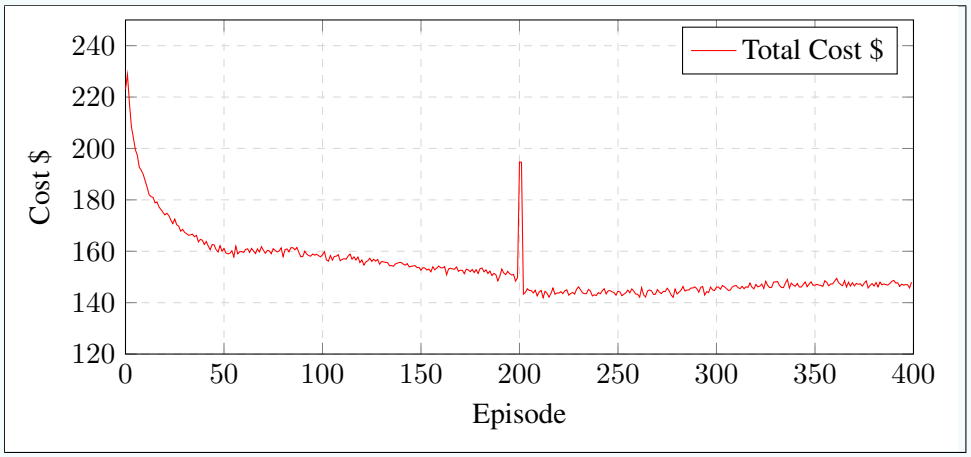
(a) Scenario # 1, cost decreases with learning



(b) Scenario # 1, latency time decreases with learning

Figure 8.7: The reduction in total cost (in figure (a)) and latency time of reading and writing (in figure (b)) reduces over time , using OFDAMCSS

8.8b illustrate how the OFDAMCSS framework adapted to change when the number of available cloud providers was reduced from four to three. Additionally, the OFDAMCSS framework learned any new cloud storage behaviour quickly and adapted to the new situation. Figures 8.9a and 8.9b show the framework's adaptation when a new cloud provider was added during the learning time.



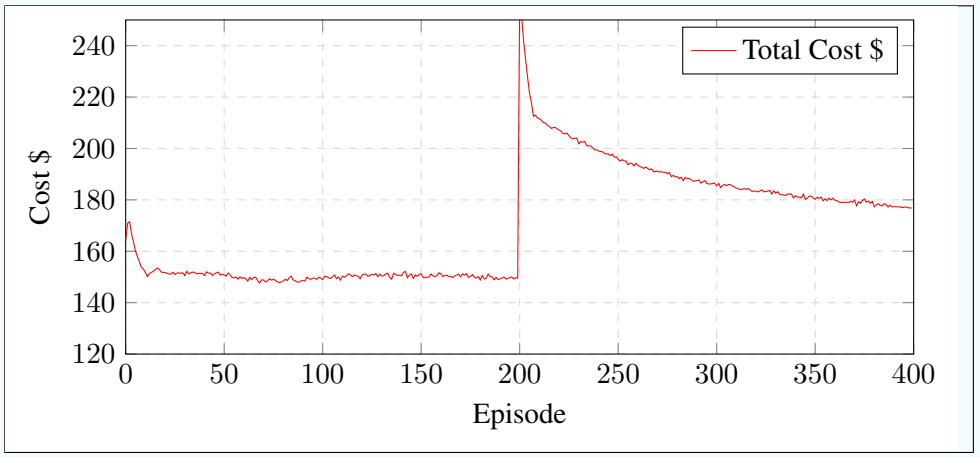
(a) Scenario # 2, Cost increased when a cloud stores was removed and the load of distribution on storage became less

(b) Scenario # 2, Latency time increased when a cloud stores was removed and the load of distribution on storage became less

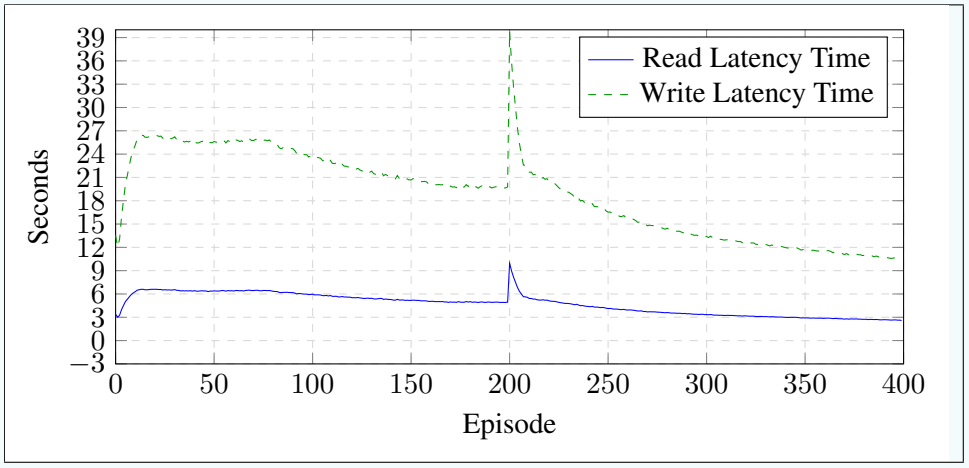
Figure 8.8: Change in the total cost (in figure (a)) and latency time of reading and writing (in figure (b)) change when a cloud stores was discontinued, (using OFDAMCSS)

8.3.3 Developmental of the System

This thesis has so far examined the proposed framework, based on real settings for the four main providers of cloud storage services (Google Cloud Storage, Amazon S3, MS Azure, and RackSpace File Cloud). In this section, the generative parameters of the



(a) Scenario # 3, cost rises as a cloud is added, then starts going down again



(b) Scenario # 3, latency increases as a cloud is added, then falls rapidly because the load is distributed

Figure 8.9: Change in the cost (in figure (a)) and latency time of reading and writing (in figure (b)) change when a new cloud stores was added to the system , using OFDAMCSS layer

OFDAMCSS framework is assessed. First, the range [minimum – maximum] of latency times is defined,based on CloudHarmony.com tests and various prices, for all services available (Table 8.8).

The next batch of experiments generated arbitrary settings and prices for arbitrary cloud storage services. In each experiment, several cloud storage services were created

each of which with different settings within the ranges shown in Table 8.8.

Figures 8.10a and 8.10b show that the OFDAMCSS framework can successfully optimise both cost and latency time at once, for any number of group cloud storage services. Several experiments were performed on a different group of cloud storage services. For each group, the emulator generated several cloud storage services, each of which had different latency times and different price schemes. Comparing with the SRD approach, the cost dropped among all experiments by about 31% for groups of three cloud stores, 34% for groups of four cloud stores, about 30% for groups of five cloud stores, and by about 46% for six cloud storage services. In addition, the results showed enhanced performance across all experimental groups, with the latency time being reduced by about 47% for the group of three cloud stores, 51% for four cloud stores, 55% for five cloud stores, and about 63% for six cloud storage services.

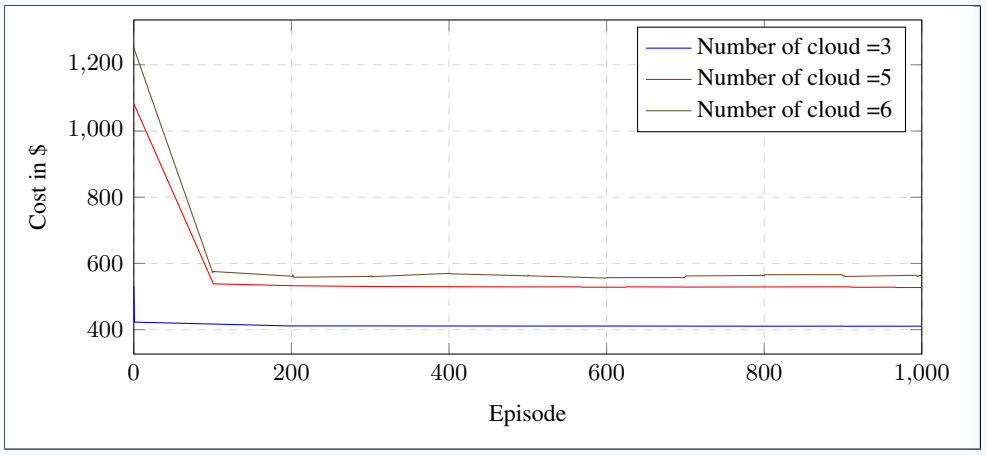
Table 8.8: Test of parameters settings for generative parameters of OFDAMCSS

–	Range	units of measure
Latency Time	[6,25]	Mb/s
Storage Cost	[0.01,0.1]	$/GB
Network Cost	[0.09,0.2]	$/GB
Operation Cost	[0.001,0.002]	$/1000 operations

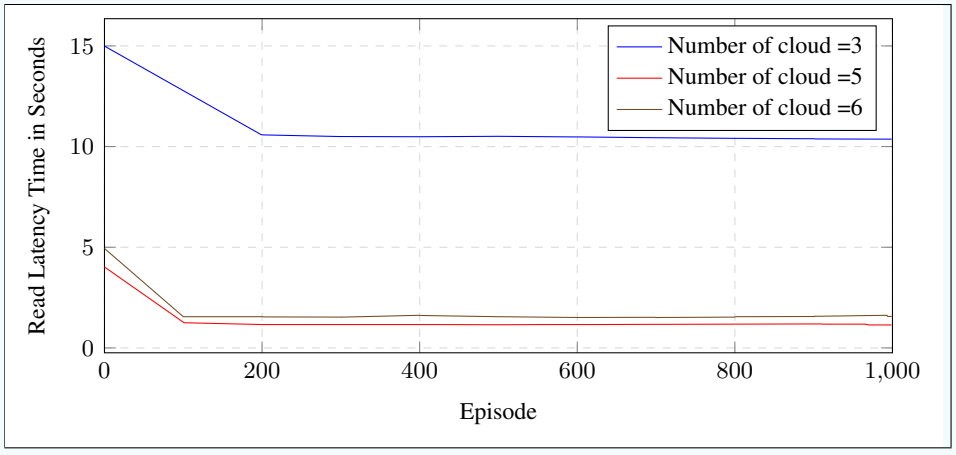
8.3.4 Impact of Access Pattern on Distribution Decisions

All the results from all the experiments discussed in the preceding section showed the robustness of the OFDAMCSS framework. To illustrate in depth how this framework works, this section provides an analysis of the changes in data used, as well as the changes in cost and latency times for each cloud provider. This part of the analysis focuses on the access pattern attributes.

First, files were divided into two separate groups. The first group was based on the number of writing attributes and the second group was based on the number of reading attributes. Each group was divided further into subgroups, based on the number of read and write attributes. The writing group was divided into eight subgroups and the reading group into 15 subgroups. The reason for this division was to observe how the



(a) Optimising cost in different groups of cloud storage services



(b) Optimising latency time in different groups of cloud storage services

Figure 8.10: Generative features of OFDAMCSS framework by testing the ability of the framework to optimise cost and latency time at the same time were tested for different groups of cloud storage services

OFDAMCSS framework distributed each subgroup and how this affected the cost and latency time for each subgroup. Moreover, to evaluate the framework accurately, the cost of the network for each file was multiplied by the number of reading and writing attributes, to estimate the real cost for each file throughout its lifetime. Tables 8.9 and 8.13 show that the distribution by OFDAMCSS, among all writing groups, decreased the cost of storage on Google Cloud Storage and RackSpace. However, the cost of MS

Azure and Amazon S3 increased with the proposed framework. Similarly, the average time required to transfer files to Google Cloud Storage and RackSpace was reduced, but it increased for Amazon S3 and MS Azure (Table 8.17). The reading subgroups showed similar results, as shown in Tables 8.11 and 8.15.

The above results show that the proposed framework was unable to minimise the cost and improve the performance of every cloud storage service. However, the framework considers the total cost and average latency time of all cloud storage services.

Distribute all dataset into all clouds using ***DFAMCSS*** framework												
Dataset Name	**Google Cloud Storage**			**Amazon**			**MS Azure**			**Roackspace**		
	Total Cost	Avg latency_r	Avg. Latency_w	Total Cost	Avg. latency_r	Avg. Latency_w	Total Cost	Avg. latency_r	Avg. Latency_w	Total Cost	Avg. latency_r	Avg. Latency_w
dataset_1	51.41	7.02	1.76	31.24	17.29	4.32	48.33	15.64	3.91	21.36	3.59	0.9
dataset_random	30.47	4.79	1.2	25.16	16.38	4.09	36.16	13.7	3.43	15.93	3.15	0.79
dataset_2	28.48	5.9	1.47	21.89	17.16	4.29	33.89	15.52	3.88	15.14	3.56	0.89
dataset_3	31.8	5.06	1.26	32.39	21.19	5.3	44.32	17.92	4.48	19.63	4.11	1.03
Average	**35.54**	**5.6925**	**1.4225**	**27.67**	**18.005**	**4.5**	**40.675**	**15.695**	**3.925**	**18.015**	**3.6025**	**0.9025**
Standard Deviation (STDEV)	10.66759892	1.003307032	0.253031619	4.991051993	2.161025991	0.543016267	6.789857632	1.72861216	0.430232495	2.967046792	0.393393357	0.098446263
CONFIDENCE INTERVAL (CI95)	0.334465378	0.031457076	0.007933399	0.156486395	0.067755488	0.0170254	0.212885047	0.054197849	0.013489247	0.093026972	0.012334215	0.003086624
confidence range at 95%	35.87446538	5.723957076	1.430433399	27.82648639	18.07275549	4.5170254	40.88788505	15.74919785	3.938489247	18.10802697	3.614834215	0.905586624

Distribute all dataset into all clouds using ***SRD*** framework												
Dataset Name	**Google Cloud Storage**			**Amazon**			**MS Azure**			**Roackspace**		
	Total Cost	Avg latency_r	Avg. Latency_w	Total Cost	Avg. latency_r	Avg. Latency_w	Total Cost	Avg. latency_r	Avg. Latency_w	Total Cost	Avg. latency_r	Avg. Latency_w
dataset_1	91.66	12.88	3.22	27.86	15.42	3.86	31.05	10.02	2.51	66.78	11.22	2.8
dataset_random	68.86	11.27	2.82	20.74	13.51	3.38	23.24	8.78	2.2	49.74	9.83	2.46
dataset_2	64.22	12.78	3.2	19.55	15.3	3.83	21.74	9.95	2.49	46.83	11.13	2.78
dataset_3	83.76	14.76	3.69	25.45	17.67	4.42	28.44	11.49	2.87	60.95	12.86	3.21
Average	**77.125**	**12.9225**	**3.2325**	**23.4**	**15.475**	**3.8725**	**26.1175**	**10.06**	**2.5175**	**56.075**	**11.26**	**2.8125**
Standard Deviation (STDEV)	12.78200167	1.429367576	0.356218566	3.915448037	1.704200692	0.425940137	4.365244361	1.1101051	0.27439327	9.380207887	1.241423914	0.307394969
CONFIDENCE INTERVAL (CI95)	0.40075907	0.044815518	0.011168659	0.122762565	0.053432467	0.013354667	0.136865204	0.034805557	0.008603159	0.294101307	0.038922847	0.009637874
confidence range at 95%	77.52575907	12.96731552	3.243668659	23.52276257	15.52843247	3.885854667	26.2543652	10.09480556	2.526103159	56.36910131	11.29892285	2.822137874

Distribute all dataset into all clouds using ***Heuristic*** Approach												
Dataset Name	**Google Cloud Storage**			**Amazon**			**MS Azure**			**Roackspace**		
	Total Cost	Avg latency_r	Avg. Latency_w	Total Cost	Avg. latency_r	Avg. Latency_w	Total Cost	Avg. latency_r	Avg. Latency_w	Total Cost	Avg. latency_r	Avg. Latency_w
dataset_1	88.95	12.49	3.12	21.94	12.15	3.04	29.79	9.62	2.41	48.15	8.08	2.02
dataset_random	66.89	10.94	2.74	16.35	10.65	2.66	22.28	8.43	2.11	35.78	7.08	1.77
dataset_2	62.37	12.39	3.1	15.41	12.06	3.01	20.88	9.55	2.39	33.66	8.02	2
dataset_3	81.3	14.31	3.58	20.07	13.93	3.48	27.29	11.03	2.76	43.84	9.26	2.31
Average	**74.8775**	**12.5325**	**3.135**	**18.4425**	**12.1975**	**3.0475**	**25.06**	**9.6575**	**2.4175**	**40.3575**	**8.11**	**2.025**
Standard Deviation (STDEV)	12.37634134	1.380540353	0.34423829	3.079820071	1.343809882	0.335993552	4.185156309	1.065125188	0.26625489	6.798550703	0.893009892	0.221284131
CONFIDENCE INTERVAL (CI95)	0.388040244	0.043284619	0.010793037	0.096562797	0.042132994	0.010534537	0.131218834	0.033395284	0.008347993	0.213157605	0.027998886	0.006938007
confidence range at 95%	75.26554024	12.57578462	3.145793037	18.5390628	12.23963299	3.058034537	25.19121883	9.690895284	2.425847993	40.5706576	8.137998886	2.031938007

Table 8.7: A full statistical analysis of distributing all data sets using SRD, Heuristic code and OFDAMCSS frameworks. Where latency_r is latency of read and latency_w is latency of write

138

Table 8.9: Percentage change in the storage cost for the writing group

	#write=1	#write=2	#write=3	#write=4	#write=6	#write=7	#write=8	#write=9
Google	-41.98%	-44.32%	-44.68%	-47.93%	-47.93%	-47.93%	-48.84%	-47.09%
Amazon S3	80.65%	76.47%	80.56%	80.00%	80.00%	80.00%	77.27%	78.79%
MS Azure	32.97%	32.00%	33.02%	41.36%	41.36%	41.36%	39.18%	40.72%
RackSpace	-64.52%	-67.65%	-66.67%	-72.31%	-72.31%	-72.31%	-72.73%	-72.73%

Table 8.10: Percentage change in network cost for each file

	#write=1	#write=2	#write=3	#write=4	#write=6	#write=7	#write=8	#write=9
Google	-3.47%	-3.13%	-2.79%	-1.60%	-1.59%	-1.58%	-1.58%	-1.56%
Amazon S3	5.94%	5.43%	5.08%	2.62%	2.66%	2.65%	2.65%	2.61%
MS Azure	2.23%	2.32%	2.10%	1.34%	1.37%	1.38%	1.37%	1.36%
RackSpace	-5.20%	-4.80%	-4.18%	-2.42%	-2.40%	-2.40%	-2.39%	-2.41%

Table 8.11: Percentage change in the storage cost for the reading group

	#read=1	#read=2	#read=3	#read=4	#read=5	#read=6	#read=7	#read=8	#read=9	#read=10	#read=11	#read=12	#read=13	#read=14	#read=16
Google	-40.00%	0.00%	-43.37%	-44.32%	-46.15%	-45.54%	-44.95%	-48.00%	-47.10%	-47.30%	-48.45%	-47.93%	-48.84%	-48.84%	-48.84%
Amazon S3	50.00%	100.00%	65.63%	61.76%	65.71%	64.10%	66.67%	66.67%	67.92%	70.18%	69.35%	70.77%	68.18%	68.18%	68.18%
MS Azure	50.00%	50.00%	53.19%	50.00%	51.46%	48.70%	50.41%	50.35%	50.64%	52.38%	51.65%	52.36%	51.55%	51.55%	51.55%
RackSpace	-50.00%	-50.00%	-65.63%	-67.65%	-65.71%	-69.23%	-69.05%	-70.83%	-71.70%	-71.93%	-72.58%	-72.31%	-72.73%	-72.73%	-72.73%

Table 8.12: Percentage change in the network cost for the reading group

	#read=1	#read=2	#read=3	#read=4	#read=5	#read=6	#read=7	#read=8	#read=9	#read=10	#read=11	#read=12	#read=13	#read=14	#read=16
Google	-11.11%	-9.09%	-3.67%	-3.38%	-3.34%	-2.84%	-2.67%	-2.32%	-2.04%	-1.91%	-1.74%	-1.65%	-1.64%	-1.58%	-1.62%
Amazon S3	9.37%	17.95%	4.89%	4.76%	4.59%	4.15%	3.74%	165.45%	2.93%	2.67%	2.44%	2.37%	2.36%	2.34%	2.28%
MS Azure	0.00%	8.47%	3.91%	3.85%	3.57%	3.33%	2.81%	2.41%	2.19%	1.98%	1.80%	1.75%	1.75%	1.78%	1.72%
RackSpace	-11.11%	-9.09%	-5.38%	-5.18%	-5.01%	-4.34%	-3.88%	-3.43%	-3.02%	-2.80%	-2.61%	-2.42%	-2.41%	-2.39%	-2.43%

Table 8.13: Percentage change in the storage used for the writing group

	#write=1	#write=2	#write=3	#write=4	#write=6	#write=7	#write=8	#write=9
Google	-44.04%	-44.49%	-44.94%	-47.85%	-47.85%	-47.85%	-47.85%	-47.83%
Amazon S3	80.01%	80.26%	80.25%	79.45%	79.43%	79.44%	79.40%	79.37%
MS Azure	31.35%	32.40%	33.46%	40.95%	40.98%	41.00%	40.99%	41.06%
RackSpace	-67.38%	-68.14%	-68.72%	-72.56%	-72.56%	-72.57%	-72.57%	-72.58%

Table 8.14: Percentage change in the network used for the writing group

	#write=1	#write=2	#write=3	#write=4	#write=6	#write=7	#write=8	#write=9
Google	-3.36%	-3.09%	-2.79%	-1.59%	-1.59%	-1.59%	-1.58%	-1.57%
Amazon S3	6.12%	5.58%	4.99%	2.65%	2.64%	2.63%	2.63%	2.61%
MS Azure	2.40%	2.25%	2.08%	1.36%	1.36%	1.36%	1.36%	1.35%
RackSpace	-5.15%	-4.74%	-4.27%	-2.42%	-2.41%	-2.41%	-2.40%	-2.38%

Table 8.15: Percentage change in the storage used for the reading group

	#read=1	#read=2	#read=3	#read=4	#read=5	#read=6	#read=7	#read=8	#read=9	#read=10	#read=11	#read=12	#read=13	#read=14	#read=16
Google	-44.24%	-44.56%	-45.45%	-45.69%	-45.93%	-46.44%	-46.93%	-47.53%	-48.01%	-48.35%	-48.62%	-48.81%	-48.82%	-48.84%	-48.84%
Amazon S3	73.33%	73.06%	62.87%	63.42%	64.08%	65.31%	66.34%	67.63%	68.47%	69.05%	69.58%	69.91%	69.90%	69.91%	69.90%
MS Azure	43.03%	43.01%	50.27%	50.34%	50.32%	50.39%	50.59%	50.99%	51.39%	51.68%	52.02%	52.24%	52.26%	52.31%	52.34%
RackSpace	-72.73%	-72.54%	-67.72%	-68.14%	-68.43%	-69.21%	-70.00%	-71.12%	-71.87%	-72.41%	-72.97%	-73.35%	-73.37%	-73.39%	-73.40%

Table 8.16: Percentage change in the network used for the reading group

	#read=1	#read=2	#read=3	#read=4	#read=5	#read=6	#read=7	#read=8	#read=9	#read=10	#read=11	#read=12	#read=13	#read=14	#read=16
Google	-8.74 %	-8.00%	-3.56%	-3.45%	-3.35%	-2.96%	-2.63%	-2.27%	-2.05%	-1.88%	-1.73%	-1.63%	-1.63%	-1.62%	-1.62%
Amazon S3	14.68%	13.12%	4.93%	4.79%	4.67%	4.17%	3.71%	3.23%	2.92%	2.68%	2.48%	2.34%	2.33%	2.32%	2.31%
MS Azure	8.74%	7.72%	3.94%	3.80%	3.67%	3.21%	2.83%	2.44%	2.19%	2.01%	1.85%	1.75%	1.74%	1.74%	1.73%
RackSpace	-14.44%	-13.02%	-5.31%	-5.14%	-4.99%	-4.42%	-3.92%	-3.40%	-3.07%	-2.81%	-2.60%	-2.46%	-2.45%	-2.43%	-2.43%

Table 8.17: Percentage change in the latency time for the writing group

	#write=1	#write=2	#write=3	#write=4	#write=6	#write=7	#write=8	#write=9
Google	-93.44%	-92.33%	-108.27%	-78.55%	-78.40%	-108.85%	-78.40%	-77.86%
Amazon S3	46.99%	44.45%	44.46%	44.43%	41.83%	41.77%	40.40%	40.51%
MS Azure	24.38%	30.53%	34.20%	26.48%	32.45%	35.98%	34.20%	35.83%
RackSpace	-257.89%	-316.71%	-400.68%	-257.58%	-312.90%	-316.92%	-312.90%	-396.30%

Table 8.18: Percentage change in the latency time for the reading group

	#read=1	#read=2	#read=3	#read=4	#read=5	#read=6	#read=7	#read=8	#read=9	#read=10	#read=11	#read=12	#read=13	#read=14	#read=16
Google	-78.55%	-78.79%	-91.84%	-92.65%	-108.21%	-108.24%	-108.55%	-108.27%	-127.18%	-127.18%	-127.55%	-127.55%	-108.85%	-126.21%	-127.27%
Amazon S3	43.19%	41.98%	44.33%	41.85%	44.41%	43.16%	43.20%	44.46%	43.20%	43.20%	40.40%	40.40%	41.77%	40.51%	39.05%
MS Azure	30.59%	30.86%	30.71%	34.15%	32.36%	34.18%	35.82%	34.20%	35.88%	35.88%	40.52%	40.52%	37.63%	40.45%	41.71%
RackSpace	-316.47%	-257.89%	-319.23%	-319.44%	-316.96%	-315.31%	-400.00%	-400.68%	-316.71%	-316.71%	-401.96%	-401.96%	-401.85%	-396.30%	-398.08%

8.3.5 Parameter Effects in the OFDAMCSS Framework

As shown in Figure 8.11,in the reinforcement learning experiments different configurable parameter values were used in the learning algorithm and representation function, which were roughly estimated. Configuring the value of these parameters is a serious practical challenge in experiments on reinforcement learning. A common approach is to take a random selection of these parameters and then adjust them throughout the learning time.

This section reports on the effect of changing individual parameters in the learning algorithm and function representation. The effects of these changes on the convergence and performance of learning in this study were assessed. Six parameters were examined, with three (η, α, and β) being related to the artificial neural network and three (γ, λ and ϵ) to the learning algorithm. A descriptive account of the results of the parameter experiments is provided below.

- **Hidden nodes:** the changing number of hidden nodes seemed to have a minor impact on the learning ability of the network, as shown in Figure 8.12. First, the hidden nodes were set to 16 nodes, followed by an evaluation of 48 nodes and 96 nodes. The only effect of changing the number of hidden nodes was the speed of learning. With a high number of hidden nodes, the system learnt very slowly

- **First Learning rate (α):** the experiments showed that a low value of the learning rate α for the hidden output layer allowed the artificial neural network to converge faster than a high value. The results are shown in Figure 8.13.

- **Second Learning rate (β):** the experiments on the learning rate β for input-hidden layer showed that the network was unable to learn $\beta >= 0.1$. The experimental results also showed that the network converged slightly more effectively if the value of learning rate β was equal or less than the learning rate α value.

- **-Temporal discount factor (λ):** the experiments showed that the effect of using

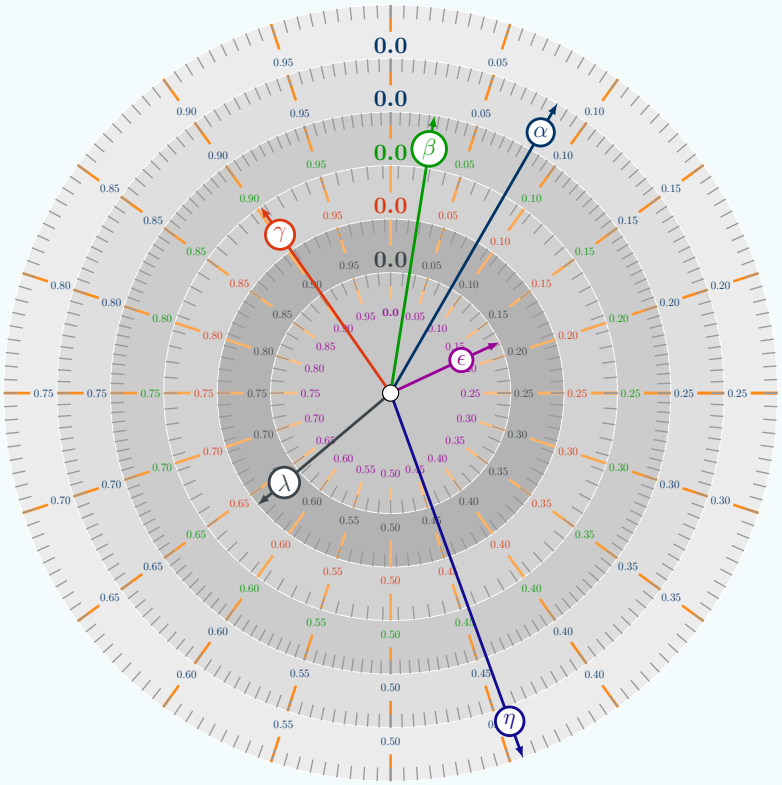


Figure 8.11: Nested meters for three artificial neural network parameters and three learning algorithms for reinforcement learning

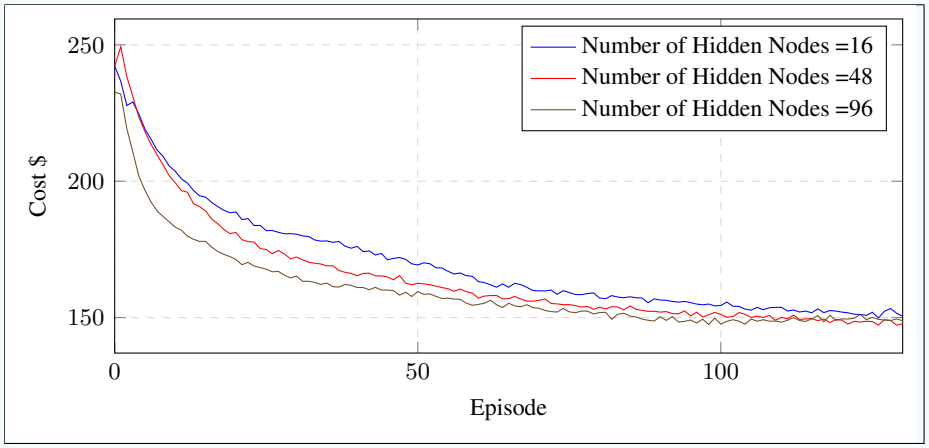
different λ was not profound (Figure 8.15). However, the network was unable to converge and learn when $\lambda = 1$. In addition, for cost optimisation, the experimental results showed that a high λ value was somewhat better than a low value.

- **Decay parameter (γ):** studying the effect of γ on system learning showed that medium and low values allowed the network to learn more effectively compared to when the values were high. The results are shown in Figure 8.16.

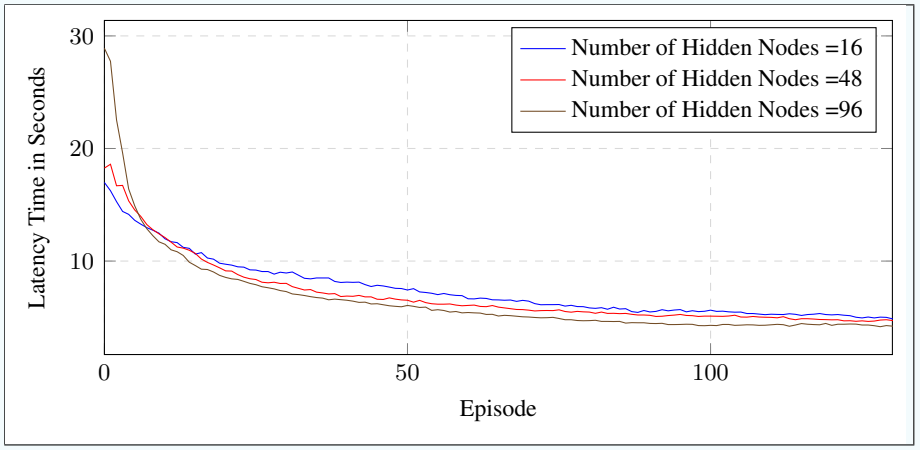
- **Momentum (η):** similar to λ, when $\eta = 1$ the network was unable to learn (Figure 8.17). However, there was no great impact on any other value of the momentum with a range of [0.0,0.9].

- **Epsilon (ϵ):** as shown in Figure 8.18, experimenting with different values of ϵ

seemed not to have any strong effect on learning.



(a) The effect of changing the number of hidden nodes on the cost

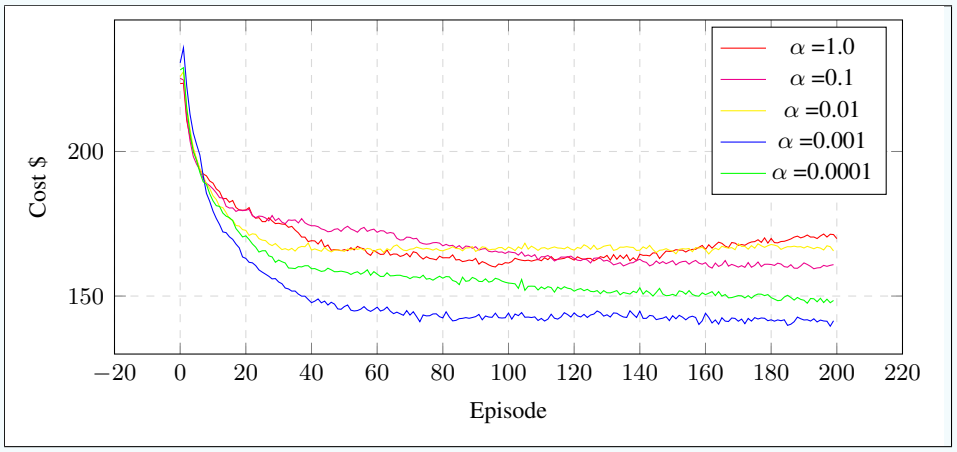


(b) The effect of changing the number of nodes on the latency time

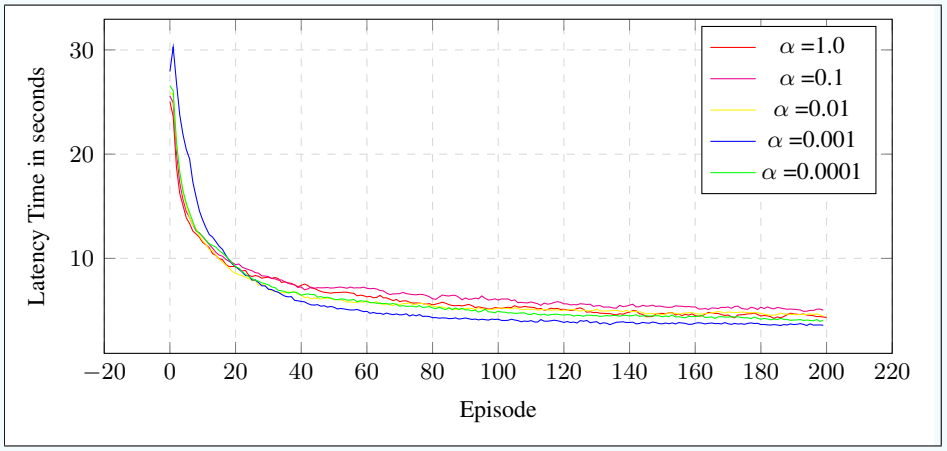
Figure 8.12: Evaluate the impact of changing the number of hidden nodes on the hidden layer

8.4 OFDAMCSS Framework Overheads

The experiments on the OFDAMCSS framework were conducted on MacBook Pro (processor 2.4 GHz Intel Core i7; memory 8 GB, 1600 MHz DDR3). The experimental code was written in JAVA using IntelliJ IDEA CE. All algorithms of the framework – namely the artificial neural network, back-propagation, all reinforcement learning algorithms,



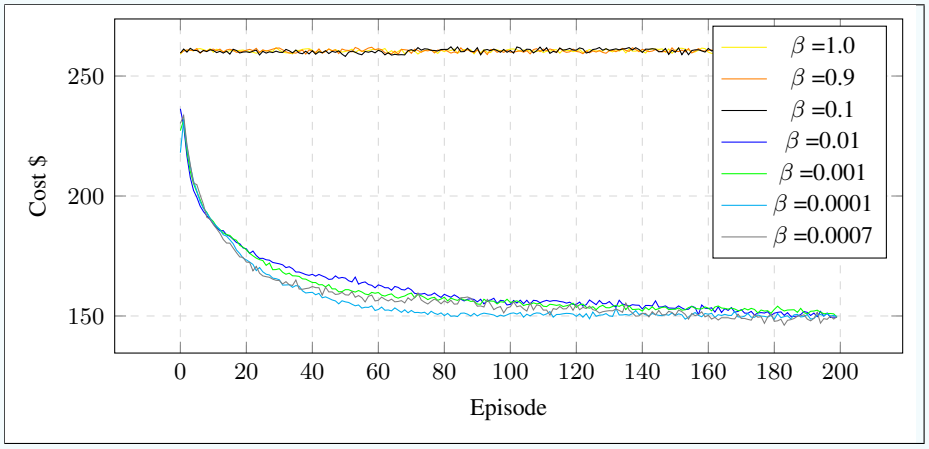
(a) The effect of α on cost optimisation



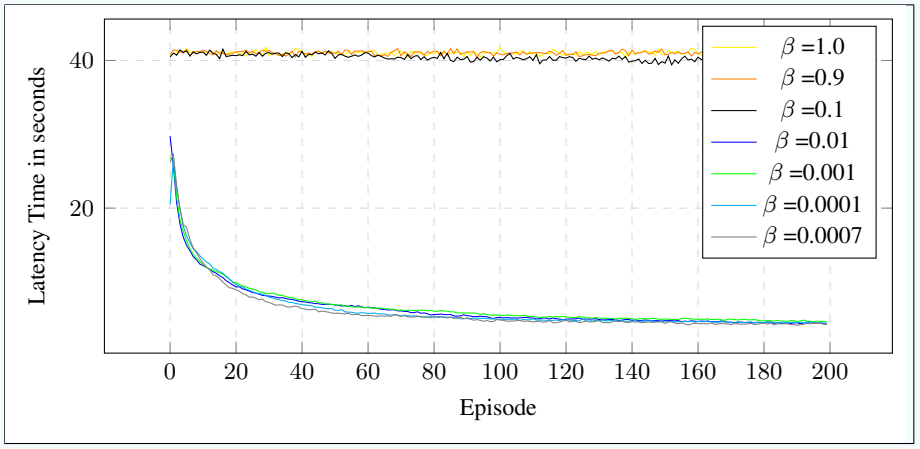
(b) The effect of α on latency optimisation

Figure 8.13: The impact of α on the performance of learning in the OFDAMCSS

and the cloud emulator and cloud monitor – were written from scratch in JAVA for this work. The study lasted about four months, including testing time. The experiment on the learning process was run several times, each time for a different dataset. Each experiment required roughly 3000 learning steps and 1000 episodes, which meant that the total for all learning steps was about 3 million steps.



(a) The effect of β on cost optimisation

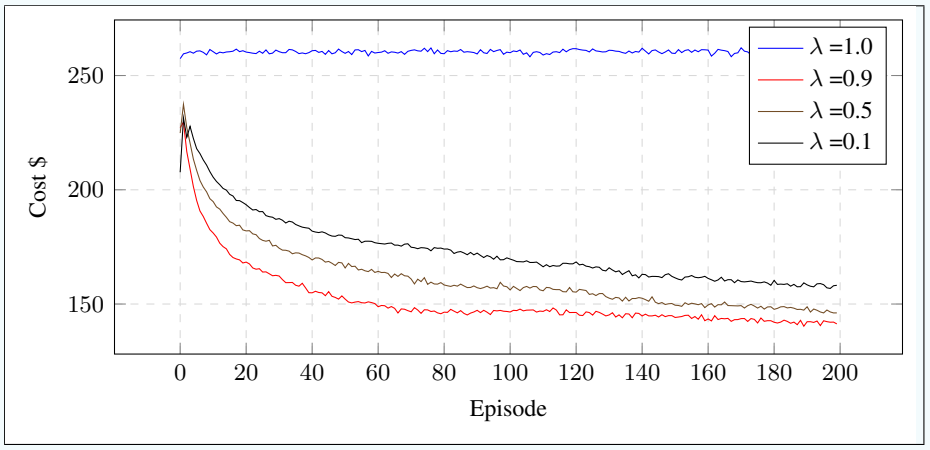


(b) The effect of β on latency optimisation

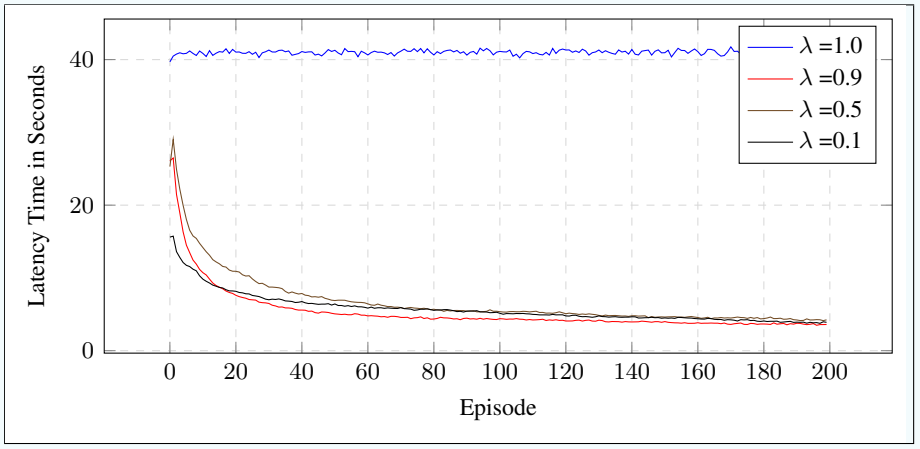
Figure 8.14: The impact of β on the performance of learning in the OFDAMCSS

8.5 Summary

This chapter has provided evidence that the proposed OFDAMCSS framework is capable of optimising cost and latency time for multiple cloud storage services. The experiments in this research showed that the cost of distributing files on multiple cloud storage services was reduced by up to 42% for certain clouds, compared with data distribution using the standard RAID (SRD) approach. Similarly, the proposed model outperformed the heuristic approach by about 24% for certain cloud services. When data were distrib-



(a) The effect of λ on cost optimisation

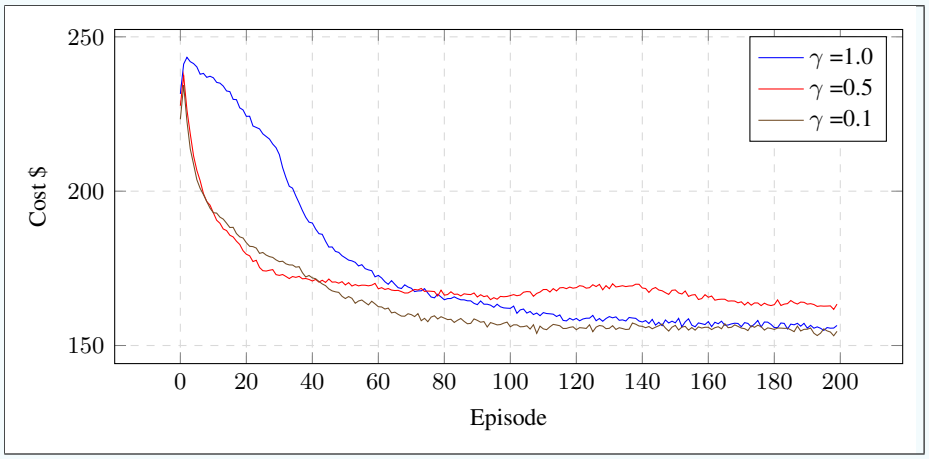


(b) The effect of λ on latency optimisation

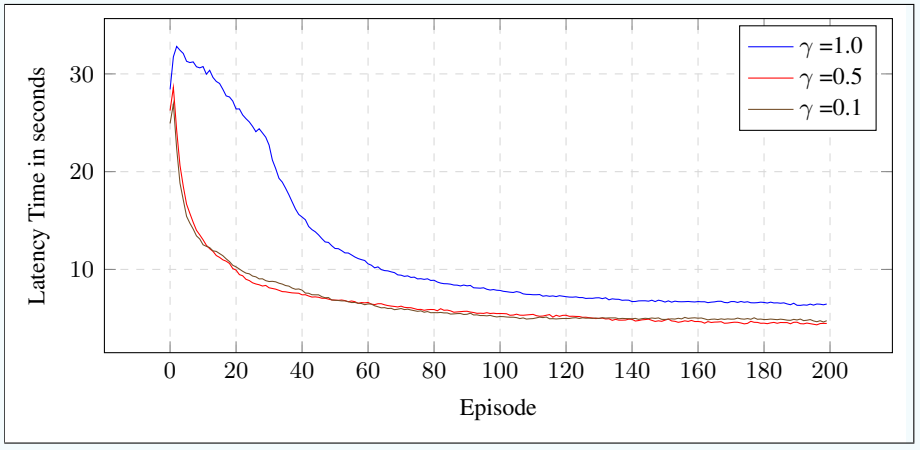
Figure 8.15: The impact of λ on the performance of learning in the OFDAMCSS

uted across four cloud storage services, the latency time decreased by about 76% when using OFDAMCSS compared with SRD, and by about 56% compared with the heuristic approach. The generative qualities of the framework were tested on multiple random cloud storage services. The results showed that the proposed framework could optimise cost and latency time on multiple cloud storage services.

Furthermore, this chapter reported on the effect of changing the learning parameters, namely the number of hidden nodes in the artificial neural network (α, β, λ, γ, η and ϵ) on the optimisation of cost and latency time. In brief, changing the number of hidden



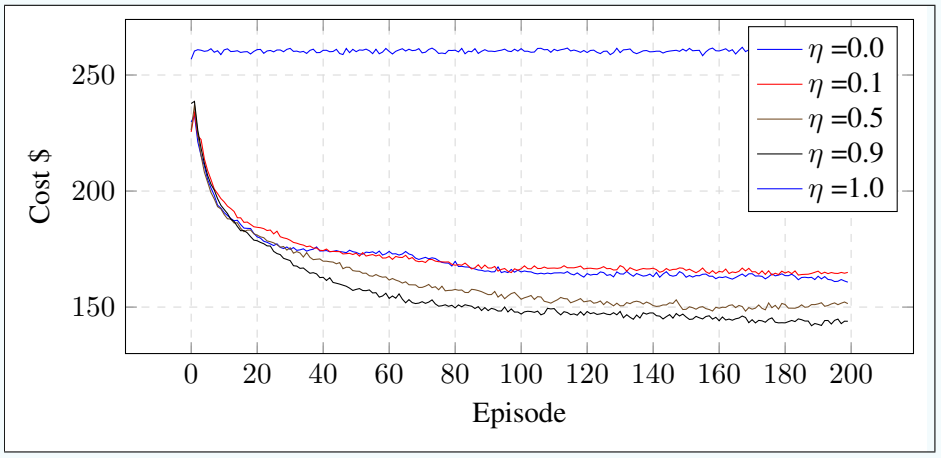
(a) The effect of γ on cost optimisation



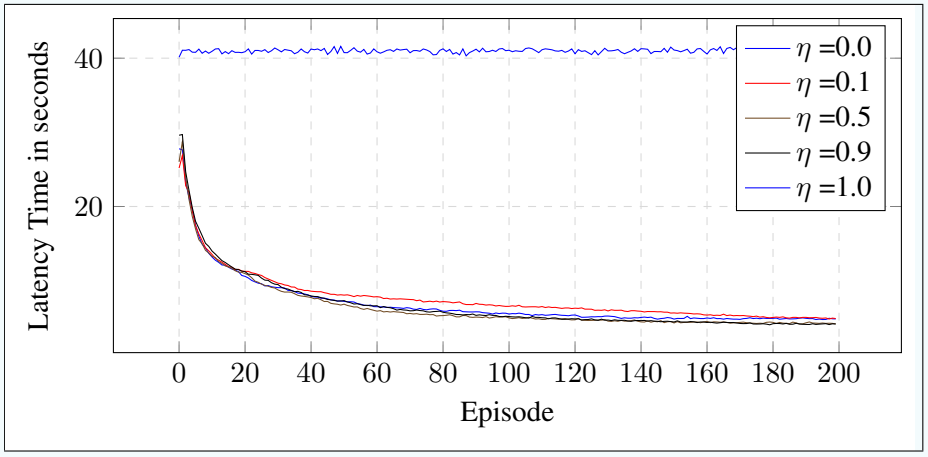
(b) The effect of γ on latency optimisation

Figure 8.16: The impact of γ on the performance of learning in the OFDAMCSS

node did not affect the quality of learning in the learning system, but it did affect the speed of learning. A small number of nodes was most effective; A small value of α was most effective; β should be less than or equal to α; γ should be between [0.0,0.7]; Changing the values of the remaining parameters did not have a notable effect on the performance of the machine learning system.

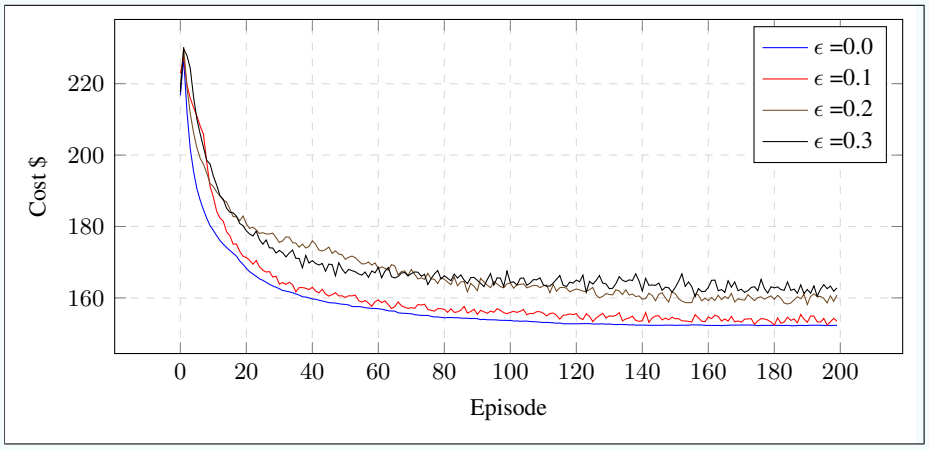


(a) The effect of η on cost optimisation

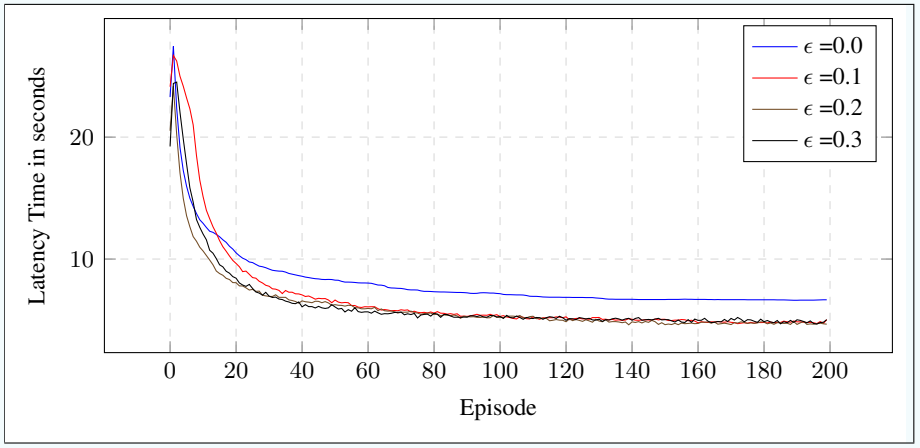


(b) The effect of η on latency optimisation

Figure 8.17: The impact of η on the performance of learning in the OFDAMCSS



(a) The effect of ϵ on cost optimisation



(b) The effect of ϵ on latency optimisation

Figure 8.18: The impact of ϵ on the performance of learning in the OFDAMCSS

CHAPTER 9

Conclusion and Future Work

This is the final chapter of the thesis, which addresses the application of machine learning algorithms in a dynamic environment. The research was focused on optimising two values that were continuous and non-stationary (cost and latency time). This work should be considered a start towards the goal of better optimisation of multiple factors in a dynamic environment such as cloud computing. This chapter gives an outline of the problem addressed in this research and how it was solved, including a synopsis of the thesis contribution. Finally, limitations of the research are outlined and recommendations for future work are provided.

9.1 Summary of the Thesis

This thesis presents an intelligent framework for automatically tuning distribution parameters over multiple cloud storage services to optimise long-term cost and latency time. The work started by providing an overview of cloud storage services, distinguishing these from what is sometimes called 'drive', as explained in Chapter 2. Moreover, Chapter 2 provided insight into the issue of storing data on a single cloud storage service.

In addition, it provided a review and discussion of the limitations of different solutions and introduced a proposed solution to solve those issues..

Chapter 3 provided an overview of the machine learning field. The framework used in this research was the product of combining two machine learning paradigms: supervised learning, to predict the access patterns for each file; and reinforcement learning, for tuning the distribution parameters based on the predicted access patterns.

Chapter 4 provided a survey of the use of machine learning algorithms in different cloud computing storage problems. Chapters 5,6,7, and 8provided empirical evidence of the benefits of the proposed framework. The results of experiments involving a cloud storage emulator were presented in Chapter 8. The main challenges in this research were how to interact with multiple environments by executing a non-fixed number of actions simultaneously, and how to deal with numerous 'non-stationary' reward signals. Therefore, the learning algorithm in reinforcement learning was designed in a novel way to satisfy the research goal. The results show the proposed framework is capable of significantly reducing both the cost and average latency time over multiple cloud storage services.

The generative parameters of the proposed framework was assessed through several experiments based on a random setting of cloud performance and pricing scheme, to test the durability of the framework. In these experiments, the settings of the cloud emulator were generated randomly between the maximum and minimum performances that were collected from CloundHarmony.com. The pricing schemes were generated based on the highest and lowest prices of Google Cloud Storage, Amazon S3, MS Azure Storage, and RackSpace Cloud File. The reason behind the arbitrary settings was that the market did not have an abundance of cloud storage services when this research was conducted. The generalisation test results showed that the framework optimises the cost and latency time in various clouds, with a minimum of three clouds; the maximum tested was six clouds. Finally, several experiments were conducted to explore the effect of parameter reinforcement learning and an artificial neural network on the learning behaviour in the framework. In this study, cost was calculated based on several small amount of syn-

thetic datasets; real-world organisations have much larger amounts of data to store in cloud services. For that reason, the cost and potential savings for organisations would be considerably higher than the amounts of data and cost calculated in this work.

There are many points to consider as limitations when assessing this research. Some of these points are discussed in the following section.

9.2 Novel Contributions of this Work

This work suggests several innovations in the fields of both cloud computing and reinforcement learning. Below is a list of the innovations offered in this work and whether each contribution is related to cloud computing (CC), reinforcement learning (RL), or supervised learning (SL).

- **Novel intelligent framework for distribution of files (CC):** 1. To our knowledge, this work is the first to apply machine learning algorithms to optimise cost and performance (latency time) for cloud storage services.

  The distinguishing characteristics of specific cloud providers, in terms of pricing schemes and service performance, make optimisation of both cost and performance at once – across multiple cloud storage services – a challenging matter. Each cloud provider has its own data centre architecture, access policies, and storing methodologies; this diversity affects performance when files are distributed across several cloud services. Furthermore, cost and latency are measured in different ways: cost in money, and latency in time. However, the researcher found a method to address these differences and difficulties, to provide a framework capable of adapting to multiple cloud storage services. The proposed framework can optimise both performance and cost together.

- **A new reinforcement learning approach to produce actions from state values (RL) :** 1. This work presents a new action policy to produce multiple actions sim-ultaneously in multiple environments, where the number of actions and number of environments at each time step are neither deterministic nor stationary.

Furthermore, each action has a continuous value. This new approach allows the reinforcement learning system to do the following:

1. Perform several continuous actions simultaneously. The proposed approach allowed the learning system to interact with several cloud storage services and allowed the neural network to produce unknown actions. This work introduced experiments in reinforcement learning based on a non-fixed number of output nodes in an artificial neural network. The results show that reinforcement learning systems can learn without any negative effect from changing the number of output nodes during the learning time.
2. Produce the action value from the state value. The system turned each state value into an action.

- **Provided a list of hints and tips (RL)** to help researchers understand how an artificial neural network can be used as a function approximator within a reinforcement learning system. .
- **Provided a synthetic dataset of file log trace (SL)**. This work generated a synthetic dataset that emulated the access behaviour of each file. This dataset was used to train a supervised learning algorithm to allow the framework to predict numerical attributes of the access pattern, based on a realistic business model. The attributes included the file's active lifespan, the number of reads, and the number of writes.
- **Numerical prediction of file access pattern attributes (SL)**. To our knowledge, this research is the first to introduce a prediction of the numerical value of file access patterns.

9.3 Limitations and Future Work

As mentioned above, this work is about distributing files across various cloud storage services, with each cloud imposing a unique performance and having a unique pricing

scheme. Furthermore, this research is the first to use machine learning to optimise cost and latency time across multiple cloud services, from the client's side. Although this thesis offers many novelties and contributions, there are several points to consider when reading it, listed below.

- Reward time delay. One of the important points to be considered is the time delay between each action and its respective reward. More specifically, in the real world when a file is distributed across multiple online storage services, it takes time to completely store all the file fragments and then compute the reward function. During this time, it is possible for many files to be queued ready for distribution. Thus, the time delay might affect the speed of learning – or worse, the quality of distribution. This issue is unrelated to the proposed framework specifically. However, in real-world reinforcement learning applications, rewards might be delayed in time. Such a delay can have different causes, including unpredictable network latency or poor quality of some sensors. [Campbell 2014] highlighted this problem in detail, with some real-world applications. This problem as it relates to OFDAMCSS requires further study .

- RAID. As described in Section 2.5, RAID is a technology that has several methods (or levels) for distributing files over several hard disks. Some of these levels, including levels 5 and 6, work by striping files uniformly over all the disks with parity distribution. In addition, RAID Level 10 has different methodology from RAID levels 5 and 6; in Level 10, the disks are divided into two groups at least and then the whole file size is sent to each group. Inside each group, the file is fragmented uniformly and then each fragment is stored on a different disk. However, the proposed framework used a varied distribution of file fragments to each cloud storage service. This means there is a need for an investigation of how to modify standard RAID methods to write different sizes of the file to each cloud.

- Learning parameters effect. Further experiments are needed to understand why the parameters of reinforcement learning and artificial neural networks result in

certain effects. As shown in this study, the parameters do affect learning behaviour in different ways. A suggestion for future research is to explore the impact of changing the parameters, as well as studying the dependence of parameters on each other.

- Multi-objective optimisation. This work is based on the trade-off between different objectives so that they do not conflict with each other. That is, one objective value can be optimised or improved but without degrading another objective value. The research approached this challenge by combining all objectives into a scalar single objective. However, further studies could examine different multi-objective optimisation techniques, including Pareto fronts and multi-agent multi-objective optimisation. Some potential exists for minor conflict between the objective values.

  - Pareto optimality, also known as 'Pareto efficiency', is a method that solves a multi-objective optimisation problem. This method received great acclaim in the field of machine learning due to its success in optimising multiple objectives using evolutionary algorithms and other population-based stochastic search methods [Jin & Sendhoff 2008]. In studies such as the current research, Pareto optimality can be examined to optimise the total cost of using multiple cloud storage services, without adversely affecting the performance (latency time). More information about applying Pareto optimal solutions to machine learning systems to optimise multiple objectives appears in [Van Moffaert & Nowé 2014],[Mukhopadhyay et al. 2014], [Zuluaga et al. 2016], and Fan et al. [2016].
  - Multiple machine learning systems solution. This work addressed the problem of interacting with multiple environments and optimising multiple objectives in each environment. This approach is also known as 'multi-agent systems'. It can be examined to solve the problem of file distribution across multiple cloud storage services. In this approach, each machine learning

system can have individual MDPs, each of which corresponds to individual cloud storage. Here, all machine learning systems require a level of cooperation to ensure the sum of file portions is equal to 100% of the file's original size. More details about this approach appear in [Panait & Luke 2005], [Nguyen et al. 2010], and [Abouheaf et al. 2012].

References

Springer (2005). *Function approximation via tile coding: Automating parameter choice*. Springer, Springer.

(2017). Definition of latency in english. https://en.oxforddictionaries.com/thesaurus/worsen. Accessed: 2017-06-08.

(2017). Latency. http://whatis.techtarget.com/definition/latency. Accessed: 2017-06-08.

Abouheaf, M., Dissertations, P., resource collection), T. E., & of Texas at Arlington. College of Engineering, U. (2012). *Optimization and Reinforcement Learning Techniques in Multi-agent Graphical Games and Economic Dispatch*.

Abu-Libdeh, H., Princehouse, L., & Weatherspoon, H. (2010). RACS: A case for cloud storage diversity. In *Proceedings of the 1st ACM Symposium on Cloud Computing*, SoCC '10, (pp. 229–240)., New York, NY, USA. ACM.

Aggarwal, C. C. (2015). *Data classification: algorithms and applications*. CRC Press.

Agrawal, N., Bolosky, W. J., Douceur, J. R., & Lorch, J. R. (2007). A five-year study of file-system metadata. *Trans. Storage*, *3*(3).

Alzain, M. A., Soh, B., & Pardede, E. (2011). MCDB: Using multi-clouds to ensure security in cloud computing. In *Dependable, Autonomic and Secure Computing (DASC), 2011 IEEE Ninth International Conference on*, (pp. 784–791).

Armbrust, M., Fox, A., Griffith, R., Joseph, A. D., Katz, R., Konwinski, A., Lee, G., Patterson, D., Rabkin, A., Stoica, I., & Zaharia, M. (2010). A view of cloud computing. *Commun. ACM*, *53*, 50–58.

Barrett, E., Howley, E., & Duggan, J. (2013). Applying reinforcement learning towards automating resource allocation and application scalability in the cloud. *Concurrency and Computation: Practice and Experience*, *25*(12), 1656–1674.

Bell, J. (2015). *Machine Learning: Hands-On for Developers and Technical Professionals* (1st ed.). Indianapolis, IN, USA: John Wiley & Sons.

Beloglazov, A., Buyya, R., Lee, Y. C., Zomaya, A., et al. (2011). *A taxonomy and survey of energy-efficient data centers and cloud computing systems*, volume 82, (pp. 47–111). Academic Press.

Bertsekas, D. P. (1987). *Dynamic Programming: Deterministic and Stochastic Models*. Upper Saddle River, NJ, USA: Prentice-Hall, Inc.

Bessani, A., Correia, M., Quaresma, B., Andr, F., & Sousa, P. (2011). Depsky: Dependable and secure storage in a cloud-of-clouds. In *Proceedings of the Sixth Conference on Computer Systems*, EuroSys '11, (pp. 31–46)., New York, NY, USA.

Birkes, D. & Dodge, Y. (2011). *Alternative methods of regression*, volume 190. John Wiley & Sons.

Bishop, C. M. (2006). *Pattern Recognition and Machine Learning (Information Science and Statistics)*. Secaucus, NJ, USA: Springer-Verlag New York, Inc.

Bort, J. (2016). Google apologizes for cloud outage that one person describes as a comedy of errors. Online; accessed 15-05-2016.

Bowers, K. D., Juels, A., & Oprea, A. (2009). Hail: A high-availability and integrity layer for cloud storage. In *Proceedings of the 16th ACM Conference on Computer and Communications Security*, CCS '09, (pp. 187–198)., New York, NY, USA. ACM.

Brinkmann, M. (2016). Copy cloud storage service's life ends on May 1, 2016. http://www.ghacks.net/2016/02/02/copy-cloud-storage-services-life-ends-on-may-1-201. Online; accessed 19-10-2016.

Bu, X., Rao, J., & Xu, C.-Z. (2011). A model-free learning approach for coordinated

configuration of virtual machines and appliances. In *2011 IEEE 19th Annual International Symposium on Modelling, Analysis, and Simulation of Computer and Telecommunication Systems*, (pp. 12–21). IEEE.

Buyya, R., Cortes, T., & Jin, H. (2001). *A Case for Redundant Arrays of Inexpensive Disks (RAID)*, (pp. 2–14). Wiley-IEEE Press.

Campbell, J. S. (2014). *Multiple Model Reinforcement Learning for Environments with Poissonian Time Delays*. PhD thesis, Carleton University Ottawa.

Chen, H., Kesavan, M., Schwan, K., Gavrilovska, A., Kumar, P., & Joshi, Y. (2011). Spatially-aware optimization of energy consumption in consolidated data center systems. In *ASME 2011 Pacific Rim Technical Conference and Exhibition on Packaging and Integration of Electronic and Photonic Systems*, (pp. 461–470). American Society of Mechanical Engineers.

Chen, T. & Bahsoon, R. (2013). Self-adaptive and sensitivity-aware QoS modeling for the cloud. In *Proceedings of the 8th International Symposium on Software Engineering for Adaptive and Self-Managing Systems*, (pp. 43–52)., Piscataway, NJ, USA. IEEE Press.

Choi, H. W., Kwak, H., Sohn, A., & Chung, K. (2008). Autonomous learning for efficient resource utilization of dynamic vm migration. In *Proceedings of the 22nd Annual International Conference on Supercomputing*, (pp. 185–194)., New York, NY, USA. ACM.

Cioara, T., Anghel, I., Salomie, I., Copil, G., Moldovan, D., & Kipp, A. (2011). Energy aware dynamic resource consolidation algorithm for virtualized service centers based on reinforcement learning. In *2011 10th International Symposium on Parallel and Distributed Computing*, (pp. 163–169). IEEE.

Cohen, J., Cohen, P., West, S. G., & Aiken, L. S. (2013). *Applied multiple regression/-correlation analysis for the behavioral sciences*. Routledge.

Crites, R. H. & Barto, A. G. (1994). An actor/critic algorithm that is equivalent to q-learning. In *Advances in Neural Information Processing Systems 7, [NIPS Conference, Denver, Colorado, USA, 1994]*, (pp. 401–408).

Dabbagh, M., Hamdaoui, B., Guizani, M., & Rayes, A. (2014). Energy-efficient cloud resource management. In *INFOCOM Workshops*, (pp. 386–391). IEEE.

Demirci, M. (2015). A survey of machine learning applications for energy-efficient resource management in cloud computing environments. In *2015 IEEE 14th International Conference on Machine Learning and Applications (ICMLA)*, (pp. 1185–1190).

Duda, R. O., Hart, P. E., & Stork, D. G. (2000). *Pattern Classification* (2nd ed.). Wiley-Interscience.

Dutreilh, X., Kirgizov, S., Melekhova, O., Malenfant, J., Rivierre, N., & Truck, I. (2011). Using reinforcement learning for autonomic resource allocation in clouds: towards a fully automated workflow. In *ICAS 2011, The Seventh International Conference on Autonomic and Autonomous Systems*, (pp. 67–74).

Eggert, C., Winschel, A., & Lienhart, R. (2015). On the benefit of synthetic data for company logo detection. In *Proceedings of the 23rd ACM International Conference on Multimedia*, (pp. 1283–1286)., New York, NY, USA. ACM.

Embrechts, M. J., Hargis, B. J., & Linton, J. D. (2010). Augmented efficient backprop for backpropagation learning in deep autoassociative neural networks. In *The 2010 International Joint Conference on Neural Networks (IJCNN)*, (pp. 1–6).

Erl, T., Mahmood, Z., & Puttini, R. (2013). *Cloud Computing: Concepts, Technology & Architecture*. USA, Upper Saddle River, New Jersey: Prentice Hall.

Fan, J. & Gijbels, I. (1996). *Local polynomial modelling and its applications: monographs on statistics and applied probability 66*, volume 66. CRC Press.

Fan, Z., Hu, K., Li, F., Rong, Y., Li, W., & Lin, H. (2016). Multi-objective evolutionary algorithms embedded with machine learning—a survey. In *Evolutionary Computation (CEC), 2016 IEEE Congress on*, (pp. 1262–1266). IEEE.

Flach, P. (2012). *Machine Learning: The Art and Science of Algorithms That Make Sense of Data*. UK: Cambridge University Press.

Frnkranz, J., Gamberger, D., & Lavrac, N. (2012). *Foundations of Rule Learning*. Springer Publishing Company, Incorporated.

Furht, B. (2010). *"Cloud Computing Fundamentals"*, (pp. 3–20). Berlin,Germany:

Springer.

Gatti, C. (2015). *Design of experiments for reinforcement learning*. Heidelberg New York Dordrecht London: Springer.

Gatti, C. J. & Embrechts, M. J. (2013). *Reinforcement Learning with Neural Networks: Tricks of the Trade*, (pp. 275–310). Berlin, Heidelberg: Springer Berlin Heidelberg.

Grondman, I. (2015). *Online Model Learning Algorithms for Actor-Critic Control*. Ivo Grondman.

Grondman, I., Busoniu, L., Lopes, G. A. D., & Babuska, R. (2012). A survey of actor-critic reinforcement learning: Standard and natural policy gradients. *IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews)*, *42*, 1291–1307.

Hans, C. (2011). Elastic net regression modeling with the orthant normal prior. *Journal of the American Statistical Association*, *106*(496), 1383–1393.

Hastie, T., Tibshirani, R., & Friedman, J. (2009). *The elements of statistical learning* (2nd ed.). Springer series in statistics Springer, Berlin.

Haykin, S. (2007). *Neural Networks: A Comprehensive Foundation* (3rd ed.). Upper Saddle River, NJ, USA: Prentice-Hall, Inc.

Heidrich-Meisner, V., Lauer, M., Igel, C., & Riedmiller, M. A. (2007). Reinforcement learning in a nutshell. In *ESANN 2007, 15th European Symposium on Artificial Neural Networks, Bruges, Belgium, April 25-27, 2007, Proceedings*, (pp. 277–288).

Hosmer Jr, D. W. & Lemeshow, S. (2004). *Applied logistic regression*. John Wiley & Sons.

Jamshidi, P., Sharifloo, A. M., Pahl, C., Metzger, A., & Estrada, G. (2015). Self-learning cloud controllers: Fuzzy q-learning for knowledge evolution. In *Cloud and Autonomic Computing (ICCAC), 2015 International Conference on*, (pp. 208–211).

Jclouds, A. (2016). The Java multi-cloud toolkit. http://jclouds.apache.org. Online; accessed 08-07-2014.

Jin, Y. & Sendhoff, B. (2008). Pareto-based multiobjective machine learning: An overview and case studies. *IEEE Transactions on Systems, Man, and Cybernetics, Part C*

(Applications and Reviews), *38*(3), 397–415.

Johnson, A. (2009). *Data Storage - peripheral view* (2nd ed.). alan@johnson.org: Self-publishing.

Kaelbling, L. P., Littman, M. L., & Moore, A. W. (1996). Reinforcement learning: A survey. *J. Artif. Int. Res.*, *4*(1), 237–285.

Kajiura, Y., Kanai, A., Tanimoto, S., & Sato, H. (2013). A file-distribution approach to achieve high availability and confidentiality for data storage on multi-cloud. In *2013 IEEE 37th Annual Computer Software and Applications Conference Workshops*, (pp. 212–217).

Kajiura, Y., Ueno, S., Kanai, A., Tanimoto, S., & Sato, H. (2015). An approach to selecting cloud services for data storage in heterogneous-multicloud environment with high availability and confidentiality. In *2015 IEEE Twelfth International Symposium on Autonomous Decentralized Systems*, (pp. 205–210).

Kanai, A., Kikuchi, N., Tanimoto, S. S., & Sato, H. (2014). Data management approach for multiple clouds using secret sharing scheme. In *2014 17th International Conference on Network-Based Information Systems*, (pp. 432–437).

Karpathy, A. (2014). Reinforcejs. Online; accessed 03-12-2016.

Khanna, R. & Awad, M. (2015). *Efficient learning machines: theories, concepts, and applications for engineers and system designers*. Apress.

Konda, V. R. & Tsitsiklis, J. N. (1999). Actor-critic algorithms. In *Advances in Neural Information Processing Systems 12, [NIPS Conference, Denver, Colorado, USA, November 29 - December 4, 1999]*, (pp. 1008–1014).

Konen, W. & Bartz-Beielstein, T. (2008). *Reinforcement Learning: Insights from Interesting Failures in Parameter Selection*, (pp. 478–487). Berlin, Heidelberg: Springer Berlin Heidelberg.

Kramer, O. (2013). *Dimensionality Reduction with Unsupervised Nearest Neighbors*. Berlin, Germany: Springer-Verlag Berlin Heidelberg.

Kundu, S., Rangaswami, R., Gulati, A., Zhao, M., & Dutta, K. (2012). Modeling virtualized applications using machine learning techniques. In *Proceedings of the 8th ACM*

SIGPLAN/SIGOPS Conference on Virtual Execution Environments, (pp. 3–14)., New York, NY, USA. ACM.

Lange, S., Gabel, T., & Riedmiller, M. (2012). *"Batch Reinforcement Learning"*, chapter 2, (pp. 46–73). Berlin, Heidelberg: Springer Berlin Heidelberg.

LeMay, R. (2013). Telstra's cloud computing suffers 24 hour outage. Online; accessed 03-12-2016.

Lewis, G. (2013). Standards in cloud computing interoperability. https://insights.sei.cmu.edu/sei_blog/2013/03/standards-in-cloud-computing-interoperability.html. Online; accessed 08-12-2016.

Li, F., Yang, Y., & Xing, E. P. (2005). From lasso regression to feature vector machine. In *Advances in Neural Information Processing Systems*, (pp. 779–786).

Liang, Y. (2016). *Towards a Standardized Quality Assessment Framework for OCCI-Controlled Cloud Infrastructures*, (pp. 58–73). Cham: Springer International Publishing.

Libclouds, A. (2016). One interface to rule them all. https://libcloud.apache.org. Online; accessed 08-07-2014.

Linden, J. (2012). *"TwinStrata Publishes A Snapshot into Cloud Storage Adoption from 2012 Cloud Computing Expo"*. TwinStrata.

Liu, S., Huang, X., Fu, H., & Yang, G. (2013). Understanding data characteristics and access patterns in a cloud storage system. In *2013 13th IEEE/ACM International Symposium on Cluster, Cloud, and Grid Computing*, (pp. 327–334).

Marshall, D. (2013). Cloud storage provider nirvanix is closing its doors. http://www.infoworld.com/article/2612299/cloud-storage/cloud- storage-provider-nirvanix-is-closing-its-doors.html. Online; accessed 19-10-2016.

Marsland, S. (2015). *Machine Learning: An Algorithmic Perspective* (2nd ed.). Chapman & Hall/CRC.

Matiisen, T. (2015). Guest post (part i): Demystifying deep reinforcement learning. Online; accessed 03-12-2016.

McCarthy, J. & Feigenbaum, E. (1990). In memoriam—arthur samuel

(1901–1990). *AI Mag.*, *11*(3), 10–11.

McClendon, L. & Meghanathan, N. (2015). Using machine learning algorithms to analyze crime data. *Machine Learning and Applications: An International Journal (MLAIJ)*, *2*(1).

Mell, P. M. & Grance, T. (2011). Sp 800-145. the nist definition of cloud computing. Technical report, National Institute of Standards & Technology, Gaithersburg, MD, United States.

Mesnier, M., Thereska, E., Ganger, G. R., Ellard, D., & Seltzer, M. (2004). File classification in self-* storage systems. In *Proceedings of the First International Conference on Autonomic Computing (ICAC-04)*, (pp. 44–51). IEEE.

Mitchell, T. M. (1997). *Machine Learning* (1st ed.). New York, NY, USA: McGraw-Hill, Inc.

Mnih, V., Kavukcuoglu, K., Silver, D., Graves, A., Antonoglou, I., Wierstra, D., & Riedmiller, M. A. (2013). Playing atari with deep reinforcement learning. *Computing Research Repository*.

Mnih, V., Kavukcuoglu, K., Silver, D., Rusu, A. A., Veness, J., Bellemare, M. G., Graves, A., Riedmiller, M., Fidjeland, A. K., Ostrovski, G., et al. (2015). Human-level control through deep reinforcement learning. *Nature*, *518*(7540), 529–533.

Mohri, M., Rostamizadeh, A., & Talwalkar, A. (2012). *Fundation of Machine Learning* (1st ed.). London, UK: Massachusetts Institute of Technology Press.

Montgomery, D. C., Peck, E. A., & Vining, G. G. (2015). *Introduction to linear regression analysis* (5th ed.). John Wiley & Sons.

Mu, S., Chen, K., Gao, P., Ye, F., Wu, Y., & Zheng, W. (2012). µlibcloud: Providing high available and uniform accessing to multiple cloud storages. In *Grid Computing (GRID), 2012 ACM/IEEE 13th International Conference on*, (pp. 201–208)., Beijing. IEEE.

Mukhopadhyay, A., Maulik, U., Bandyopadhyay, S., & Coello, C. A. C. (2014). A survey of multiobjective evolutionary algorithms for data mining: Part i. *IEEE Transactions on Evolutionary Computation*, *18*(1), 4–19.

Murphy, K. P. (2012). *Machine Learning: A Probabilistic Perspective*. The MIT Press.

Naqa, I. E. & Murphy, M. J. (2015). *"What Is Machine Learning"*, (pp. 3–12). Switzerland,Cham: Springer International Publishing.

Ng, A. Y., Parr, R., & Koller, D. (1999). Policy search via density estimation. In *Advances in Neural Information Processing Systems 12, [NIPS Conference, Denver, Colorado, USA, November 29 - December 4, 1999]*, (pp. 1022–1028).

Nguyen, N.-T., Howlett, R. J., & Jain, L. C. (2010). *Agent and multi-agent systems: technologies and applications*. Springer.

Nikolaev, N. & Iba, H. (2006). *Adaptive Learning of Polynomial Networks*. New York, NY, USA: Springer-Verlag New York, Inc.

Nonnemaker, J. E. (2008). *The Safe Use of Synthetic Data in Classification*. PhD thesis, Computer Science & Engineering, Bethlehem, PA, USA. AAI3358110.

Panait, L. & Luke, S. (2005). Cooperative multi-agent learning: The state of the art. *Autonomous agents and multi-agent systems*, *11*(3), 387–434.

Papaioannou, T. G., Bonvin, N., & Aberer, K. (2012). Scalia: An adaptive scheme for efficient multi-cloud storage. In *High Performance Computing, Networking, Storage and Analysis (SC), 2012 International Conference for*, (pp. 1–10).

Paraiso, F., Merle, P., & Seinturier, L. (2016). socloud: a service-oriented component-based paas for managing portability, provisioning, elasticity, and high availability across multiple clouds. *Computing*, *98*(5), 539–565.

Patterson, D. A., Gibson, G., & Katz, R. H. (1988). A case for redundant arrays of inexpensive disks (raid). In *Proceedings of the 1988 ACM SIGMOD International Conference on Management of Data*, SIGMOD '88, (pp. 109–116)., New York, NY, USA. ACM.

Pettey, C. & Goasduff, L. (2008). *"Gartner Highlights Five Attributes of Cloud Computing"*. Gartner, Inc. Online ; accessed 11-06-2016).

Pettey, C. & van der Meulen, R. (2008). *"Gartner Says Contrasting Views on Cloud Computing Are Creating Confusion"*. Gartner, Inc. Online ; accessed 11-06-2016).

Prevost, J. J., Nagothu, K., Kelley, B., & Jamshidi, M. (2011). Prediction of cloud

data center networks loads using stochastic and neural models. In *System of Systems Engineering (SoSE), 2011 6th International Conference on*, (pp. 276–281).

Rao, J., Bu, X., Xu, C.-Z., Wang, L., & Yin, G. (2009). Vconf: a reinforcement learning approach to virtual machines auto-configuration. In *Proceedings of the 6th international conference on Autonomic computing*, (pp. 137–146). ACM.

Raschka, S. (2015). *Python Machine Learning*. Birmingham , UK: Packt Publishing Ltd.

Rebello, J. (2012). Subscriptions to cloud storage services to reach half-billion level this year.

Riedmiller, M. (2005). Neural fitted q iteration–first experiences with a data efficient neural reinforcement learning method. In *European Conference on Machine Learning*, (pp. 317–328). Springer.

Rokach, L. & Maimon, O. (2014). *Data Mining With Decision Trees: Theory and Applications* (2nd ed.). River Edge, NJ, USA: World Scientific Publishing Co., Inc.

Salimian, L. & Safi, F. (2013). Survey of energy efficient data centers in cloud computing. In *Proceedings of the 2013 IEEE/ACM 6th International Conference on Utility and Cloud Computing*, (pp. 369–374)., Washington, DC, USA. IEEE Computer Society.

Sammut, C. & Webb, G. I. (2011). *Encyclopedia of Machine Learning* (1st ed.). Springer Publishing Company, Incorporated.

Shamir, A. (1979). How to share a secret. *Commun. ACM*, *22*, 612–613.

Silver, D., Huang, A., Maddison, C. J., Guez, A., Sifre, L., Van Den Driessche, G., Schrittwieser, J., Antonoglou, I., Panneershelvam, V., Lanctot, M., et al. (2016). Mastering the game of go with deep neural networks and tree search. *Nature*, *529*(7587), 484–489.

Smart, W. D. & Kaelbling, L. P. (2000). Practical reinforcement learning in continuous spaces. In *ICML*, (pp. 903–910).

Smith, C. (2012). Amazon ec2 outage takes down netflix, instagram and pinterest. Online; accessed 03-12-2016.

Solomon, M. G., Kim, D., & Carrell, J. L. (2014). *Fundamentals Of Communications And Networking* (2nd ed.). USA: Jones and Bartlett Publishers, Inc.

Staten, J. (2009). *"Cloud Is Defined, Now Stop the Cloudwashing"*. Forrester, Inc. Online ; accessed 11-06-2016.

Staten, J., Yates, S., Gillett, F. E., & Saleh, W. (2008). *"Is Cloud Computing Ready For The Enterprise?"*. Forrester, Inc. Online ; accessed 11-07-2016).

Stone, M. (1993). Hefty storage in a budeget. *PC Mag*, *12*(17), 247–250.

Sugiyama, M. (2016). *Introduction to Statistical Machine Learning*. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc.

Sutton, R. (2004). Reinforcement learning faq: Frequently asked questions about reinforcement learning. Online ; accessed 30-10-2016.

Sutton, R. S. & Barto, A. G. (1998). *Reinforcement Learning: An Introduction*. MIT Press.

Sutton, R. S. & Barto, A. G. (2012). Reinforcement learning: An introduction. unpublished book.

Szita, I. (2012). *Reinforcement Learning in Games*, chapter 7, (pp. 539–577). Berlin, Heidelberg: Springer Berlin Heidelberg.

Tanimoto, S., Murai, C., Seki, Y., Iwashita, M., Matsui, S., Sato, H., & Kanai, A. (2013). *A Study of Risk Management in Hybrid Cloud Configuration*, (pp. 247–257). Heidelberg: Springer International Publishing.

Tanimoto, S., Sakurada, Y., Seki, Y., Iwashita, M., Matsui, S., Sato, H., & Kanai, A. (2013). A study of data management in hybrid cloud configuration. In *2013 14th ACIS International Conference on Software Engineering, Artificial Intelligence, Networking and Parallel/Distributed Computing*, (pp. 381–386).

Tesauro, G. (1992). Practical issues in temporal difference learning. *Machine learning*, *8*(3-4), 257–277.

Tesauro, G. (2002). Programming backgammon using self-teaching neural nets. *Artif. Intell.*, *134*(1-2), 181–199.

Thakur, N. & Lead, Q. (2010). Performance testing in cloud: A pragmatic approach.

Technical report, diaspark.

Tsidulko, J. (2015). Overnight aws outage reminds world how important aws stability really is. CRN. Online; accessed 01-06-2016.

van Haaelt, H. (2012). *"Reinforcement Learning in Continuous State and Action space"*, chapter 7, (pp. 207–251). Berlin, Heidelberg: Springer Berlin Heidelberg.

Van Hasselt, H., Guez, A., & Silver, D. (2016). Deep reinforcement learning with double Q-Learning. In *AAAI*, (pp. 2094–2100).

van Hasselt, H. & Wiering, M. A. (2007). Reinforcement learning in continuous action spaces. In *2007 IEEE International Symposium on Approximate Dynamic Programming and Reinforcement Learning*, (pp. 272–279).

Van Moffaert, K. & Nowé, A. (2014). Multi-objective reinforcement learning using sets of pareto dominating policies. *Journal of Machine Learning Research*, *15*(1), 3483–3512.

van Otterlo, M. & Wiering, M. (2012). *"Reinforcement Learning and Markvo Decision Processes"*, chapter 1, (pp. 207–251). Berlin, Heidelberg: Springer Berlin Heidelberg.

Vasić, N., Novaković, D., Miučin, S., Kostić, D., & Bianchini, R. (2012). DejaVu: Accelerating resource allocation in virtualized environments. In *Proceedings of the Seventeenth International Conference on Architectural Support for Programming Languages and Operating Systems*, (pp. 423–436)., New York, NY, USA. ACM.

Vengerov, D. (2008). A reinforcement learning framework for online data migration in hierarchical storage systems. *The Journal of Supercomputing*, *43*(1), 1–19.

Voas, J. & Zhang, J. (2009). Cloud computing: New wine or just a new bottle? *IT Professional*, *11*, 15–17.

Whiteson, S. (2012). *"Evolutionary Computation for Reinforcement Learning"*, chapter 10, (pp. 325–355). Berlin, Heidelberg: Springer Berlin Heidelberg.

Wiering, M. A. & van Hasselt, H. (2007). Two novel on-policy reinforcement learning algorithms based on td(λ)-methods. In *2007 IEEE International Symposium on Approximate Dynamic Programming and Reinforcement Learning*, (pp. 280–287).

Witten, I. H. & Frank, E. (2011). *Data Mining: Practical Machine Learning Tools and*

Techniques (3rd ed.). Burlington, USA: Morgan Kaufmann Publishers Inc.

Wooldridge, M. & Jennings, N. R. (1995). Intelligent agents: Theory and practice. *The knowledge engineering review*, *10*(02), 115–152.

Xu, C.-Z., Rao, J., & Bu, X. (2012). Url: A unified reinforcement learning approach for autonomic cloud management. *Journal of Parallel and Distributed Computing*, *72*(2), 95–105.

Yang, K. & Jia, X. (2014). *Security for cloud storage systems*. Germany, Springer-Verlag Berlin Heidelberg: Springer.

Zhou, A. C., He, B., Cheng, X., & Lau, C. T. (2015). A declarative optimization engine for resource provisioning of scientific workflows in iaas clouds. In *Proceedings of the 24th International Symposium on High-Performance Parallel and Distributed Computing*, HPDC '15, (pp. 223–234)., New York, NY, USA. ACM.

Zuluaga, M., Krause, A., & Püschel, M. (2016). ϵ-pal: An active learning approach to the multi-objective optimization problem. *Journal of Machine Learning Research*, *17*, 1–32.

List of Symbols

α	(**Alpha**) Learning rate used to control the degree to which representation parameters are changed at each step.
β	(**Beta**) Another Learning rate, usually used with inner layer connection weights of an artificial neural network.
λ	(**Lambda**) Temporal discount factor; it scales the influences of the previous state value on the current state during learning time.
γ	(**Gamma**) Discount factor; it determines the importance of future value states.
η	(**Momentum**) Used to prevent artificial neural networks from converging to a local minimum during the learning process.

ϵ	(**epsilon** or ϵ-greedy) A method by which the learner-machine selects random actions with uniform distribution from a set of the available actions.
π	(**pi**) Action selection policy.
r	Correlation coefficient.
MAE	Mean absolute error.
$RMSE$	Root mean squared error.

Acronyms

LOMCSS	Learning to Optimise Multiple Cloud Storage Services
VMs	Virtual machines
IaaS	Infrastructure as a service
PaaS	Platform as a service
SaaS	Software as a service
NIST	National Institute of Standards and Technology
API	Application programming interface
RAID	Redundant array of inexpensive disks, or redundant array of independent disks
MDP	Markov decision processes

APPM	Access Pattern Prediction Model
BP	Back-propagation function
SRD	Standard RAID distribution policy
TD	Temporal difference (algorithm)
HR	Human resources

Glossary

Cloud computing	Computing resources delivered as services to cloud consumers through the Internet, in a pay-per-use model
Cloud consumers	Organisations or humans who use cloud services provided by a cloud provider.
Cloud provider	A company that provides one or more of the cloud services to the cloud consumers, either directly or through a broker.
Cloud performance	refers to the time that cloud servers require to process a task. In this thesis, the term refers to network latency, which is the time required to completely transfer a file from the client side to a cloud storage service.
Cloud carrier	A network which a cloud consumer uses to access cloud services; in general it refers to the Internet.

Virtual machines	A simulation of computer hardware to provide the functionality of a physical computer. The VM runs by means of an actual computer.
Vendor lock-in	A situation in which a customer depends on a vendor for products and services and cannot switch to another vendor without incurring substantial costs.
Computing resources	Any physical or virtual components of available computer hardware or software.
Data centre	A centralised repository containing a large group of networked computer servers and associated components. The centre is used for storing and processing large amounts of data.
Machine learning	A large set of algorithms that are used by computer software to learn from a specific dataset how to perform specific tasks.
Machine learning system	Computer software that is capable of learning from data or interacting with a specific environment, to perform future predictions or actions.
Hypothesis	In this work, a set of coefficient parameters that are defined by some underlying representation functions (e.g., linear functions or artificial neural networks).
Representation	A parametrised function that can learn a pattern of specific data.

Environment	the system in which the machine learning system interacts.
Supervised learning	A type of machine learning by which the machine learning system learns from labelled data.
Reinforcement learning	A type of machine learning in which the machine learning system interacts with its environment in a sequential decision-making scenario.
Artificial neural network	A mathematical model of a fully interconnected group of nodes, which is able to represent complex linear and non-linear functions.
Linear function	A mathematical model that represents a linear relationship between a dependent variable and one or more independent variables.
Episode	set of learning steps or states that a reinforcement learning system visits throughout its interaction with an environment. Each episode may terminate when the machine learning system reaches its goal, completes a terminal state or runs the maximum number of time steps.
Critic	In reinforcement learning, the critic is a synonym for the state value function.
State value	The value of a state is an estimation of the value of being in a given state. Basically, this value is the total reward value that can be accumulated in the future, starting from that state.

Actor	In reinforcement learning an actor is a synonym for action selection policy.
Action selection policy	The methodology that a reinforcement learning system uses to select its action. Usually action selection policy is based on the value of the state.
Actor-critcs	A method that separates the action selection policy from the state value function. That is, the action selection policy is independent of the state value function).
Reward	Feedback that can be received from the environment after visiting a certain state or performing an action in a certain state.
State	A set of different situations within an environment.
Synthetic data	Artificial data that are produced to resemble real data.

Index

cloud storage services, 34

action, 65
action-selection policy, 65
actor-critic, 73
actor-only, 72
Amazon S3, 34
API, 18, 27, 33, 34
applications, 17
APPM, 120
architecture, 91
artificial neural network, 51, 53–55, 57, 59, 60, 70–72, 109, 111
availability, 1

back propagation, 57, 59
backward pass, 51, 60
backwards pass, 55, 56
bandwidth, 34
bias, 52
black-box, 45, 46, 52, 53
Broad network access , 28
business drivers, 31

characteristics, 32, 48
classification, 62
cloud auditor, 30
cloud broker, 30
cloud carrier, 30
cloud computing, 17, 25–27, 30, 32, 40, 79, 87
cloud consumer, 29, 30, 34
Cloud consumers, 34
cloud performance, 18
cloud provider, 18, 30, 33, 34
cloud providers, 17, 80
cloud services, 18, 25
Cloud storage, 18
cloud storage, 19

cloud storage emulator, 87, 119
cloud storage service, 1, 17, 34
cloud storage services, 1, 25
CloundHarmony.com, 154
clustering, 61
Community Cloud, 29
computational resources, 17, 31
computer networks, 17
continuity, 1, 33
continuous, 66
cost, 19
cost-effective, 17
CPU, 80
critic, 72
critic-only, 72

data mirroring, 36
Data storage, 35
data storage, 32
data stripping, 36
decision tree, 46
Deep reinforcement learning, 75
deterministic, 67
dimension reduction, 62
dimensionality reduction, 62
discrete, 66

Elastic Net Regression, 57
elasticity, 17
environment, 30, 40, 48, 65, 79, 82

feature, 51, 53, 66
Feature continuity, 48
Feature space dimensionality, 48
Feature type, 48
features, 49
file access pattern, 44, 103
file access patterns, 87
flexibility, 34
Forrester Inc., 27
forward pass, 51, 56
framework, 1, 41, 65, 87, 91, 119
Fuzzy control, 83

gaussian function, 54
Google Cloud Storage, 34

hard disk, 35
hardware, 80
heuristic, 44
hidden layer, 53
Hybrid Cloud, 29
hyperbolic tangent function, 54
hyperbolic tangent function, 110
hypotheses, 46, 57
hypothesis, 45, 56
Hypothesis representation, 46
hypothesis representation, 50

IaaS, 17, 29
infrastructure, 28, 31, 34
input layer, 53

intercept, 52
Internet, 17, 27, 29
Internet browser, 34
IT, 31

Jcloud, 18

Lasso Regression, 57
latency, 18
latency time, 39, 121
learning algorithms, 79
Least Mean Squares, 58
Libcloud, 18
Linear Regression, 57
linear function, 54
linear model, 51, 52, 57
linear regression, 58
LMS, 58
Logistic Regression, 57
look-up table, 51, 70

machine learnin, 48, 62
Machine Learning, 45
Machine learning, 44
machine learning, 41, 44, 46, 51, 52, 57, 76, 79, 82, 84, 87
machine learning system, 47, 70
Markov decision process, 65
MDP, 65
MDPs, 159
Measured service, 28
Microsoft Azure Storage, 34
multi-agent systems, 158
multi-tenant model, 28
multiple cloud storage, 155
Multiple machine learning systems, 158
multiple objectives, 158

nearest neighbour, 46
network cost, 40
network latency time, 18
network throughput, 18
network usage, 19
NFQ, 75
NIST, 27, 30, 34
node, 53
nonparametric, 46
nsupervised learning, 49

OFDAMCSS, 87, 117, 119, 121, 129
On-demand self-service, 28
on-line storage, 34
operational cost, 40
operational costs, 17
optimisation algorithm, 52, 56, 57
Optimisation algorithms, 46
optimisation algorithms, 56
optimisation method, 51
output layer, 53
over-provisioning, 31

PaaS, 17, 30

packet, 18
Pareto optimality, 158
parity, 36
pay-per-use mechanism, 34
performance, 1, 40, 65
policy-search, 72
Polynomial Regression, 57
pool of resources, 27
prediction, 50
Private Cloud, 28
Public Cloud, 29

Q-Learning, 83
QoS, 82

RackSpace, 34
RAID, 35, 37, 157
ramp function, 54
Rapid elasticity, 28
regression, 59, 62, 63
regression analysis, 52, 57
reinforcement learning, 44, 46, 51, 57, 61, 63, 65, 80, 88, 109, 111, 155, 157
reliability, 35
remote storage, 17
residual, 57–59
Resource pooling, 28
reward, 63, 64, 67
Reward time delay, 157
Ridge Regression, 57
rule learning, 46

SaaS, 17, 30
scalability, 17, 34
scale down, 27
scale up, 27
security, 1
servers, 17
Service Availability, 19
Service continuity, 19
service models, 25
sigmoid function, 54, 110
simultaneously, 156
SLA, 81–84
SLED, 35
state, 63–65
    state features, 63
stationary, 68
step function, 54
Stepwise Regression, 57
stochastic, 67
storage, 17
storage cost, 40
supervised learning, 44, 46, 49, 61–63, 88
synthetic dataset, 156

tabular representation, 51
TD-Gammon, 74
transfer function, 54, 60

under-provisioning, 31

unsupervised learning, 44, 61
usage metrics, 27, 28

value approximation, 72
vendor lock-in, 1, 18
virtual, 33
virtual machine, 29
virtual machines, 80
virtualisation, 80
VM, 29, 81–84
VMs, 80, 82, 83

white-box, 45

XOR, 36