<table>
<thead>
<tr>
<th>Title</th>
<th>The application of machine learning to optimise live migration in cloud data centres</th>
</tr>
</thead>
<tbody>
<tr>
<td>Author(s)</td>
<td>Duggan, Martin</td>
</tr>
<tr>
<td>Publication Date</td>
<td>2019-08-06</td>
</tr>
<tr>
<td>Publisher</td>
<td>NUI Galway</td>
</tr>
<tr>
<td>Item record</td>
<td><a href="http://hdl.handle.net/10379/15310">http://hdl.handle.net/10379/15310</a></td>
</tr>
</tbody>
</table>
The Application of Machine Learning to Optimise Live Migration in Cloud Data Centres

Martin Duggan

Discipline of Information Technology
College of Engineering & Informatics
National University Of Ireland, Galway.

Submitted for the Degree of Doctor of Philosophy

July 2019
Abstract

Cloud computing providers utilise large-scale data centres to provide computing resource to users’ worldwide. However, an ongoing challenge facing cloud providers is to ensure that the required resource is available to users at all times. This problem is compounded further by the fact that resource consumption is in a constant state of flux. One approach leveraged to improve resource availability for users is ‘Live Migration’. This thesis presents a number of novel intelligent live migration solutions, by applying machine learning to improve the performance of a data centre.

The first contribution utilises a control algorithm known as ‘Reinforcement Learning’ to decide which virtual machines to migrate, based on RAM size when peak traffic condition is occurring and bandwidth availability is fluctuating. A Reinforcement Learning agent is implemented to decide which virtual machines to migrate from over-utilised hosts depending on currently available bandwidth and the host machine CPU utilisation.

The second contribution utilises the power of Neural Networks to predict when a host will become over-utilised. A ‘Recurrent Neural Network’ is implemented, trained with both traditional training algorithms (Back-Propagation and Back-Propagation-Through-Time) and evolutionary algorithms (Particle Swarm Optimisation, Covariance Matrix Adaptation and Differential Evolution) to best predict CPU utilisation of a host for a single and multiple time steps into the future.

The final contribution implements a ‘Recurrent Neural Network’ to predict both CPU of a host and bandwidth availability between host to decide optimal times to schedule live migration within a data centre. The Recurrent Neural Network predicts first if a host will become over-utilised and secondly predicts the available bandwidth before that host becomes over-utilised to best decided what times to migrate virtual machines.
The work presented in this Thesis demonstrates how Artificial Intelligence and Machine Learning algorithms can have a positive impact on live migration in cloud data centres.
Acknowledgements

I want to thank the following people for their support and encouragement throughout my PhD:

- Sincere thanks go to Dr Enda Barrett, Dr Enda Howley, and Dr Jim Duggan, who provided me with the opportunity to carry out the PhD. Without their invaluable support, it would not have been possible to conduct this research.

- Next, I would like to thank Jemma for all your love and support you have given me throughout my PhD experience.

- I would like to thank all of the researchers whom I have spent time with in Rooms 306 & 307. I thoroughly enjoyed all of our stimulating discussions we had in the last four years.

- Finally, I would like to thank my parents, Tom and Barbara Duggan for all their love and unwavering support.
Publications

Journals


Conferences


**Workshop**

# Contents

## 1 Introduction
- 1.1 Live Migrations & Unpredictability ........................................ 3
- 1.2 Research Opportunities ....................................................... 4
- 1.3 Goals, Objectives & Hypotheses ............................................. 6
- 1.4 Research Contribution .......................................................... 7
- 1.5 Thesis Structure ................................................................. 8

## 2 Cloud Computing & Live Migration
- 2.1 Introduction ........................................................................... 10
- 2.2 Cloud Computing ................................................................. 11
  - 2.2.1 Origins of Cloud Computing .............................................. 13
  - 2.2.2 Grid Computing .............................................................. 14
  - 2.2.3 Computational Clusters ................................................... 15
  - 2.2.4 Computing as a Service ................................................... 15
    - 2.2.4.1 Infrastructure as a Service .......................................... 16
  - 2.2.5 Virtualisation Technology ............................................... 16
  - 2.2.6 Virtual Machine Monitor - Hypervisor ............................... 17
  - 2.2.7 Live Virtual Machine Migration ........................................ 19
  - 2.2.8 Live Migration & Unpredictability .................................... 24
  - 2.2.9 Predicting VM Provisioning, Placement & Consolidation ...... 25
- 2.3 Machine Learning & Cloud Computing .................................... 29
  - 2.3.1 Reinforcement Learning & Cloud Computing ..................... 30
  - 2.3.2 Neural Network & Cloud Computing .................................. 32
- 2.4 Summary .............................................................................. 34

## 3 Reinforcement Learning & Neural Networks
- 3.1 Introduction ........................................................................... 36
- 3.2 Reinforcement Learning ........................................................ 36
3.2.1 Markov Decision Processes ........................................... 36
3.2.2 Dynamic Programming ................................................. 38
  3.2.2.1 Policy Iteration ................................................. 39
3.2.3 Value Iteration ......................................................... 40
3.2.4 Reinforcement Learning Approaches ................................. 40
  3.2.4.1 Monte Carlo Methods ............................................ 41
  3.2.4.2 Temporal Difference Learning ................................. 42
  3.2.4.3 SARSA Learning ................................................. 42
  3.2.4.4 Q learning ....................................................... 43
  3.2.4.5 Agents ............................................................ 44
3.3 Artificial Neural Networks ............................................... 45
  3.3.1 Forward Pass ....................................................... 46
  3.3.2 Output Layers ...................................................... 47
  3.3.3 Back-propagation .................................................. 47
3.4 Recurrent Neural Networks ............................................ 49
  3.4.1 Forward Pass ....................................................... 50
  3.4.2 Back-propagation Through Time .................................. 51
3.5 Neural Network Optimisation Algorithms .............................. 51
  3.5.1 Swarm and Evolutionary Algorithms ............................. 52
    3.5.1.1 Particle Swarm Optimisation ................................. 52
    3.5.1.2 Covariance Matrix Adaptation Evolutionary Strategy 54
    3.5.1.3 Differential Evolution ....................................... 55
3.6 Comparative Forecasting Methods .................................... 56
  3.6.1 Previous Walk .................................................... 56
  3.6.2 Moving Average .................................................. 57
  3.6.3 Linear regression ................................................ 57
  3.6.4 ARIMA Modelling ............................................... 57
3.7 Summary ................................................................. 59

4 An Autonomous Approach for the selection of Virtual Machines for Live Migration ............................. 61
  4.1 Introduction .......................................................... 61
  4.2 Motivations and Aims ............................................... 62
  4.3 Learning Methods .................................................... 62
    4.3.1 Q-Learning ....................................................... 62
    4.3.2 SARSA ............................................................. 63
6 A Multi-Time Steps Ahead Prediction Approach for Scheduling Live Migration in Cloud Data Centres

6.1 Introduction ................................................. 101
6.2 Business Understanding ................................. 101
6.3 Data Understanding and Data Preparation .............. 103
6.4 Modelling .................................................. 106
6.5 Experiment Details ......................................... 107
6.5.1 Metrics ................................................ 108
6.5.2 Simulator Model ........................................ 109
6.5.3 Experiments ............................................ 111
6.6 Results ..................................................... 112
6.6.1 Bandwidth Prediction Results ......................... 112
   6.6.1.1 Single Time Step Ahead ......................... 112
   6.6.1.2 Multi-Time Steps Ahead ......................... 115
6.6.2 CPU Prediction Results ............................... 116
   6.6.2.1 Single Time Step ahead ....................... 116
   6.6.2.2 Multi Time Step ahead ......................... 117
   6.6.2.3 Unseen Host Data-Set ......................... 119
6.6.3 Simulation Results .................................... 120
<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>6.6.4 Discussions</td>
<td>122</td>
</tr>
<tr>
<td>6.7 Limitations</td>
<td>124</td>
</tr>
<tr>
<td>6.8 Comparison with the State of the Art</td>
<td>124</td>
</tr>
<tr>
<td>6.9 Summary</td>
<td>125</td>
</tr>
<tr>
<td><strong>7 Conclusion</strong></td>
<td>126</td>
</tr>
<tr>
<td>7.1 Introduction</td>
<td>126</td>
</tr>
<tr>
<td>7.1.1 An Autonomous Approach for the Selection of Virtual Machines for Live Migration</td>
<td>126</td>
</tr>
<tr>
<td>7.1.2 Predicting Host CPU Utilisation Cloud Computing Using Recurrent Neural Networks</td>
<td>127</td>
</tr>
<tr>
<td>7.1.3 Multi-Time Step Ahead Prediction Approach for Scheduling Live Migration in Cloud Data Centres</td>
<td>128</td>
</tr>
<tr>
<td>7.1.4 Limitations</td>
<td>129</td>
</tr>
<tr>
<td>7.1.4.1 State-Action-Space of Reinforcement Learning</td>
<td>129</td>
</tr>
<tr>
<td>7.1.4.2 Uni-variate Forecasting of Cloud resources</td>
<td>130</td>
</tr>
<tr>
<td>7.1.4.3 Implementing Resource Predictions in Data Centre Management</td>
<td>130</td>
</tr>
<tr>
<td>7.1.4.4 The Adaption of AI in Data Centres</td>
<td>130</td>
</tr>
<tr>
<td>7.1.5 Future Work</td>
<td>130</td>
</tr>
<tr>
<td>7.1.6 Final Conclusion</td>
<td>131</td>
</tr>
<tr>
<td><strong>Bibliography</strong></td>
<td>132</td>
</tr>
</tbody>
</table>
List of Figures

1.1 Google Cluster Trace Workload .................................................. 2
1.2 Static vs Dynamic Infrastructure Resources Provisioning [6]. ........... 4

2.1 Overview of Cloud Computing .................................................... 13
2.2 Virtual Machine Abstraction [207] .............................................. 18
2.3 Virtualised Physical Machines with Live Migration Capability [25],
   (PM represent a Host/Server) ..................................................... 20

3.1 The Agent’s Environment Interaction [176] .................................... 41
3.2 Neural Network Architecture ...................................................... 45
3.3 Recurrent Neural Network Architecture ........................................ 50

4.1 Illustration of How an Agent Perceives it’s State Action space .......... 65
4.2 The RL Agent’s Interaction With The Cloud Environment ............... 67
4.3 Gaussian Bandwidth Model Implemented in Cloudsim ....................... 72
4.4 Training Convergence of the RL Agent ....................................... 73
4.5 Simulation Energy Consumption ............................................... 76
4.6 Simulation Service Level Agreement Violations ............................. 77
4.7 Simulation ESV Performance ...................................................... 78

5.1 CRISP-DM [197] ........................................................................... 83
5.2 Histogram of the Planet Lab CPU Training Data-set ........................ 84
5.3 Histogram of the Planet Lab CPU Testing Data-set .......................... 85
5.4 Fully Connected Recurrent Neural Network [129] ............................ 87
5.5 Convergence Rate of Neural Networks ......................................... 90
5.6 Host CPU Training Prediction (Time Steps 1 - 500) ........................ 91
5.7 Host CPU Training Prediction (Time Steps 500 - 1000) .................... 92
5.8 Host CPU Training Prediction (Time Steps 1000 - 1500) ................. 93
5.9 Forecasting Accuracy Spread ...................................................... 94
5.10 Host Utilisation Predictions for Test Data ..................................... 95
5.11 Prediction Residuals on Test Data................................. 96

6.1 Simulator Bandwidth Model......................................... 104
6.2 CPU Training Data Set................................................. 105
6.3 Histogram of the Bandwidth Data-set.............................. 105
6.4 Histogram of the CPU dataset...................................... 106
6.5 BPTT and BP( Sliding Window 20 )................................. 114
6.6 BP( Sliding Window 10 ) and ARIMA).............................. 114
6.7 Multi-time Steps Ahead Bandwidth Predictions.................. 116
6.8 Actual, BPTT and LR CPU Utilisation Predictions.............. 117
6.9 BPTT MAE for 1 and 6 Time steps................................ 118
6.10 Energy Consumption of all Algorithms........................... 122
6.11 SLA Performance of all Algorithms............................... 123
# List of Tables


2.1 Research Papers Which Are Of Interest To This Thesis ........... 35

4.1 Different increments of the State Space .............................. 65
4.2 Average Migration Times for Each of the RL Policies .............. 74
4.3 Energy Consumption for Each of the RL Policies .................. 75
4.4 SLAV for Each of the RL Policies .................................. 75
4.5 Average Migration Times ........................................... 78
4.6 Summary of all Performance Metrics for Each Algorithm ........... 79

5.1 Summary Metrics of the Training and Testing CPU Data-sets .......... 86
5.2 Training Data Accuracy ........................................... 91
5.3 Test Data Accuracy .............................................. 94
5.4 Multi-Step Prediction Accuracy .................................... 95

6.1 Summary Metrics of the Bandwidth and CPU Data-sets .............. 106
6.2 Host power consumption (KWh) .................................... 110
6.3 Bandwidth Training Data Prediction Accuracy ...................... 113
6.4 Bandwidth Test Data Prediction Accuracy .......................... 113
6.5 BPTT Multi-step Ahead Training Data Prediction Accuracy ........ 115
6.6 BPTT Multi-step Ahead Test Data Prediction Accuracy ............ 115
6.7 Train CPU Data Prediction Accuracy ............................... 116
6.8 Test CPU Data Prediction Accuracy ................................ 117
6.9 Multi-step Ahead Trained Data Prediction Accuracy ............... 118
6.10 Multi-step Ahead Test Data Prediction Accuracy ................... 119
6.11 Test Prediction of New Host CPU Data ............................. 119
6.12 Multi-step Ahead Test Data Prediction Accuracy (Different host) .... 120
6.13 SLAV and ESV Results .......................................... 121
Abbreviation

AI = Artificial Intelligence
AMT = Average Migration Time
ANN = Artificial Neural Network
ARIMA = AutoRegressive Integrated Moving Average
BP = Back-Propagation
BPTT = Back-Propagation-Through-Time
BW = Bandwidth
CMA-ES = Co-variance Matrix Adaptation - Evolutionary Strategy
CRISP-DM = Cross-Industry Process for Data Mining Model
CRM = Customer Relationship Management
DDoS = Distributed Denial-of-Service
DE = Differential Evolution
DP = Dynamic Programming
DVMC = Dynamic Virtual Machine Consolidation
E-greedy = Epsilon-greedy
EMA = Exponential Moving Average
ESV = Energy * SLAV
FFNN = Feed Forward Neural Network
HPC = Highest Potential Growth
IaaS = Infrastructure as a Service
ICT = Information Communication Technology
KVM = Kernel Virtual Machine
LAN = Local Area Network
LR = Linear Regression
LSTM = Long Short Term Memory
MA = Moving Average
MAE = Mean Absolute Error
MAPE = Mean Absolute Percentage Error
MC = Maximum Correlation
MDP = Markov Decision Process
ML = Machine Learning
MLP = Multi-Layer Perceptron
MM = Minimum Migration
MMT = Minimum Migration Time
MSE = Mean Squared Error
NIST = National Institute of Standard and Technology
PaaS = Platform as a Service
PAC = Pattern Drive Application
PDM = Performance Degradation due to Migration
PM = Physical Machine
PSO = Particle Swarm Optimisation
PUE = Pattern Usage Effectiveness
PW = Previous Walk
QoS = Quality of Service
RL = Reinforcement Learning
RMSE = Root Mean Squared Error
RNN = Recurrent Neural Network
RS = Random Selection
RTRL = Real-Time Recurrent Learning
SaaS = Software as a Service
SARSA = State-Action-Reward-State-Action
SDN = Software Defined Network
SLA = Service Level Agreement
SLATAH = Service Level Agreement Time Per Active Host
SLAV = Service Level Agreement Violation
TCP = Transmission Control Protocol
TD = Temporal Difference
UDP = User Datagram Protocol
UMC = Utilisation and Minimum Correlation
VM = Virtual Machine
VMM = Virtual Machine Monitor
WAN = Wide Area Network
Chapter 1

Introduction

Cloud computing has revolutionised the Information and Communication Technology (ICT) industry by enabling on-demand provisioning of elastic computing resources on a pay-as-you-go basis. The Cloud Computing paradigm has emerged as a solution for hosting large-scale online applications (e.g., social networking, web search and video gaming), and has transformed the ICT sector by delivering, managing and consuming IT capabilities. Cloud computing has enabled organisations to outsource their computational needs to a range of IT companies including Amazon, Cisco, Yahoo, Salesforce, Facebook, Microsoft and Google who have their own data centres and provide ‘pay-as-you-go’ cloud services. This outsourcing of technologies allows organisations to avoid the high investment costs of building their private computing infrastructure and associated costs of maintenance, upgrades and enabling the organisations to focus on their primary business operations [116].

Cloud Computing is categorised into three services platforms: 1) Infrastructure as a Service (IaaS), which rents computing resources, such as processing, storage, and network, 2) Platform as a Service (PaaS), which offers services that host and manage applications, and 3) Software as a Service (SaaS) that directly provides applications to the end users. The IaaS platform is the main focus of this thesis and will be discussed in greater detail in chapter 2. Typically, the IaaS servers or host machines resources are delivered to users in the form of Virtual Machines (VMs) instances. The configuration of each of these VMs can be heterogeneous or homogeneous as other VMs. A recent forecast [70] identifies the IaaS model as the fastest expanding Cloud service, with a revenue of $24.6 billion in 2018 and a compound annual growth rate of 31% from 2014 to 2018. One of the main reasons for the high investment in the IaaS platform is that by 2020, it is estimated there will be 51,974 GB of internet traffic generated per second [97]. With large amounts of data being processed each second,
Cisco has predicted that more than 86% of workloads will be processed in Cloud data centres by 2019 [95].

The cloud workloads can cause the IaaS physical resource’s usage to fluctuate. Depending on the times of the day, workload demands can have a detrimental effect on the user’s VM performance. Not knowing how a VM will perform in the future, introduces the problem of uncertainty and unpredictability, which causes significant problems for the scheduling of tasks and allocation of resources in the cloud.

High resource consumption is not only difficult to manage but also tricky to predict. Therefore, the strategy a data centre utilises to mitigate uncertainty is a growing concern in the cloud community. For instance, when intensive workload demands occur and a host is not capable of providing sufficient resources to users, this will lead to over-utilisation within that host. Once a host’s resources are depleted and become incapable of providing the resources required to a VM or groups of VMs, a process known as live migration is initiated, (i.e. the process of moving VMs from one host to another). The reason for using 'live migration' is to ensure the desired level of Quality of Service (QoS) is maintained and satisfies end users performance. The QoS is characterised by the Service Level Agreements (SLAs), which defines the
required performance levels, such as minimal response time and minimal throughput. Therefore, the main challenges facing live migration are: 1) To determine when a host’s resources will become depleted, and 2) Providing a fast live migration process between host machines.

1.1 Live Migrations & Unpredictability

Live migration is the standard approach to ensuring that QoS is being delivered to users’ VMs [42]. Live migration leverages the virtualisation technology that shares a host among multiple VMs, where each VM runs one or more applications. Live migration is advantageous when a host is over-utilised (i.e., move VMs to a different host to provide relief to another host’s resources) or when a host is under-utilised (i.e., consolidate VMs on to fewer hosts to reduce energy consumption in a data centre). Even though live migration can help maintain SLA levels and reduce energy, it can also have a negative impact on the performance of an application running on a VM during migration. Therefore, a live migration approach must be designed to ensure the fast movement of a VM from a host machine.

The IaaS providers are faced with constant demands for resources. This in-turn generates fluctuating workloads which can cause host machines to become over-utilised or under-utilised. Fluctuating workloads makes it very difficult for the cloud management system to decide when to initiate live migration.

The analysis of the Google cluster workload traces illustrates how unpredictability can cause significant disruption to a host’s performance. Figure 1.1 shows that user demands experience seasonal fluctuations with sporadic bursts of up to 20 times the average load. These sporadic bursts make it difficult for the cloud management system to organise resources or events such as live migration at specific times to optimise QoS. Therefore good resource planning must be in place to improve a performance of data centres. Short-term and long-term resource planning is seen as a major problem area in cloud computing [164][73]. Research has shown that organising host resources for workloads on a short timescale such as 5 to 10 minute intervals is a more difficult problem to solve [19].

Typical types of long-term resource planning of physical machine resources are illustrated in Figure 1.2. Figure 1.2 (A) highlights how traditional static resource provisioning solutions can lead to poor performance. When cloud providers over-provision resources to meet potential future demands, significant costs and unused resources are often the result. Figure 1.2 (B) depicts host resource provisioned for
average workloads, which lead to degradation of the quality of service (QoS). To avoid such issues, IaaS providers must be able to dynamically adjust their hosting capacity in response to demand fluctuations as in Figure 1.2 (c). However, planning and the provisioning of resources for the short-term is just as vital as planning for long-term demands to ensure live migration can be scheduled at optimal times. In this thesis we define short term as 5 minutes to 1 hour and long term as 1 hour and greater.

As with all processes in the cloud environment, unpredictability has a significant effect on processes such as live migration. Being able to mitigate unpredictability and decide with certainty when best to schedule a VM for live migration will greatly improve the performance of a data centre. Unpredictability of resource usage underlines the need for a more advanced live migration mechanism to help providers to achieve better QoS and higher profits.

### 1.2 Research Opportunities

Many researchers have examined uncertainty or unpredictability in areas such as user performance [152] [187] and although there is initial research on the guarantees of live migration [191], the role of uncertainty in the scheduling of live migration has not yet been sufficiently discussed in the cloud literature.

Armbrust et al. detail a list of factors which continue to impact the growth of cloud computing [6] as shown in Table 1.1. They identify *Performance Unpredictability* as one of the ten most disruptive factors to the development of cloud computing. Performance unpredictability refers to the dynamic nature of the infrastructure of cloud computing where competition between virtual machines for the underlying shared resources causes contention issues. Much uncertainty exists about how live migration is scheduled. For instance, should scheduling algorithms only consider the CPU of
a host when deciding times for live migration or should the availability of network bandwidth also be considered.

Tchernykh et al. outlines the primary sources of unpredictability in cloud computing. However, they provide a more detailed discussion of the actual components causing unpredictability [179]. They list a variety of unpredictability types such as data, virtualisation, jobs arrival, migration, energy consumption and scalability. Tchernykh et al. also detail a number of parameters which lead to unpredictability in a host’s performance such as network capacity, resource capacity, data transfer time, available storage, number of processors, available memory, processing time, sufficient bandwidth and effective performance. In relation to this Thesis, they describe *migration* as one of the significant types of unpredictability. They also highlight *Network Capacity, Resource Capacity, Data Transfer Time and Bandwidth* as causes of unpredictability in a host, which are also parameters that must be considered for live migrations.

To reduce unpredictability Armbrust et al. propose a number of possible solutions [6]. These solutions range from the increase of RAM on servers to the development of higher bandwidth network switches to the application of machine learning. Recently machine learning approaches have been applied to cloud computing to improve scheduling [57], intrusion detection [189], Distributed Denial of Service (DDoS) attack defence [105] and load forecasting [149].

This thesis focuses on investigating the abilities of machine learning algorithms to mitigate performance unpredictability when live migration arises. We examine 3 major problems areas in migration:

1. Selecting which VM to be migrated from an over-utilised host.
2. How to accurately predict when over-utilisation is going to occur on a host.
3. Deciding optimal times to schedule live migration.

To accurately pre-empt live migration of VMs, an IaaS virtual machine management system must be able to predict a host machine’s expected resource consumption in the future. For this scenario, we investigated a machine learning algorithm known as the Recurrent Neural Network (RNN), which has the ability to learn from historical data and also to keep a memory of previous predictions.

We also implement a Reinforcement Learning algorithm to select VMs to be migrated based on network traffic load. The majority of research to date which implements Reinforcement learning in the IaaS domain focuses on powering up and
<table>
<thead>
<tr>
<th>Problem</th>
<th>Opportunity</th>
</tr>
</thead>
<tbody>
<tr>
<td>Availability of Service</td>
<td>Use multiple cloud providers</td>
</tr>
<tr>
<td>Data Lock-In</td>
<td>Standardise APIs; Compatible software for Surge or Hybrid Cloud Computing</td>
</tr>
<tr>
<td>Data Confidentiality and Auditability</td>
<td>Deploy encryption, VLANs, firewalls</td>
</tr>
<tr>
<td>Data Transfer Bottlenecks</td>
<td>FedExing Disks; Higher Bandwidth Switches</td>
</tr>
<tr>
<td>Performance Unpredictability</td>
<td>Improved VM support; Flash Memory; Gang Schedule VMs</td>
</tr>
<tr>
<td>Scalable Storage</td>
<td>Invent Scalable Store</td>
</tr>
<tr>
<td>Bugs in Large-Scale Distributed Systems</td>
<td>Invent Debugger that relies on Distributed VMs</td>
</tr>
<tr>
<td>Scaling Quickly</td>
<td>Invent Auto-scaler that relies on Machine Learning, Snapshots for Conservation</td>
</tr>
<tr>
<td>Reputation Fate Sharing</td>
<td>Offer reputation guarding services like those for email</td>
</tr>
<tr>
<td>Software Licensing</td>
<td>Pay-per-use licenses</td>
</tr>
</tbody>
</table>

Table 1.1: Top 10 Problems Preventing the Growth of Cloud Computing, Armbrust et al. [6]

powering off host machines [181]. However, we implemented Reinforcement Learning as a decision support system for the scheduling of different types of virtual machines. By applying several machine learning algorithms to different domains within the live migration problem, our goal is to advance upon existing work to deal with uncertainty in cloud environment.

1.3 Goals, Objectives & Hypotheses

Through the deployment of learning and predicting techniques, this Thesis proposes novel approaches which advance the current state of the art for live migration within cloud infrastructures.

More explicitly we examine the following research questions:

1. Will a learning agent utilising Q-learning improve data centre performance when deciding the types of virtual machines to be migrated under a variety of network traffic conditions such as high/low resource usage workload models?

2. Is a Recurrent Neural Network capable of out-performing traditional non-linear and linear forecasting methods to anticipate when a host will become over-utilised?

3. Would implementing a Recurrent Neural Network to schedule live migration improve upon traditional methods and reduce network resource usage at critical times in a cloud environment?

Given these research questions, a number of hypotheses can be proposed as follows:
• **H1**: Through the interaction with the environment, a Reinforcement Learning agent can learn to select optimal VMs to be migrated between hosts while utilising bandwidth availability.

• **H2**: Employing a predictive algorithm trained with traditional Back-propagation algorithms and evolutionary methods which are able learn from historical data to improve upon traditional non-linear and linear algorithm’s prediction accuracy of future host resource utilisation.

• **H3**: Implementing a recurrent predictive algorithm which stores previous predictions as memory, will aid in developing an efficient scheduling live migration to best utilise resources at critical times better when compared to heuristic approaches.

To evaluate these objectives and hypotheses, we present a series of simulations which empirically analyse our approaches using real-world data from hosts and virtual machines deployed in real-world data centres.

### 1.4 Research Contribution

To achieve the objectives listed above, this Thesis makes three main contributions. A brief overview of the contributions is presented in the following:

1. **An Autonomous Approach for the Selection of Virtual Machines for Live Migration**
   The first contribution of this Thesis is a novel ‘VM selection policy’. A Reinforcement Learning algorithm is used to select virtual machines for live migration. Reinforcement learning provides an opportunity for highly autonomous and adaptive management in dynamic environments. The balance of exploration and exploitation of the Reinforcement Learning algorithm aids in finding optimal policies (VMs to migrate) which standard heuristic do not have the ability to do. The Reinforcement Learning agent employed in this chapter learns an effective policy for selecting which virtual machines to be migrated (through trial and error) from an over-utilised host depending on network bandwidth availability.

2. **Predicting Host CPU Utilisation in Cloud Computing Using Recurrent Neural Networks**
The second contribution of this thesis is applying Recurrent Neural Network to improve the prediction of CPU in a host, thus predicting times when a host will become over-utilised. A number of traditional training algorithm such as Back-propagation and Back-propagation-Through-Time along with a number of state of the art swarm and evolutionary optimisation algorithms (Particle Swarm Optimisation (PSO), Differential Evolution (DE) and Co-variance Matrix Adaptation Evolutionary Strategy (CMA-ES)) are implemented to train the neural networks to predict host utilisation. Each Neural Network trained has the ability to learn and model non-linear and complex relationships, which is important in real-life as many of the relationships between inputs and outputs are non-linear as well as complex. All three evolutionary algorithms will be evaluated against each other, the traditional training algorithms and non-linear and linear approaches on a real world CPU data-set.

3. A Multi-Time Step Ahead Prediction Approach for Scheduling Live Migration in Cloud Data Centres

In the final contribution of this Thesis we present a ‘Multi-Step-Ahead’ prediction model that utilises a Recurrent Neural Network to determine optimal times to schedule the live migration process, to best utilise network resource available. This model performs three essential steps. In step 1 the Recurrent Neural Network will predict when a host will become over-utilised. The second step comprises of using the Recurrent Neural Network to predict bandwidth usage for the next 30 minutes, once a host is predicted to become over-utilised. One of the main advantage of recurrent networks is that a Recurrent Neural Network has the ability to store information and model sequence of data (i.e. time series), so that each sample can be assumed to be dependent on previous ones, where as traditional non-linear and linear prediction methods consider each data point is independent of each other. The third step decides the most optimal time to migrate a virtual machine depending on the predicted bandwidth for the next 30 minutes.

1.5 Thesis Structure

This section provides an overview of each chapter contained within this thesis. The initial chapters (1,2,3) describe the domains of cloud computing, live migration, Reinforcement Learning and Neural Networks, as well as describing the current and ongoing work relevant to our research. The ensuing results chapters (4,5,6) present our contributions and findings within these domains. Through empirical analysis,
we demonstrate our advancements over existing works. Finally, we summarise the contributions and discuss possible extensions to this work in the concluding chapter (7).

The rest of this thesis is organised as follows:

- Chapter 2: Provides an overview of cloud computing, live migration and forecasting cloud resources. Firstly, we analyse the history of the cloud, tracking it through a number of paradigm changes from cluster to grid to cloud computing. We discuss the ‘as a service’ movement describing ‘Infrastructure as a Service’ within computational clouds. Secondly, we discuss live migration, describing the related works in the field, the various techniques currently employed and possible directions going forward. Finally, we discuss the forecasting methods used to predict cloud resources and the advancement of machine learning in cloud computing.

- Chapter 3: Details an unsupervised learning technique known as Reinforcement Learning and a supervised learning algorithm known as Neural Networks. We describe the Markov Decision Process and discuss Neural Networks training algorithms. We also detail four traditional forecasting techniques, which will be used to benchmark the neural network’s performance.

- Chapter 4: Describes the first of our three main result chapters, detailing the Reinforcement Learning algorithm that acts as a decision support system for deciding the types of virtual machines that are to be migrated from over-utilised hosts depending on the network traffic loads.

- Chapter 5: Examines how well a Recurrent Neural Network is able to predict different hosts CPU utilisation compared to traditional non-linear and linear models. We demonstrate that by using Recurrent Neural Networks, CPU prediction can be improved for single step ahead and multiple steps into the future.

- Chapter 6: This chapter presents a predictive model which determines optimal times to migrate virtual machines, utilising recurrent neural networks. By simulating variable network loads and virtual machine performance we demonstrate the effectiveness of our proposed method for scheduling live migration.

- Chapter 7: Summarises the research presented in this Thesis, offering conclusions and possible future directions going forward within the cloud domain.
Chapter 2

Cloud Computing & Live Migration

2.1 Introduction

“As of now, computer networks are still in their infancy, but as they expand and become more sophisticated, we will probably see the spread of ‘computer utilities’, like electric and telephone utilities, will service individual homes and offices throughout the country.”

(Prof. Leonard Kleinrock, UCLA press release, July 3rd, 1969)

Computing as an essential utility for everyday life has been proposed since the 1960s by John McCarthy of Massachusetts Institute of Technology (MIT), who was one of the first people to publicly state that “computing may one day be organised as a public utility”. This view has since been iterated by researchers such as Buyya, who states that the operation of existing utilities such as electricity, telecommunications, water, gas and a fifth utility which is the computing resource, could one-day service our homes and businesses throughout the land [33]. Traditional utility providers operate a pay-per-use model offering a seemingly unlimited resource supply to the end consumer. If we take the electricity market as an example, the consumer only has to assume that there will always be a constant supply of electricity resource on demand whenever they may need it. Consumer’s usage of electricity is metered on a per unit basis and billed by their level of consumption. In the context of cloud computing, Amazon EC2, GoGrid and Rackspace manage the provision of large amounts of computational resources in an on-demand pay-per-use fashion. The responsibility for managing the cloud services, hosting application and providing security is delegated to the cloud provider. Cloud computing is similar to traditional utilities as the
consumer only pay for what they consume. Cloud computing also allows consumers to export all their computing needs to a third party and not having to design, build and maintain their own data centre saving money on expensive purchasing of equipment or the on-going staffing and maintenance costs. The rapid development of cloud computing technologies has enabled anyone with a credit card and connection to the internet, with the opportunity to acquire the equivalent of what once required a serviced office space, a skilled system admin staff, an abundance of hardware/software components and all from the confines of their office or home chair. However, the security element has always been a significant concern and a stumbling block for cloud users. Outsourcing ones entire computing requirements to an external provider is a daunting prospect for many, which raises concerns about data security or even data ownership and other factors such as performance unpredictability [6]. When migrating to the cloud, consumers need to ensure that their applications and services can still maintain the same level of quality, similar to their previous in-house hosted scenario and not be effect by factors such as unpredictability.

Our examination of the literature concerning cloud and infrastructure computing details; cloud, grid, cluster computing, virtualisation technology and the ‘as a service’ computing movement. We review the relevant research in these areas, discussing the merits of each and providing a detailed overview of the topics. We then follow up this examination with a discussion on live migration and factors which affect it at critical times. Using virtual machines from computational clouds as the resources to be distributed, we detail the relevant works within the said areas and provide a general overview of the topics.

2.2 Cloud Computing

Cloud Computing refers to the idea of delivering various computing resources “as a service” to consumers over the internet. It is difficult to provide one standard universal definition of cloud computing which is broad enough to encompass the entirety of the spectrum, with many efforts focusing on defining a particular aspect of the cloud [91]. A good concise definition which has been widely accepted by the cloud community that encapsulates cloud computing, is the definition provided by the National Institute of Standards and Technology (NIST) [135]:

Cloud computing is a model for enabling convenient, on-demand network access to a shared pool of configurable computing resources (e.g., networks, servers, stor-
age, applications, and services) that can be rapidly provisioned and released with minimal management effort or service provider interaction. This definition of cloud computing encompasses the majority of what is considered cloud computing by modern standards, covering infrastructure clouds, software as a service and platform as a service. Today, cloud computing underpins a significant portion of the world wide web with cloud data centre traffic predicted to represent 95% of total data centre traffic by 2021 [96], and is the de-facto means of deploying services and applications at scale.

Figure 2.1 provides a visual overview of cloud computing from the perspective of the current three principle layer model; the infrastructure layer, the platform layer and the software layer. The infrastructure of the cloud, also known as the Infrastructure as a service (IaaS) layer, consists of the virtual servers, the network infrastructure and the block storage mechanisms, in other words, the “nuts and bolts” layer of the cloud. This provides the platform for the consumer to access their data, applications and resources anywhere in the globe, across a range of devices (laptops, tablets and smartphones). However, the IaaS layer requires greater technical support and management from the end user to configure their servers from an operating system level installing required applications/business logic and managing their data. The consumer is effectively required to build their platform layer on top of the infrastructure in the cloud.

The platform layer allows users to deploy their applications on a pre-configured stack of virtual machines and removes the need to build their supporting platform. The platform layer is also known as the Platform as a service layer (PaaS) and along with the IaaS is managed by the service provider offering the user the freedom to focus on their core operations. Microsoft Windows Azure is a good example of this, offering an environment suitable for the deployment and scaling of applications designed specifically for running on Windows. The solution stack consists of components such as the .NET framework and a scalable MS SQL Server solution, which assists the speedy deployment of compatible applications to the cloud. The software layer also known as the Software as a Service (SaaS) layer in cloud computing refers to a subscription-based software model which is hosted on the cloud and accessed via the internet. Salesforce’s Customer Relationship Management (CRM) is a prime example of this, where you pay a monthly subscription to use the service which is hosted remotely by them. The service is usually accessible via a controlled web interface. Infrastructure as a Service is discussed in greater detail in Section 2.2.4.1. This thesis
refers primarily to IaaS environment as it is the IaaS platform which serves as the foundation of the management and scheduling of resources in cloud computing.

### 2.2.1 Origins of Cloud Computing

Many years before the development of virtualisation technologies, time-sharing operating systems were deployed on mainframe computers. This allowed CPU time to be shared across multiple users. The user’s access to these mainframe computers was facilitated through dumb terminals with each user able to submit jobs for processing simultaneously [2]. However, this approach had several issues, the main problem being that the misuse of the system by a single user could degrade performance for all other users [156].

To counteract this problem, IBM invented the ‘virtual machines’ (VMs), which enable the isolation of user’s instances from each other. With a virtual machine a single large mainframe computer could be split up via virtualisation into smaller components which were isolated from each other, resulting in greater system utilisation and performance.
Today’s cloud computing paradigm has been primarily driven by a requirement to handle user requests on a massive worldwide scale, which in-turn has forced cloud providers such as Amazon and Google to develop large-scale data centres to support these requests. Cloud computing today is effectively commercialisation of the knowledge obtained in building data centres over the years [101]. The current cloud infrastructures would not have been developed if it was not for the advancements made in the areas of Grid and Cluster computing in the years leading up to its development.

2.2.2 Grid Computing

Grid Computing emerged in the 1990s which enabled multiple institutions to connect lower specification machines into a grid structure to process large computational tasks. A number of grids have been developed such as the Science Grid [148], the Earth System Grid [20] and TeraGrid [36]. The term Grid was chosen, similar to the electric power grid, which provides access to electricity with seemingly unlimited supply [66]. It was envisioned that computing power could be accessed on-demand as a utility, similar to what was suggested by John McCarthy back in 1961, when he said, “Computation may someday be organised as a public utility”. In many ways, the vision behind the grid computing movement is similar to what modern day cloud computing offers concerning on-demand computing [67].

Grid computing is created when a group of interested parties who agree to share small amounts of their compute resources with the grid, to create an overall larger more powerful pool for resources and this is facilitated by the Virtual Organisation (VO). Once the VO is established the community can share resources amongst each other. This can be done through defined interfaces in a large distributed network. Each member of the VO is allocated pre-reserved slots to execute their own tasks on the Grid. However, these types of platforms have a number of limitations in terms of technical and bureaucratic restrictions [187]. Many of these grids are controlled by large organisations that already have the infrastructure needed to partake, leaving it difficult for small research groups to join. Furthermore, many of these grids use pre-defined platforms with specific operating systems, application suites and APIs. Thus specific platform requirements across the Grid could not be facilitated easily as it would require re-imaging of all the devices and was generally not feasible.
2.2.3 Computational Clusters

As with Grid Computing, Cluster Computing involves networking large volumes of machines to present a unified computing viewpoint to the end user with a single operating system being installed across all nodes in the cluster and special software is used to deploy and schedule tasks across the cluster. The computational cluster idea was developed in the 1960’s. Clusters are networked over a high-speed Local Area Network (LAN) connection controlled by a single owner. One of the most popular cluster frameworks is the Beowulf, which exploits mass-market personal computers known simply as commodity machines [172]. Powerful programs utilise Message Passing Interfaces (MPI) by organising machines into n-node clusters. This type of computing architecture focuses mainly on developing scheduling algorithms for optimal distribution of computational tasks across the nodes of the cluster [60], [212], [167]. Research by Zaharia et al. analyses the conflict between scheduling fairness amongst users and data locality [208]. They proposed an algorithm known as “delay scheduling”, which involved delaying the execution of jobs according to fairness but cannot achieve significant locality constraints, allowing other tasks to be completed while this delay is in place. Zaharia et al. tested this on a 600-node cluster at Facebook and demonstrated that delay scheduling can achieve near optimally for a wide variety of workloads while also yielding twice the improvement in throughput whilst preserving fairness. However, large clusters that consist of thousands of nodes are often subject to failures within the cluster itself, which adversely affects job completion times and overall cluster performance. This was shown by Zhang et al. who developed a general failure model framework, using data extracted from large cluster computations [213]. This framework models the impact failure will have on a wide range of scheduling algorithms. The results of Zhang et al. demonstrated that current scheduling approaches which ignore failures, experience significant degradation with regard to mean job response times and mean job slowdown.

2.2.4 Computing as a Service

Today’s cloud infrastructures are designed on the principles of offering packaged up computing components as a service. Consumers, who use cloud platforms, lease these services on hourly, daily, monthly, and even yearly plans. The cloud service provider is responsible for maintaining the service integrity according to agreements defined between the consumer and provider. These service level agreements (SLAs) describe the acceptable metrics such as service availability, uptime and even response time.
levels for a user. There are three platforms of cloud computing Infrastructure as a Service (IaaS), Software as a Service (SaaS) and Platform as a Service (PaaS). Our discussion in this thesis focuses on the IaaS platform. Both PaaS and SaaS are outside the scope of this research as the main focus is on live migration which resides in the IaaS platform.

2.2.4.1 Infrastructure as a Service

The Infrastructure as a Service (IaaS) platform delivers the hardware components such as a server, storage, network, and associated software such as operating systems, virtualisation technology and file systems as a service to consumers [23]. This IaaS platform can be described as the leasing or renting of virtual servers and storage as a service over the internet. This enables users to access their resources from anywhere in the world, needing just an internet connection. The provider of the IaaS service to consumers is responsible for the upkeep of the service and ensuring quality constraints such as up-time and availability are met. This agreement known as a service level agreement (SLA) ensures that the quality of service for the user is guaranteed. If the SLA is not adhered to, the service providers are penalised based on the contractual agreement with the users. The IaaS is billable by the hour or on an annual basis. Hourly leasing of server resources is one of the main benefits of IaaS cloud system, allowing users to effectively cater for temporary spikes in demand through resource scaling and limiting cost. Additional offers include paying per hour and releasing servers when the demand has risen. This allows the eradication of previous wasteful practice of ‘over-provisioning’ where individuals had to procure resources over their average need to cater for temporary increases. Amazon’s Elastic Compute Cloud (EC2) is the most famous IaaS provider, along with a number of other providers such as Microsoft Azure, GoGrid and Rackspace. The success of the IaaS delivery of cloud resources to consumers is largely based on virtualisation technologies, which allows for multiple virtual machines containing similar guest operating systems to be deployed in apparent isolation on the same physical host. A more extensive discussion on Virtualisation technologies is covered in Section 2.2.5.

2.2.5 Virtualisation Technology

Virtualisation is the replication of an entire architecture in software which provides the illusion of a real machine to all software running above it [115]. It enables larger physical servers to be partitioned into multiple small virtual machines in the cloud.
These machines are insulated from each other with respect to fault tolerance, security and operating system heterogeneity. However, interference of virtual machines on another co-located virtual machine can cause degradation problems. Xen is a popular open source virtualisation framework, which supports a wide range of guest operating systems and is used by a large number of cloud providers including Amazon Web Services. IBM first experimented with Virtualisation in the 1960s where it was first applied to mainframe computers. In 1967 and 1968 IBM created the first hypervisor, the CP-67, which allowed memory sharing between virtual machines. Prior to the development of modern virtualisation technologies, many high-end servers were underutilised as each server was responsible for the management of a specific piece of software or application component. Companies would have to separate servers for different applications and functions, such as a print server, mail server and domain server, which led to enormous under-utilisation of the available resources even though it was considered best practice at the time. The reason why this was done was to isolate the possibility of system-wide failures due to a single fault or bug proliferating throughout the server. However, with the advancements of today’s virtualisation technology, it is possible for multiple applications to reside in apparent isolation on the same physical resource, secure from each other. Isolation makes it very difficult for malware infection to spread to any other virtual machine. Virtualisation enables a single physical server to host all ones application and business functions securely on the same host machine, reducing the server overhead significantly. The Virtualisation technology is made up of a number of sub-domains as follows;

1. ‘Full virtualisation’ is the complete simulation of the actual hardware, allowing software which usually includes the guest operating system to run completely unmodified as if it were running on the physical machine directly.

2. ‘Para virtualisation’ requires software modifications to successfully run in the environment. Special operating system images have to be used in order to deploy software successfully under this framework. However, most new operating systems contain updated kernels and special drivers to allow the operating system to be deployed in a para-virtualised environment unmodified.

2.2.6 Virtual Machine Monitor - Hypervisor

Virtualisation works by abstracting physical hardware and devices from the applications running on that hardware. The process of virtualisation manages and provisions
Virtualisation required the use of a hypervisor also known as the ‘Virtual Machine Monitor’ (VMM), which controls the sharing of resources amongst virtual machines and has direct access to the underlying hardware [9], illustrated in Figure 2.2. The hypervisor creates a single instance known as a virtual machine (VM). The virtual machine is an emulation of a computer system upon which the operating system and subsequent applications reside. The physical hardware that a hypervisor or VMM runs on is usually referred to as a host machine (or server). There are two types of hypervisors: Type 1 and Type 2 hypervisors.

A Type 1 hypervisor runs directly on the host machine’s physical hardware, and
it’s referred to as a bare-metal hypervisor. Type 1 hypervisor does not have to load an underlying OS first, with direct access to the underlying hardware and no other software. A Type 2 hypervisor is typically installed on top of an existing OS, and known as a hosted hypervisor because it relies on the host machine’s pre-existing OS to manage calls to CPU, memory, storage and network resources.

Each type of hypervisor serves similar purposes to be inserted between the server’s hardware and the operating system, allowing for the deployment of multiple virtual machines in isolation and decoupling the operating system from the physical host. From the perspective of the operating system, the virtual machine represents a completely isolated environment that looks and feels just like a physical machine.

Xen and Linux Kernel Virtual Machine (KVM) [9] [114] are examples of hypervisor technologies. The Xen hypervisor is a software controlled solution which is inserted between the server’s hardware and the operating system. The hypervisor is responsible for memory management and CPU scheduling of its guest virtual machines. The Xen project has been supported by industry and academia alike, with partners such as Citrix, IBM, Intel, Hewlett-Packard and more, contributing to its development and production. Amazon’s EC2 has implemented the Xen as its virtualisation technology. The Kernel Virtual Machine (KVM) is a Linux based domain level hypervisor which supports paravirtualisation of the physical host and leverages the x86 virtualisation extensions to add VMM hypervisor capability to Linux [114]. The KVM supports a number of guest operating systems, including Microsoft Windows, Mac OSX, and numerous flavours of Linux. One of the main performance challenges encountered by the VMM relates to migrating resource or VMs from one server to another server to maximising resource usage on all servers and avoid degradation of performance. This Thesis focus on live migration as it is the most adapted approach in cloud data centres.

2.2.7 Live Virtual Machine Migration

Live migration is a process of copying memory pages of a virtual machine from one physical machine to another as illustrated in Figure 2.3. Live migration is controlled by the Virtual Machine Monitor (VMM). Live migration is performed in two phases; iterative pre-copy phase and stop-and-copy phase. These two approaches are designed for service migration with continuous transfer of files and minimal disruption to users but they differ in implementation.

In the pre-copy phase the VM is still running on the source host while its memory pages are copied to the destination host. Every single page that has been modified
after its previous transfer has to be copied again this is referred to as the dirty rate. Hansen and Jul pre-copy based self-migration technique was the catalyst for the popularisation of live migration [81]. In the past decade the pre-copy technique has been a widely examined topic [124], [142]. Clark et al. proposed a pre-copy approach that runs on top of the Xen VMM [42]. They introduced and analysed the concept of writable working set, and presented the design, implementation and evaluation of high performance OS migration. Zhang et al. used hash based fingerprints to find similar pages to be copied at a time to reduce total data transferred during migration [214]. VM-Flock performed the non-live migration of VMs using memory duplication while Riteau et al. optimized the live migration of a single VM over wide area network (WAN) [155]. Wood et al. developed a model known as “Cloudnet” to optimise pre-copy WAN live migration, by optimising both disk and memory over high latency internet links with low bandwidth [198].

Deshpande et al. examined the idea of moving multiple VMs from one physical server to another [49]. The stop and copy phase terminates the VM on the source host after a certain amount of iterations of memory page copying and the remaining unsynchronised memory pages of the VM are then copied over to the destination host. Stop-and-copy was proposed by Hines et al. for para virtualised VMs on Xen
hypervisors [86]. This approach reduces the total data transferred from source to destination server by implementing a dynamic self-ballooning technique, by using free pages in the guest VM and releasing them back to the hypervisor. Hines et al. mentioned the idea of combining both pre-copy and post-copy in their work.

Deshpande et al. research shows the higher the bandwidth available the faster the transfer of memory pages [48]. Dirty rate is the rate at which a VM’s memory pages are modified. Thus, the dirty rate has a direct effect on the number of pages transferred in each iteration. The higher the dirty rate value, the more data will be transmitted, which in-turn will lead to longer total migration times. This in-turn results in a long down times for VMs. However, the relationship between the dirty rate and live migration is not linear because of the stop condition in the stop-copy phase. If the dirty rate is lower than the link capacity, then faster migration will take place. However, if the dirty rate is close to the value of the link capacity and the migration process is in the final iteration, a huge amount of data will be transferred on the final stage. This leads to an increase in modified pages being transferred and a long stop-copy phase. Therefore the downtime will increase proportionally to the growth in the number of modified pages transferred in the stop-copy phase. High dirty rates also lead to an increase in total migration times, due to the fact that more pages need to be transferred in each pre-copy phase round.

In the first pre-copy iteration the entire allocated memory of the VM memory is copied to the destination physical machine. The first pre-copy phase has a direct effect on the total migration time based on the memory size of the VM. Total migration time can be assumed linear, based on the memory size of a VM. However, a low dirty rate downtime is also the same regardless of memory size as the migration copies all of the dirtied pages in a successive iteration, which results in a shortstop and copy phase. When network bandwidth links are unable to provide enough resources for the dirty rate, a large VM will suffer longer down times because the physical pages need to be copied in the stop and copy phase. In the stop-and-copy phase, the virtual machine being migrated is suspended and pages dirtied during the last iterative pre-copy round are transferred at a rate equal to the available network capacity.

Researchers have suggested various performance metrics to measure the effects of live migration. Voorsluys et., al. highlight that service levels of running applications could be affected when they are migrated [190]. They demonstrate that it is very important to migrate an OS with minimal downtime. Their experiments consisted of running VMs in Xen hypervisor during live migration. Kuno et al. examine the performance of live and non-live-VM-migration, where the latter stops the VM to ensure
no performance degradation whereas the former keeps VMs running and performance may degrade [118]. Their results showed that memory writing and physical machine OS communication are important factors for performance degradation. Kuno et al. highlight that migration time and the memory size of VM is proportional and in both methods migration time is almost identical. Their results also show that live migration provides better performance when running a CPU intensive task if network speed is high.

The performance of live migration can be evaluated in terms of two metrics; total migration time and downtime. Total migration represents the total time between the initiation and the completion of the pre-copy and stop and copy phase along with the reservation of resources and activation of a new virtual machine on the destination host machine. Total Migration Time is explained as follows:

1. **Total Migration Time** is the summation of all migrant VM’s migration time. Its value can vary due to the amount of data to be moved during migration and migration throughput. It depends on 1) the total amount of memory transferred from source to destination server, and 2) allocated bandwidth or link speed. equation 2.1:

   \[ t_m = \frac{VM_s}{bs} \]  

   where \( VM_s \) is the size of the RAM of the VM to be migrated and \( bs \) denotes the bandwidth speed.

2. **Downtime** is the time when service is not running or available due to migration of processor states and is defined as follows:

   \[ t_d = \frac{d \times l \times t_n}{bs} \]  

   The down time \( t_d \) is dependant on the dirty rate \( d \), the page size \( l \), the duration \( t_n \) of the last pre-copy round, and \( b \) denotes bandwidth speed.

Equations 2.1 and 2.2 highlight that memory size of VMs and network congestion will produce unpredictable performance live migration. Both parameters have a major impact on the amount of bandwidth resource available for live migration. The speed of live migration is closely aligned with the network traffic conditions. Live migration consumes large amounts of network bandwidth and when bandwidth is depleted this will lead to slower total migration times. When a number of VMs have
to be migrated from a physical machine, it is important for the virtual machine migration to schedule the migrations at optimal times, in order to minimise the impact on both the infrastructure and quality of service delivered.

Even though live migration has attracted attention in the cloud computing community in recent years, it has primarily focused on power management and the efficiency of physical hosts, such as Beloglazov et al. and Verma et al. [14] [188]. Recently, researchers have been looking closely into how network traffic contributes to long delays when migration of VMs occurs.

Chen et al. proposed a novel migration strategy that quantifies the benefits and cost of VM migration and VM placement to a network link load in data centres [41]. Their MWLAN strategy factors in the current work link load and the link bandwidth code to improve the efficiency of live migration. Sarker et al. propose an ad-hoc heuristic approach to schedule migrations [163]. Their heuristic approach reduces the total migration time according to the network topology and a fixed dirty pages rate and is measured against a custom algorithm that schedules the migrations randomly with regards to their theoretical completion time.

The number of VMs being migrated at any particular time has a major impact on the current bandwidth link resource availability. Migration of VMs can happen in two ways: sequential or parallel. Sequential migration moves VMs one at a time off the source host to a destination host whereas parallel migration moves a group of VMs all at once. Bari et al. propose CQNCR a technique for determining the execution order of massive VM migrations within data centres [10]. CQNCR schedules the sequences of each VM migration so as to efficiently reach the optimal configuration of VMs to be migrated in order to minimise total migration time and impact on performance. Yao et al. designed an Individual Performance Guaranteed VM Migration (IPG-VMM) algorithm which restricts parallel migration within the same server rack space with the goal of maximising the migration bandwidth by minimising the correlations between migration schemes of different VMs and guarantee the migration performance of individual VMs [206]. Wang et al. consider a non-blocking network and the multi-path routing feature made available by SDN controllers to increase the migration bandwidth [191]. Haikun et al. propose a migration performance model based on the memory transfer algorithm implemented in Xen [122]. They consider both static and refined dirty pages rate, building on historical observations and assume that the Writable Working Set size should be transferred in one round thereby determining the VM downtime. Hirofuchi et al. implemented the pre-copy migration algorithm in the Simgrid simulator [88]. They reproduce the memory dirty pages generation
behaviour by using a single rate but with unusual linear correlation on the CPU usage. In this thesis for simplicity we only consider sequential migration, as parallel migration has shown to be to demanding on computing resources, resulting in worse network performance from the cloud provider’s [37].

Piao et al. presented a network-aware VM placement and migration approach for data-intensive applications [146]. They developed a model that places a VM on a host machine, taking into account the network conditions between the source host and destination. Stage et al. proposed a migration and scheduling model, while considering bandwidth requirements and network topology [171]. Chen et al. researched coordination of multiple VM migration while sharing network links and global bandwidth resource [40]. Ghorbani et al. presented a method for guaranteeing network bandwidth. However these approaches come at a cost of high network resources utilisation [72].

2.2.8 Live Migration & Unpredictability

Unpredictability of resource availability is a major concern within the cloud community and is still a major obstacle for organisations switching from traditional in-house servers to becoming fully hosted on the cloud. Tchernykh et al. discussed the main sources of unpredictability in cloud computing, [179]. They list a variety of unpredictability types such as virtualisation, Jobs arrival, Migration, Energy consumption and Scalability. Tchernykh et al. also detail a number of parameters which lead to unpredictability of a host performance such as Network capacity, Resource capacity, Data transfer time, effective bandwidth and effective performance. Both Graubner et. al. and Butikofer et al. discuss in detail how unpredictability in network resource consumption and users can have a major effect on a data centres performance [76] [31]. Most applications deployed in the cloud require the availability of communication resources for information exchanged between tasks. However, this is a concern for cloud providers as they might not know the quantity of data that can be managed or quantity of computation required by tasks. This in-turn presents challenging issues for cloud management systems as to how much resources are required to satisfy future demands, how to determine which physical machines to place the VM on, and how to determine if a host is over/under-utilised. Each of these processes is affected by unpredictable dynamic workloads affecting energy consumption, SLAs and migration performances. Much research has been conducted in these areas to improve the performance of live migration. Hacking et al. presented a design, implementation and evaluation of a system which supports the transparency of live migration of virtual
machines and minimises unpredictability [79]. Their system minimises service disruption over LAN and WAN networks. They implement a delta compression algorithm to transfer memory and do not interrupt the open connection to and from VM during migration.

2.2.9 Predicting VM Provisioning, Placement & Consolidation

The cloud resource management system in a IaaS platform is required to manage the mapping the allocations of resources to VMs and allocation of VMs of to host machines. The main goal of VM provisioning is to ensure sufficient resources are provided to meet users expected QoS. For this purpose cloud providers such as Amazon EC2, for instance, provide a variety of VM instances with different amounts of resources.

Unpredictability in cloud resource usage can affect the provisioning of cloud resources. Predicting resource usage in the future can help physical machines cope with real-time demands. Gong et al. propose an approach for VM resource allocation based on resource demand predictions [75]. Their resource demand predictor is based on two techniques: Fast Fourier Transform to find periodicity or a signature of resource demand and the state-based approach using Markov chains. Their approach predicts the long-term predicted usage of a physical machine's resources. Once a resource is deemed depleted, live migration is instantiated to provide relief to the physical host. Farahnakian et al. presented a heuristic algorithm for VM placement based on a linear regression algorithm which has the ability to detect if a host is under or over-utilised [62]. They implement a linear regression algorithm on each host server machine which monitors the current CPU usage of a physical server machine and forecasts the short term CPU utilisation of that particular server. If the predicted CPU utilisation is larger than the CPU capacity, the server is considered over-utilised, otherwise if the CPU is less than a defined threshold, the server is considered under-utilised. Their approach predicts for the short-term.

Khatua et al. propose an approach for resource prediction multiple times steps into the future [112]. They apply an Auto-regressive Integrated Moving Average (ARIMA) model for horizontal scaling of cloud resource. Their model detects if over-utilisation will occur based on a threshold value and switch on more VMs to help deal with the workload demand. Ashraf et al. propose a load prediction technique for VM resource allocation and the admission control of multi-tier web applications in cloud computing [8]. Their prediction model is based on a two-step procedure. First,
Exponential Moving Average (EMA) acts as a load tracker to construct a representative view of the workload by filtering out noise. Second, a linear regression model is applied where the inputs are the load tracker signature and provides an output of the predicted load $n$ time steps into the future.

Resource unpredictability, however, can cause a lot of disruption to the allocation of resources to VMs on the physical machine. Excess use of the resource by co-located virtual machines can lead to a physical host machine’s resource being depleted. This is known as over-utilisation in host machines and is one of the key factors that initiate live migrations. To improve live migration over-utilisation detection methods are implemented. One of the simplest over-utilisation detection methods is based on an idea of setting a CPU utilisation threshold. This method distinguishes the non-overload and overload states of the host by comparing the current CPU utilisation of the host with the defined threshold. If the threshold is exceeded, the algorithm detects a host overload. However, this approach is not able to deal with changeable workloads.

In the Live migration domain, there has been a move towards applying adaptive algorithms to efficiently manage unpredictable workloads in a cloud environment. Wood et al. propose an approach known as “sandpiper” for detecting live migration of VMs from over-utilised physical machines to non-utilised machines [199]. Their approach determines if a host is over-utilised based on the previous times steps. They also employ a “greedy” algorithm to migrate the heaviest loaded VMs to the least utilised hosts.

Beloglazov and Buyya conduct a comparative analysis of optimal online deterministic algorithms for detecting over/under utilisation techniques based on historical resource usage data [15]. Their work proposes an adaptive usage threshold base on statistical parameters of previous data such Median Absolute Deviation (MAD) or inter-quartile range (IQR). The authors also apply a local regression algorithm for predicting future CPU usage value, and also consider migration penalties in decision making.

Andreolini et al. propose an approach for detecting over-utilisation, when a physical machine is deemed to be over-utilised where there is a substantial change in the load trend [4]. Their research avoids unnecessary live migrations by analysing past historical data.

Beloglazov and Buyya propose an approach for detecting migration for physical machine based on a Markov chain and optimisation of inter-migration time with QoS constraints [16]. The goal of their approach is to find a solution that optimises the
inter-migration time while keeping the Over-utilised Time Fraction (OTT) metric
within a certain range. Beloglazov and Buyya apply a sliding windows approach, to
take into account the dynamic and non-stationary workloads.

Arianyan et al. propose a dynamic over-utilisation technique [5]. Their algorithm
VDT computes utilisation of a host based on the number of VMs on a host and the
current CPU utilisation. The VDT algorithm determines a new host for each VM to
be migrate to, based on the minimum CPU utilisation of each destination host.

The VM Consolidation approach is another method to utilise the power of live
migration, in order to save energy. Dynamic VM consolidation (DVMC) reduces
energy consumption in a data centre. This is done by moving multiple VMs on to a
single host or switching idle host into power saving modes.

There is two types of VM consolidation; static and dynamic. In static, the virtual
machine manager assigns resources to VMs based on peak loads (over-provisioning),
however, workloads are not always at peak levels, this then leads to resource wastage.
In dynamic consolidation (DVMC), VMs are configured based on their current re-
source requirement, which is known as “re-sizing”. This can lead to more efficient
resource usage. DVMC periodically consolidates VMs, however depending on the
time of day and if the network traffic is high, it can cause issues for current network
resource available. Intelligent DVMC techniques not only analyse current network
resource consumption but also have the ability to predict future demand levels of the
destination host so that each of the VMs acquires the resources they require during
critical times in a data centre. Khanna et al. propose an approach for the dynamic
consolidation of VMs based on live migration. They utilise live migration to minimise
the number of physical machines running in a data centre [110]. They achieve this by
ordering VMs in a non-decreasing fashion of their resource usage and then migrates
the least utilised VMs to the least utilised physical machines. This reduces the affect
migration has on the network load. Ferreto et al. presents an approach known as
dynamic consolidation with migration control [65]. Their research formulates the con-
solidation problem as a linear programming problem with constraints implemented
to prohibit the migrating of VMs with steady workloads. Their results show that
they can lower the number of VM migrations by increasing the number of physi-
cal machines. Gong and GU propose a dynamic consolidation approach known as
Pattern-driven Application Consolidation (PAC). This approach is based on extract-
ing patterns of resource usage called signatures using signal processing approaches
such a ”Fast Fourier Transform” and dynamically time warping (DTW). Based on
the extraction signatures they perform dynamic placement of VMs to physical
machines, which have the highest match between VM resource usage signature and host
free capacity signature. Gong and GU research focuses on a global consolidation of
VMs resource usage.

The VM selection phase is applied once a host is deemed to be over-utilised.
Beloglazov et al. suggest three different approaches to select a VM [17]. The first
approach named Minimisation of Migrations (MM) moves VMs to under-utilise host.
The first step of the algorithm orders the VMs in descending order based on CPU
usage. A VM is selected based on an upper threshold value. The second algorithm
which is named the Highest Potential Growth (HPG), categories the VMs based on
the likelihood that they will increase SLA Violation. The third algorithm selects a
VM at Random Selection (RS). Al-Ayyoub proposes a Host Fault Detection (HFD)
algorithm that selects a VM to migrate from the over-utilised host based on the
maximum impact on the cause of the overload [3]. If the host’s over-utilisation is
caused by RAM then the VM with the maximum allocated RAM is selected by the
algorithm.

The VM placement process involves the selection of the most suitable host to
deploy a virtual machine which has to be migrated from the source host. VM place-
ment techniques usually focus on the efficient placement of VMs in terms of power
consumption, resource maximisation and load balancing between hosts. However,
VM placement techniques must not just place a VM on a host but also satisfy cur-
rent QoS and ensure adequate resource are provided into the foreseeable future to
prevent live migration of the same virtual machines. Beloglazov et al. propose an
energy-aware heuristic algorithm to dynamically allocate VMs to a physical machine
based on live migration [13]. Beloglazov et al. decide a physical machine is either
over/underutilised based on whether the CPU is higher or lower than the specified
threshold. The authors apply a Best Fit Decreasing heuristic to place VMs on to less
hosts, which also takes into account the power increase to each host.

Xu et al. propose a VM placement strategy through a multi-objective optimisa-
tion problem to reduce resource waste-age and reduce energy [204]. This approach
incorporates a fuzzy multi-objective evaluation to search for a solution for allocating
VMs. Horri et al. propose a host Utilisation and Minimum Correlation (UMC) VM
placement algorithm to reallocate VMs from over/underutilised host machines [90].
The UMC approach considers the correlation between the resource of a VM and the
VMs present on the destination host. The algorithm selects the host with the lowest
correlation between host machine and selected VM.
All of the above research either applies a threshold or a heuristic algorithm to help improve live migration. However, the unpredictable nature of cloud resource consumption has led researchers to explore more adaptive and pro-active methods which have the ability to analyse previous data and learn from it. In the next section, we discuss machine learning and Artificial Intelligence algorithms which have been applied to cloud computing.

### 2.3 Machine Learning & Cloud Computing

Artificial Intelligence (AI) is defined by Charniak and McDermott as “The study of mental faculties through the use of computational models” [39]. Machine Learning (ML) uses statistical techniques to give the computer systems the ability to learn with data without needing to be explicitly programmed [161].

One potential solution to mitigating unpredictability during live migration is to design an autonomous model which has the ability to make decisions based on real-time information. AI and ML algorithms have been applied to a range of applications such as improving energy systems [127], forecasting energy demand [132] optimising power generators [128][130], improving water management [134], gaming purchase [202], multi-agent systems[108], patter restore method [183] click-stream data [182] and online gamer engagement [203]. Recently there has been a move towards integrating AI and ML techniques to improve the overall efficiency of a cloud data centre. Feller et al. present a VM placement approach based on the Ant Colony Optimisation meta-heuristic [64]. VMs are placed by modelling the workload consolidation problem as an instance of the Multidimensional Bin Packaging problem. The main goal is to pack as many VMs into fewer host machines. Wu et al. implement a genetic algorithm to address the placement of VMs on host machines, in order to reduce energy consumption [201]. Their approach considers two functions, the first being a linear function of the energy consumed by the workload and energy consumed by the idle server. The second function considers the amount of data exchanged among VMs that display energy consumed by the network. Ismaeel and Miri developed a prediction model using K-means clustering and Extreme Learning Machines (ELM) to estimate future VM requests in a data centre, using historical patterns [99]. The research evaluates its model on the Google cluster data. This approach creates various categories of VM clusters with the goal of developing a prediction model for each of these clusters. They utilise the K-means algorithm to create a cluster of VMs.
Then for each cluster, an ELM workload prediction model is designed to estimate the number of requests for each cluster that could arise in the future.

AI or ML learning based techniques such as Reinforcement Learning or Neural Networks can address a number of issues related to live migration. Firstly, these algorithms can reason over the inherent uncertainty observed and devise strategic plans in order to optimally choose actions or make predictions which maximises QoS in a data centre. The second benefit is that algorithms such as reinforcement learning have the ability to learn without prior knowledge, operating directly within the environment, which will reduce the need for expertly defined thresholds and mappings for each state of the environment. Neural Networks, on the other hand, have the ability to learn from past information to detect patterns which can improve predictions for the future. In the cloud domain, there has been a move towards integrating AI and ML algorithms to improve the optimisation of problems, which are discussed further in section 2.3.1 and section 2.3.2.

2.3.1 Reinforcement Learning & Cloud Computing

Reinforcement Learning is a method which has been shown to produce good results in optimising cloud resources. Barrett et al. implemented a novel reinforcement learning approach to optimising the allocation of resources to support application scalability in cloud computing environments [11]. Their approach applied a simple state-action space, capable of enabling the learning agent through the Q-learning algorithm to produce good VM allocation policies in the IaaS clouds with no prior experience. Their approach implemented a parallel reinforcement learning approach to deal with the ‘curse of dimensionality’ (high dimensional space), thus reducing the time taken to converge to optimal resource allocation policies. Each learning agent attempts to approximate optimal policies based on its experience. Through the agents sharing of information, the time taken to converge to a stable policy is greatly reduced.

The research by Gerald Tesauro from the IBM J Watson scientific research centre examined the problem of over-provisioning of server resources and proposed a solution through autonomic computing methods [180]. They implemented a “Trade 3” application, (simulation of a real-time trading platform application) and model an open loop Poisson process for users requested. The paper compares a queuing theoretic model based solution, a static allocation, a random allocation and a model-free SARSA(0) reinforcement learning algorithm. The results show that when allocating a Trade 3 application to virtual servers both the model based queuing algorithm and the model-free SARSA(0)(State Action Reward State Action) methods
performed best, with no performance difference between them. Tesauro suggests that the SARSA(0) requires less of the system knowledge than queuing approaches. In addition Tesauro suggest that SARSA(0) would perform better in scenarios where user behaviours change. However, this would require complete re-learning of value functions should the service-level utilities change, where the queuing models would not require re-training but could lead to “model drift” (model drifts away from initial training model) making the approach less accurate over time [180].

Tesauro later extended this work, by investigating the efficacy of a hybrid reinforcement learning technique for autonomic resource allocation which combines model-based approaches with online reinforcement learning [181]. This approach differs from his previous work as it involved training the reinforcement learning algorithm offline using a data-set generated from a good model-based policy. This in turn provides a solid offline training environment for the algorithm, removing the initial poor performance of the reinforcement learning algorithm at the initial exploratory stages. This allowed the agents to perform well once the algorithm goes live.

Perez et al. applied reinforcement learning to Grid computing in-order to optimise resource allocation [145]. Their research focused on optimising when the users submit jobs and grid to provide sufficient resources. The Virtual Organisations consist of a set of individuals or organisations that share resources and data in a conditional and controlled manner.

Galstyan et al. proposed a decentralised multi-agent reinforcement learning approach to Grid resource allocation by [68]. The learning agents had incomplete global knowledge, and had no communication with other agents. The results showed that the reinforcement learning approach was capable of improving the performance of a large scale grid.

Dutreilh et al. developed a Q-learning approach for allocating resources to applications in the cloud [55]. Their work used a reinforcement learning agent to decide when to add and remove virtual machines based on variable workloads. The authors applied convergence speedups by initialising the Q function to further optimising their results.

Rao et al. developed a reinforcement learning approach to virtual machine auto-configuration in clouds [153]. Their learning agent responded to changes in demand for resources to reconfigure the virtual machine. Their results showed that the approach was able to determine near-optimal solutions in small-scale systems.

network comprised of irregular typologies and the usage patterns were unpredictable. The main goal of Boyan and Littman research was to learn a routing policy which minimised the number of hops each packet takes against the possibility of congestion along a particular route. They implemented a discrete environment to model the movement of network packets in the network. The authors compared the Q-learning algorithm with the shortest path approach. For each experiment conducted in the research, the Reinforcement Learning agent was able to sustain a greater amount of network traffic than the non-learning approach.

Shaw et al. applied Reinforcement Learning agent to reduce energy consumption within data centres [165]. Their approach named ‘Advanced Reinforcement Learning Consolidation Agent’ (ARLCA) is capable of optimising the distribution of virtual machines across the data centre. From their results they showed that their ARLCA approach was able to reduce energy and service level violations while also outperforming algorithms proposed by [15].

Malialis et al. proposed a novel Multiagent Router Throttling approach to respond to service attacks over the Internet. Their results showed that the approach used can significantly scale-up using hierarchical communication and coordinated team learning for network intrusion response. They incorporated a reward shaping method known as ‘difference rewards’ and show that the scalability of their system is significantly improved in experiments involving over 100 reinforcement learning agents [126].

Reinforcement Learning is concerned with what actions to take based on the current state in order to maximise rewards. However, as resources change, the cloud system must be able to predict when these change will happen in order to reconfigure resources. This is what has prompted many researcher to apply a powerful prediction algorithm known as ‘Neural Networks’ to further enhance machine learning in the cloud domain.

### 2.3.2 Neural Network & Cloud Computing

This section covers a number of application areas where neural networks have been applied to predict resources or events.

Neural networks are one of the most effective and versatile machine learning algorithms currently available and have been successfully applied to areas of cloud computing such as scheduling [57], intrusion detection [189], DDoS attack defence [105] and load demand forecasting [149]. Further general areas where neural networks are applied to can be found in the following: [132, 196, 93, 103, 107, 185, 144, 123, 210, 125, 136, 111, 59, 113]
Tani et al. examined the use of Artificial Neural Networks and their abilities to solve the problem related to CPU scheduling for Cloud Computing. They implement a set of different neural networks. The performance of each neural network was evaluated in regard to the performance of the scheduling algorithms in the Cloud. Their results showed that a ‘Multi-Layered Perceptron’ (MLP) neural network outperformed other networks by optimising CPU scheduling thus reducing the average waiting time of tasks on the execution queue and stimulating the response time.

Saied trained an Artificial Neural Network (ANN) algorithm to detect TCP, UDP and DDoS attacks based on characteristic patterns that separate genuine traffic from DDoS attacks [160]. Their experiment evaluated the ANN effectiveness by identifying different types of DDoS attacks when they were injected into normal traffic flowing through the network. Once their ANN was trained as a detection mechanism, it was then tested against known and unknown DDoS attacks. To test the effectiveness of their approach they evaluated their solution with signature-based and other related academic research.

Duy et al. employed a neural network predictor for the optimisation of server power consumption in a data centre [57]. The neural network exploits the historical workload demands to predict future load demands based on collected historical data. The cloud management system uses the neural networks prediction policy to decide on how many servers should be working at a given time, in order to reduce the energy consumption of a data centre. Through simulations with the use of the ClarkNet and NASA workload data-sets, they show that the Neural Network can reduce power by 46% and 12% for respective data sets.

Prevost et al. examined neural networks along with an auto-regressive linear predictor algorithm to forecast future workloads to determine the optimal number of servers that should be running to deal with current demand [149]. Their work investigates that where a client requests multiple VMs to run a certain application and rather than reserving just a fixed number of virtual machines their approach adjusted the need for resources for each application based on predicting the application’s demands.

Janardhanan et al. implement a Long-Short-Term-Memory (LSTM) neural network to predict day long CPU usage of host machines. They utilise the Google cluster trace data set and compare their results with an Autoregressive Integrated Moving Average (ARIMA) modelling approach [102]. Based on their results the LSTM outperformed the ARIMA approach by reducing the error in the prediction of CPU load of up to 20 % when comparing the two model’s accuracy.
Cao et al. utilised an ensemble model & fuzzy neural network optimised using a subtractive clustering algorithm for Self-Adaptive prediction model for estimating cloud resource demands [35]. The training phase of the neural network is optimised using self-adjusting learning rate and momentum weight to improve testing performance.

2.4 Summary

This chapter reviewed the relevant literature and theoretical components across the field of cloud computing and AI. More specifically, it first introduced the history and background of cloud computing and the unpredictable nature of the live migration process. Secondly, this chapter focuses on the domain of AI, in particular Reinforcement Learning and Neural Networks and how they have shown to improve performances in data centres.
<table>
<thead>
<tr>
<th>Authors</th>
<th>Technique</th>
<th>Reconfig</th>
<th>Resource allocation</th>
<th>Scheduling</th>
<th>Energy</th>
<th>SLA</th>
<th>Prediction</th>
<th>Security</th>
<th>Server</th>
</tr>
</thead>
<tbody>
<tr>
<td>Feller et. al [64]</td>
<td>ACO</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>x</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Wu et al. [201]</td>
<td>ACO</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>x</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Ismaeel &amp; Miri [99]</td>
<td>K-means</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>x</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Barret et al. [12]</td>
<td>RL</td>
<td></td>
<td></td>
<td></td>
<td>x</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Teasuro et al. [180]</td>
<td>SARSA</td>
<td></td>
<td></td>
<td>x</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Teasuro et al. [181]</td>
<td>RL</td>
<td></td>
<td></td>
<td>x</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Perez et al. [145]</td>
<td>RL</td>
<td></td>
<td></td>
<td>x</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Galstyan et al. [68]</td>
<td>Multi RL</td>
<td></td>
<td></td>
<td>x</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Dutreilh et al. [56]</td>
<td>RL</td>
<td></td>
<td></td>
<td>x</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Rao et al. [153]</td>
<td>RL</td>
<td>x</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Shaw et al. [165]</td>
<td>RL</td>
<td>x</td>
<td></td>
<td>x</td>
<td>x</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Tani et al. [177]</td>
<td>Neural Network</td>
<td></td>
<td></td>
<td>x</td>
<td></td>
<td></td>
<td>x</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Saied et al. [160]</td>
<td>Neural Network</td>
<td></td>
<td></td>
<td>x</td>
<td></td>
<td></td>
<td>x</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Duy et al. [160]</td>
<td>Neural Network</td>
<td></td>
<td></td>
<td>x</td>
<td></td>
<td></td>
<td>x</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Prevost et al. [149]</td>
<td>Neural Network</td>
<td></td>
<td></td>
<td>x</td>
<td></td>
<td></td>
<td>x</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Janardhanan et al. [102]</td>
<td>Neural Network</td>
<td></td>
<td></td>
<td>x</td>
<td></td>
<td></td>
<td>x</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Zhan et al. [210]</td>
<td>PSO</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Tsai et al. [185]</td>
<td>DE</td>
<td></td>
<td></td>
<td>x</td>
<td>x</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Chapter 3

Reinforcement Learning & Neural Networks

3.1 Introduction

This chapter outlines the background of the machine learning algorithms applied to the cloud environment in this Thesis. It discusses both Reinforcement Learning and Neural Networks and provides a detailed review of the application areas that these algorithms have been applied to.

3.2 Reinforcement Learning

The first learning algorithm we examine in this chapter is ‘Reinforcement Learning’ (RL) [158]. Reinforcement learning is a subset of Machine Learning [74]. Reinforcement Learning enables an agent or multiple agents to learn through the direct interaction with their ‘environment’. For instance, an agent learning to escape a maze must first interact with its environment to select correct actions and avoid incorrect actions, in order to find the correct path out of a maze. Through the interaction with the environment, the agent updates their experience in a trial and error manner to build up a model of its environment. The agent will be rewarded for each action (step) it takes in the maze, and the least amount of wrong actions in a maze the greater the reward will be after each episode.

3.2.1 Markov Decision Processes

Markov Decision Processes (MDPs) [21], [150] are mathematical frameworks that are particularly suited to modelling problems under uncertainty [53]. Markov processes satisfy a constraint known as the “Markov property”, which says that the “current
state” in an environment is required to make predictions about “future states”, for example a robot learning to walk could be in a standing state and a future state could be to move the left or right leg. All of the previous states which have to lead up to the current states are irrelevant, making the process ‘memoryless’. A good example of this process is the game of checkers. At any given time the checker’s piece configuration on the board is all the information a player will require to make the next move [174].

MDPs are represented as a 4-tuple (S, A, P, R) as explained as follows:

1. \( S \) denotes the set of all possible states of an environment

2. \( A \) represents the action space, which defines the set of actions that are allowed to be executed within each state.

3. \( P(s' | s,a) \) is known as the transition probability (\( P \)) i.e., this function measures the likelihood an action (\( a \)) will lead to a future state (\( s' \)).

4. \( R(s|a,s') \) is the reward signal (\( R \)) received at each time step. This reward signal can be either positive or negative and is received after each action performed in a particular state.

The transition probability (\( P \)) to the next state \( s' \) is defined as:

\[
P^a_{s,s'} = P\{s_{t+1} = s | s_t = s, a_t = a\} \tag{3.1}
\]

where \( s \) is the state, \( a \) is the action and \( t \) is a specific interval.

In Reinforcement Learning a reward signal help guide a learning agent to make the correct decisions. The reward function is often represented as a single scalar value as follows:

\[
R^a_{ss'} = E\{r_{t+1} | s_t = s, a_t = a, s_{t+1} = s'\} \tag{3.2}
\]

where \( s \) is the state, \( a \) is the action, \( s' \) is the next state and \( E \) is the expected reward. The goal of a learning agent is to maximise the rewards over a long term, even if this is at the expense of the short term gains [176].

The problem with the MDP property is that its performance often suffers from a problem known as the ‘curse of dimensionality’. This refers to the exponential increase in the size of the problem domain, as additional states and actions are added to an environment. Thus, designing the appropriate state space representation is a
challenging problem, which will have a direct impact on the performance of a learning system. An MDP computes a function known as a state value $V(s)$, estimates for all states. This state value informs the agent "how good" it is to be in a particular state i.e. the future reward that is to be expected from the current state. Discovering the optimal solution or the optimal policy $\pi$, for the MDP is the main goal of the value function. This policy $\pi$ denotes the mapping from states to actions. This value function also governs the action whilst in a particular state. The learner’s goal is to achieve the optimal policy to receive the greatest reward. The value function is also known as the Bellman optimality equation ($V^\pi(s)$), is denoted as:

$$V^\pi(s) = \sum a \pi(s, a) \sum s' P_{s,s'}^a [R_{s,s'}^a + \gamma V^\pi(s')]$$

(3.3)

Once the value function is calculated for each state, then the algorithm selects an action at each time step which yields the greatest reward. This is known as the greedy function and is denoted as follows:

$$\pi^* = \arg\max_a \sum s' P_{s,s'}^a [R_{s,s'}^a + \gamma V^\pi(s')]$$

(3.4)

where $\gamma$ represents a discount factor, usually valued between 0 and 1, $\gamma$ trades off the importance of early versus late rewards and also acts to ensure convergence of MDPs which do not have a terminal state. By wrapping the Bellman equation into an iterative technique, this in turn develops a method for solving the MDP known as ‘Dynamic Programming’.

### 3.2.2 Dynamic Programming

Dynamic Programming (DP) is a collection of techniques used to solve a problem by iteratively computing a solution in a recursive manner. This approach was first used by Richard Bellman in the 1940s. The key attributes of the DP technique are the splitting up of a problem into smaller sub-problems, which then can be solved individually. For it to be possible to use DP to solve a problem, it needs to have an optimal substructure and overlapping sub-problems [47]. The optimal substructure refers to the fact that it must be possible to efficiently compute the optimal solution through the combination of the sub-problem solutions. A DP method will store the outcomes to specific sub-problems in a look-up table to optimise the solution. Thus
the next time a sub-problem is encountered, the stored solution can be retrieved from the look-up table, negating the need to re-compute and compare each previous solution at each time. The following paragraphs introduces two dynamic programming methods (Policy Iteration and Value Iteration) for computing solutions to MDP problems.

3.2.2.1 Policy Iteration

The Policy Iteration method begins with a specific policy, which is not optimal and improves the policy iteratively [147]. The policy will determine what action to take in each state by mapping between state-action pairs. DP methods which are used to solve an MDP problem are focused on finding an optimal policy. This can be achieved through the manipulation of the policy directly as opposed to the value function where it finds the policy indirectly [106].

The policy iteration technique has a two-stage process: policy evaluation and policy improvement. The first part of the policy iteration algorithm outlines the steps involved in the approach. First, all of the state’s values are initialised to zero (e.g. S=0). An initial policy is evaluated and the state values under that initial policy are determined via the equation. The approach recursively backs up the current state value estimates and loops until \(|v - V(s)| < \theta\) (which is a very small threshold number).

\[
V^\pi(s) = \sum s' P_{sa}^{ss'} [R_{ss'}^a + \gamma] V(s')
\]

Policy improvement is implemented when a state value function for a given policy \(\pi\) is known and then the policy is adjusted to see if a better one can be found. All subsequent policies must be an improvement upon previous policies. This is achieved by implementing a greedy function, which selects the policy with the highest reward with respect to the state value function estimates computed from the previous policy. This then guarantees that any new policy found will be an improvement upon the previous policy [176]. The improved policy is then evaluated again and the state value function estimates are computed for all of the states. This process will continue until no further improvement can be made to the policy. Thus, if this is true then the policy has converged to an optimal policy (\(\pi^*\)).
3.2.3 Value Iteration

The ‘value iteration’ approach differs from the ‘policy iteration’ as when computing the optimal policy it restricts the number of iterations of the policy evaluation to a single iteration and adjusts the policy based on these values. Often it is not necessary to perform too many sweeps of a policy evaluation. Even after a small number of iterations, the estimates of the policies can be roughly approximated. This in-turn will lead to a more efficient process [176]. As with policy iteration, all states are initialised to zero (S=0). The Value Iteration function then calculates the value of each of the states by computing the maximum valued action achievable from within the current state, which is denoted as:

\[ V(s) \leftarrow max_a \sum a' P_{s,s'}^a[R_{s,s'}^a + \gamma]V(s') \]

(3.6)

If the difference between the successive value estimates \(|v - V(s)|\) is lower than \(\theta\) (a small threshold value) then convergence is assumed. The output value of the value iteration method is computed by acting greedily with respect to the value function estimates.

Over the years DP methods have been changed and applied to different domains. These domains range from robotics [168] to water resource [205]. The main modification to the algorithm focuses on limiting the number of sweeps [151] or prioritising sweeps [141] in which there were large changes in V or \(\pi\).

In the next section, we discuss the reinforcement learning approaches to solving MDPs problems.

3.2.4 Reinforcement Learning Approaches

Dynamic programming approaches work well in MDPs where the environment is completely observable. However, in a real-world situation, complete information is not always available or cannot easily be approximated. In these situations, an alternative approach, known as Reinforcement Learning is required.

Reinforcement Learning involves learning through ‘trial and error’, by utilising corrective stimuli where positive rewards received will indicate relatively good actions and negative rewards received will indicate bad decisions, as illustrated in Figure 3.1. Typically when applying Reinforcement learning the problem is modelled as an
MDP, however, with the difference being that complete information is unavailable, which renders the DP approaches unsuitable for discovering good policies [46]. In Reinforcement Learning, a learning agent’s goal is to maximise the reward it receives in the long run by selecting the action which will yield the greatest reward.

Reinforcement learning is split into two categories: Model Base and Model Free learning. Model base learning utilises statistical techniques to estimate the missing MDP model [78] and then solves the problem utilising DP approaches once the estimates for the missing model have been calculated. Many established techniques exist with approaches such as [175], Certainty Equivalent Methods [117] and Prioritised Sweeping [141] are prevalent in the RL area.

A Model-free approach attempts to estimate a control policy through direct interactions with their environment. By observing the rewards received through actions the agent learns to select the action with the biggest rewards. The most popular algorithms in this area are Q-learning [192]), SARSA [157] and Temporal Difference(θ) [174] have received significant attention from the research community in recent years.

3.2.4.1 Monte Carlo Methods

Monte Carlo (MC) Reinforcement Learning extends upon dynamic programming methods such as the policy iteration technique, through the computing of the value function in domains where only partial information exist [176]. MC methods sample the environment directly and observe how it responds. One of the main contributions
of the MC method is that it learns episodically. The MC learner will only receive the environmental reward after an episode of learning has been completed. Thus, the MDP problem will have to have a termination state for the MC learner to receive a reward. An example is the game of blackjack. If this game was broken down into a finite MDP it would break down into a selection of games. After the result of each game (episode), the player would be rewarded depending on whether the game resulted in a win, lose or draw. The most important property of the MC algorithm is that each episode terminates and the result (or return) is observed at the end of the episode. The MC method functions as policy iterators which is similar to the DP policy iteration approach. An MC method can be used to evaluate a policy for a given MDP, given that a state action space and stationary deterministic policy is defined. This provides an advantage to the MC method over the DP policy as it does not require the complete environment model. An MC method will average the return observed from successive visits to the same states in a current policy $\pi$. The higher the amounts of rewards received, the average value will converge to the expected value, thus estimations of $V^\pi(s)$, which is the state value function under the current policy can be determined.

### 3.2.4.2 Temporal Difference Learning

In 1998 Sutton and Barto designed a suite of methods know as Temporal Difference (TD), which was seen as a significant breakthrough in reinforcement learning [176]. The TD algorithm combines elements of both Monte Carlo methods and DP and is applicable in problems where there is only a partial environment model. TD methods distinguish themselves from the MC model as it does not have to wait until the end of the episode to make predictions. TD methods increment the update value function approximations $V^\pi(s)$, at the end of each time step. TD methods are particularly useful when the tasks are continuous and are not easily broken down into terminable episodes.

### 3.2.4.3 SARSA Learning

State-Action-Reward-State-Action (SARSA) is an online policy reinforcement learning algorithm. SARSA was first introduced by Rummery and Niranjan [157] in a technical report in 1994. SARSA learning has proven to be a powerful algorithm even when applied to cloud computing, for example, allocating virtual machines service to support an application depending on the level of user demand [181], and SARSA has also been applied to the auto-configuration of virtual machine [153].
The SARSA learning algorithm has the ability to learn without the need for a complete environmental model as it approximates the optimal control policy directly instead of evaluating with a complete MDP model.

SARSA learning provides an improvement upon MC models as the approach learns in an incremental online manner and new information is stored into the action value $Q(s, a)$ function approximations. In addition the episodes do not have to be terminated for the algorithm to update the value-estimates. The SARSA algorithm is known as an online policy as the policy denotes the actions selected by the learning agent in the current state and the information is stored as part of the learning process.

The SARSA value function approximator can be defined as:

$$Q(s, a) < Q(s, a) + \alpha[r + \gamma(Q(s', a') - Q(s, a))] \quad (3.7)$$

The full SARSA learning algorithm is shown in Algorithm 1

---

3.2.4.4 Q learning

Christopher Watkins introduced Q-learning in 1992 [192]. Q—learning can be considered as equivalent to the value iteration method. This Q-learning method represents only the actions chosen each time but does not affect which next action $a'$ contributes to the value function’s estimate. The Q-learning algorithm is often known as an ‘off-policy’ as the learning agent or Q-learner is allowed to directly updates the value function as if it was following the ‘greedy policy’ whilst actually following another. The Q-learning pseudo-code below highlights the major steps involved. Many steps outlined in the SARA algorithm are repeated for the Q-learning algorithm. The one main difference being the update rule
\[ Q(s, a) < -Q(s, a) + \alpha[r(s, a) + \gamma \max Q'(s', a') - Q(s, a)] \] (3.8)

The max action for the next time step \( \max \) represents the highest value of future states from the current state.

Initialise \( Q(s,a) \) arbitrarily

\begin{algorithm}
\begin{algorithmic}
\State \textbf{for} each episode \textbf{do}
\State \quad Initialise s
\State \textbf{for} each step of episode or until s is terminal \textbf{do}
\State \quad Choose a from s using policy derived from Q (e.g., \( \epsilon \)-greedy)
\State \quad Take action a, observe r, s'
\State \quad \( Q(s, a) < -Q(s, a) + \alpha[r(s, a) + \gamma \max Q'(s', a') - Q(s, a)] \)
\State \textbf{end}
\State \textbf{end}
\end{algorithmic}
\end{algorithm}

\textbf{Algorithm 2: Q-learning Algorithm}

3.2.4.5 Agents

An Agent can be defined as an entity that has the ability to perceive its environment often through sensors and reacts to its environment through actuators [159]. Wooldridge and Russell & Norvig (2016) defined several types of agents [200] [159]:

- Simple Reflex Agents: These types of agents react to their environment based on their current perception rather than having an accumulated historic perception.

- Model Based Reflex Agents: This type of agent deploys an internal data structure which is a method for tracking environment states and accumulated historic data.

- Goal-Based Agents: This type of agent makes decision based on its goal information, which lets the agent know what is desirable.

- Utility Based Agents: This type of agent utilise a factor known as performance measure. This measures the mapping of state or sequence of states to a utility with the object being to maximise expected reward.

- Learning Agents: This agent is a more advance agent type. It has of learning capabilities which allow such an agent to operate and learn in unknown environments, building its knowledge up over time.
3.3 Artificial Neural Networks

Artificial Neural Networks (ANNs) were originally developed as a mathematical model or function approximators which are inspired by nature, in particular, the biological brain [24] [84]. Since they were first proposed in the 1960’s, neural networks have been applied to a wide range of problem domains including classification, regression, forecasting, control, learning and robotics. Neural Networks is one of the most prominent fields of Machine Learning research. A basic structure of an ANN is a network of small processing nodes, which are joined to each other by weighted connections. In terms of the biological brain model, the node represents neurons and the weighted connections represent synapses between the neurons, as shown in Figure 3.2 below:

![Neural Network Architecture](image)

Figure 3.2: Neural Network Architecture

The ANN is activated by a signal or series of signals delivered to the input layer. This signal is then spread throughout the ANN along the weighted connections. There are two types of connections within a network, which can be cycles or acyclic. ANNs which have cycle connections are referred to as feedback, recursive, or recurrent, neural networks, whereas acyclic networks are known as ‘Feed Forward Neural Networks’ (FFNN).

The standard feedforward network consists of an input layer of neurons, one or more hidden layer of neurons and an output layer of neurons as shown in Figure 3.2.
A neuron is a signal processing unit that reads in a number of signals as input and then outputs a signal using the activation function.

### 3.3.1 Forward Pass

Consider a neural network with ‘I’ inputs which is activated by an input vector $x$. Each neuron in the first hidden layer calculates a value known as a "weight sum", which is the input times the connection weights of that neuron. For the hidden neuron $h$, we refer to this as the network input and denote as $a_h$. An activation function ($\theta$), is then applied, resulting in the output value of $b_h$. Denoting the weights from neuron $i$ to neuron $j$ as $w_{ij}$, which is represented in equation 3.9 and 3.10:

$$a_h = \sum_{i=1}^{I} w_{ij} x_i \quad (3.9)$$

$$b_h = \theta(a_h) \quad (3.10)$$

Normally neural networks output a value between 0 and 1. This output value is determined by the activation function of the neuron. One of the most popular activation functions to apply in ANNs is the Sigmoid function. This is described by Equation 3.11:

$$\sigma_x = \frac{1}{1 + \exp(-a_{hj})} \quad (3.11)$$

One powerful feature of the Sigmoid activation function is its non-linearity. Non-linear neural networks are far more powerful than linear networks since they can model nonlinear equations. Non-linear networks can gain considerable power by using successive hidden layers to re-represent the input data [87].

Another key function of the Sigmoid activation function is that it is differentiable, which allows the network to be trained with gradient descent. This is described by Equation 3.12.

$$\sigma'(a) = \sigma(a)(1 - \sigma(a)) \quad (3.12)$$

Once calculated the activation for the first hidden layer, the process of summation and activation is repeated for each hidden layer, as shown by Equations 3.13 and 3.14.

$$a_h = \sum_{h'=1}^{H_{n-1}} w_{h'h} b_{h'} \quad (3.13)$$
\[ b_h = \theta(a_h) \]  

(3.14)

Where \( H_l \) is the number of neurons in hidden layer \( l \)

### 3.3.2 Output Layers

The output vector \( y \) of a neural network is given by the activation of the neurons in the output layer. The network input \( a_k \) to each output neuron \( k \) is calculated by the summation over the neurons connections (the same process as for a hidden layer neuron). This is represented in the following equation 3.15:

\[ a_k = \sum_{h=1}^{H_u} w_{hk} b_h \]  

(3.15)

where \( u \) represents hidden layers

### 3.3.3 Back-propagation

ANNs consist of differentiable operators, meaning they can be trained to minimise any differentiable objective function using gradient descent. The idea behind gradient descent is to find the derivative of the objective function with respect to each of the weights of the network and adjust the connection weights in the direction of the negative slope. Gradient descent repeatedly takes a small fixed-sized step in the direction of the negative error gradient (refer to Equation 3.16):

\[ \Delta w(n) = \alpha \frac{\delta \theta}{\delta w(n)} \]  

(3.16)

where \( 0 < \alpha < 1 \) is the learning rate and \( w(n) \) represents the vector of connections weights after the \( n^{th} \) weight update. One problem with gradient descent is that it can easily get stuck in a local minimum (sub optimal solution). However, by applying a momentum parameter to the algorithm through weight space to speed up convergence and helping to escape from the local minimum [24]. This is shown is the following equation.

\[ w(n) = m \Delta w(n - 1) - \alpha \frac{\delta \theta}{\delta w(n)} \]  

(3.17)

where \( m \) denotes momentum. One advantage a basic gradient descent algorithm has on its rivals is that adapts well to online learning, where the weights need to be updated after each sequence of learning. To effectively calculate the gradient or error we use a popular technique known as Back-propagation or back pass of a network.
Back-propagation is the repeated calculation of the chain rule for partial derivatives. The Back-propagation function first calculates the derivatives of the output neurons predicted values, which is represented in the following equations, 3.18, 3.19, 3.20:

\[ \frac{\delta O}{\delta y} = \frac{y - z}{y(1 - z)} \]  (3.18)

Thus the chain rule informs that:

\[ \frac{\delta O}{\delta a} = \frac{\delta O}{\delta y} \frac{\delta y}{\delta a} \]  (3.19)

and we can then substitute 3.18 and 3.19 into 3.20 to get

\[ \frac{\delta O}{\delta y} = y - z \]  (3.20)

For a multi-class network, differentiating 3.21 gives

\[ \frac{\delta O}{\delta y} = \frac{z_k}{y_k} \]  (3.21)

The activation function of each neuron in a layer depends on the network’s input to every neuron in the layer. The chain rule calculation gives us the following equation:

\[ \frac{\delta O}{\delta a_k} = \sum_{k' = 1}^{K} \frac{\delta O}{\delta y_{k'}} \frac{\delta y_{k'}}{\delta a_k} \]  (3.22)

Differentiating 3.22 we get

\[ \frac{\delta y_{k'}}{\delta a_k} = y_k \delta_{kk'} - y_k y_{k'} \]  (3.23)

and then when we substitute 3.21 and 3.22 into 3.23 to give

\[ \frac{\delta O}{\delta a_k} = y_k - z_k \]  (3.24)

We now continue to apply the chain rule, working back through hidden layers. For simplicity, we introduce the following notation.

\[ \delta_j = \frac{\delta O}{\delta a_j} \]  (3.25)

where \( j \) is any neuron in the network. For the neurons in the last hidden layer we have, as shown in Equation 3.26:

\[ \delta_h = \frac{\delta O}{\delta b_h} \frac{\delta b_h}{\delta a_h} = \frac{\delta b_h}{\delta a_h} \sum_{k=1}^{K} \frac{\delta O}{\delta a_k} \frac{\delta a_k}{\delta b_h} \]  (3.26)
where any objective function $O$ depends only on neuron $h$ through its influence on the output neuron. Differentiating 3.10 and 3.15 and substituting 3.26 produces:

$$\delta_h = \theta'(a_j) \sum_{k=1}^{K} \delta_k W_{hk}$$

(3.27)

The $\delta$ for each hidden layer $l$ before the previous layer then can be calculated as shown in Equation 3.28:

$$\delta_h = \theta'(a_h) \sum_{h'=1}^{H_{l+1}} \delta_{h'} W_{hh'}$$

(3.28)

Once the $\delta$ terms for all the hidden layer have been calculated then we can use equation 3.27 to calculate the derivatives with respect to each of the network weights as shown in Equation 3.29:

$$\frac{\delta O}{\delta w_{ij}} = \frac{\delta O}{\delta a_j} \frac{\delta a_j}{\delta w_{ij}} = \delta_j b_i$$

(3.29)

### 3.4 Recurrent Neural Networks

In this section, we consider ANNs whose weight connections are ‘cyclical’. Recurrent Neural Networks (RNN), illustrated in Figure 3.3, are different from the standard feed-forward networks as the hidden layer neurons have recurrent connections. These connections allow the hidden layer neurons to connect to itself. In Figure 3.3 ‘A’ represents a neural network that processes each input $X_i = X_0, X_1, ...$ This individually allows information from each unrolled layer (previous network) to be traversed to the next time step.

Neurons are connected by weighted synapses (or connections) that pass signals between neurons. The recurrent connections can be seen in the hidden layer of neurons. Thus, giving the neural network history of previous inputs to outputs which makes it well suited to the problem of predicting cloud resource. There are other varieties of RNN proposed such as Elman networks [58], Jordan networks [104], time delay neural networks [120], echo state networks [100] and Long-Term-Short-Term-Memory (LSTM) [71]. A LSTM neural network are one of the more popular RNN algorithms to predict sequence of data. In LSTM a mechanism known as gates are utilised to act on signals they receive similar to standard neural networks nodes, they block or pass on information based on its strength and import, which they filter with their own sets of weights. Those weights, like the weights that modulate input and hidden
states, are adjusted via the recurrent networks learning process. That is, the cells learn when to allow data to enter, leave or be deleted through the iterative process of making guesses, back-propagating error, and adjusting weights via gradient descent. Even though LSTM have shown to be very useful in sequence predictions, however in this thesis we have chosen a more light weight approach in back propagation through time to predict sequence data.

![Recurrence Neural Network Architecture](image)

Figure 3.3: Recurrent Neural Network Architecture

Hammer states that an RNN with a sufficient number of hidden neurons has the ability to approximate any measurable sequence-to-sequence mapping to arbitrary accuracy [80]. The main advantage of an RNN is that it stores the previous prediction as memory which is then be used to influence future predictions.

### 3.4.1 Forward Pass

The forward propagation of an RNN is the same as that of a normal FNN, with a single layer, except that activation arrives at the hidden layer from both the current external input and the hidden layer activation from the previous time step. For instance consider an input sequence \( x \) of length \( T \) presented to an RNN with \( I \) input neurons, \( H \) hidden neurons and output neurons represented as \( K \). \( x_t \) is the value of the input \( i \) at time \( t \) and \( a_t \) is the network input to neurons \( j \) at time \( t \) and \( b_t \) is the activation function value of neuron \( j \) at time \( t \). Thus we can define the hidden layer output function as in equation 3.30 as follows:

\[
    a_t^h = \sum_{i=1}^{L} w_{ih} x_i^t + \sum_{h=1}^{H} w_{h', h} b_{h'-1}^t 
\]

(3.30)

Once the hidden layer output is calculated, the differentiable activation functions are applied in the same way as for a normal FFNN. Refer to equation 3.31:

\[
    b_t^h = \theta_h(a_t^h) 
\]

(3.31)
The complete sequence of hidden activation function can be calculated, starting at time \(i = 1\) and then applying equation 3.30 recursively while increasing \(t\) at each time step. The activation function \(b'_t\) must be initialised for the hidden neurons, in most cases. In this Thesis where an RNN is used, the initial \(b'_1\) is set a zero.

The RNN output neuron is calculated as follows:

\[
a^t_k = \sum_{h=1}^{H} w_{hk} b^t_h
\]

(3.32)

### 3.4.2 Back-propagation Through Time

The Back-propagation of the error through the RNN is similar to that of Back-propagation in the standard feed forward neural network (FNN). Two well-known algorithms have been devised to efficiently calculate the connection weights derivatives for the RNN; Back-propagation through time (BPTT) [193] and real-time recurrent learning (RTRL) [195]. In this thesis, we focus on implementing BPTT as it is simpler and more efficient. Similar to the Back-propagation algorithm the BPTT consists of repeatedly applying the chain rule. The difference is that the RNN neurons output depends on the activation function of the hidden layer and through its influence on the output layer and also through the influence of the hidden layer at the next time step. This can be calculated as follows in equation 3.33:

\[
d^t_h = \theta'(a^t_h) \sum_{k=1}^{K} \delta^t_k W_{hk} + \sum_{h' = 1}^{H_{t+1}} \delta^t_{h'} W_{hh'}
\]

(3.33)

Finally, we sum over the whole sequence to get the derivatives with respect to each of the RNN connection weights as shown in 3.34:

\[
\frac{\delta O}{\delta w_{ij}} = \sum_{t=1}^{T} \frac{\delta O}{\delta a^t_j} \frac{\delta a^t_j}{\delta w_{ij}} = \sum_{t=1}^{T} \delta^{t_j} b^{t_i}
\]

(3.34)

### 3.5 Neural Network Optimisation Algorithms

For a neural network to perform optimally a algorithm which trains it weights must be selected. Two traditional methods used to train a neural network are back-propagation and back-propagation-through-time. Three other optimisation algorithms used in this Thesis in chapter 5 are a branch of the swarm and evolutionary optimisation algorithms and are discussed in the following paragraph.
3.5.1 Swarm and Evolutionary Algorithms

Since the proposal of Genetic Algorithms (GA) by John Holland in the 1970’s [89], a large body of research has developed surrounding evolutionary algorithms due to their effectiveness as optimisation problem solvers. The way in which the weights are optimised for neural networks is the main issue with continuous variables. There have been a number of highly effective swarm and evolutionary algorithms for solving continuous optimisation problems notably Particle Swarm Optimisation by James Kennedy in 1995 [109], Differential Evolution in 1997 by Storn and Price [173] and Covariance Matrix Adaptation Evolutionary Strategy by Hansen and Ostermeier in 1996 [82, 83] have been shown to be the most effective optimisation algorithms. The advantage of these methods over traditional mathematical methods is that they are more robust and suitable to non-separable, noisy and non-convex optimisation problems as they do not rely on any problem gradient [209]. They are also capable of approximating very large NP-Hard problems (Non-deterministic in polynomial time), i.e., problems that grow exponentially with the number of variables [45]. As a result, these algorithms are widely used as problem solvers for complex continuous optimisation problems and are used to train the neural network for predicting host CPU utilisation in this research.

This section presents the three state of the art optimisation algorithms, that are used to optimise the weights of the neural networks outlined in the previous section. There are many examples in the literature where Particle Swarm Optimisation, Covariance Matrix Adaptation Evolutionary Strategy and Differential Evolution have been applied to train neural networks [211, 94, 93, 131].

3.5.1.1 Particle Swarm Optimisation

Of the three optimisation algorithms evaluated in this research, Particle Swarm Optimisation (PSO) is the only algorithm that falls under the heading of swarm intelligence. James Kennedy first proposed the PSO algorithm in 1995 [109]. The algorithm consists of a number of particles that move throughout the problem space evaluating potential solutions and move towards the best solutions. The algorithm is initialised by creating N particles with random positions \( \vec{x}_i \) and velocities \( \vec{v}_i \). At each iteration \( t \), each particle moves to a new position (i.e., solution) based on its velocity and previous position.

In contrast, evolutionary algorithms typically use the operator’s crossover and mutation to evaluate new solutions. The fitness of a position is calculated based on
an objective function. This fitness determines which positions are good candidate solutions. Each particle will update its position based on its best personal position $\vec{p}_b$ and that of its neighbourhood $\vec{g}_b$. After a predetermined number of problem evaluations, the particles should converge on the best-found location using the equations of motion outlined in equations 3.35a and 3.35b;

$$v_{t+1} = \chi(v_t + r_1 c_1 (\vec{p}_b - \vec{x}_t) + r_2 c_2 (\vec{g}_b - \vec{x}_t)), \quad (3.35a)$$
$$x_{t+1} = x_t + v_t \quad (3.35b)$$

Where $c_1$ and $c_2 = 2.05$ are acceleration coefficients, $r_1$ and $r_2$ are random numbers between 0 and 1, $\vec{p}_b$ is the best location of the current particle and $\vec{g}_b$ is the best location found within the particle’s neighbourhood. The $\chi$ term is the constriction factor and is defined in equation 3.36a and 3.36b.

$$\chi = \frac{2}{|2 - \varphi - \sqrt{\varphi^2 - 4\varphi}|}, \quad (3.36a)$$
$$\varphi = c_1 + c_2. \quad (3.36b)$$

Here $\chi \approx 0.72984$, and $c_1 = c_2 = 2.05$ [43]. As the particles move around the problem space and evaluate candidate solutions, they should eventually converge on the best position. The constriction factor guarantees convergence. The PSO parameters outlined here are considered to be the standard parameters [30]. Algorithm 3 describes the functionality of the PSO algorithm. In this research $N = 50$ was found to give the best performance.

Create N particles with random position and velocity

\textbf{while} Evaluation e $<$ $E_{max}$ \textbf{do}

\textbf{for} Particle = 1 to $N$ \textbf{do}

Update personal best position $\vec{p}_b$
Update neighbourhood best position $\vec{g}_b$
Evaluate particle’s current position $\vec{x}_t$
Update particle’s velocity $v_t$
Update particle’s position $\vec{x}_t$

\textbf{end}

\textbf{end}

Return best solution

\textbf{Algorithm 3: PSO Algorithm}
3.5.1.2 Covariance Matrix Adaptation Evolutionary Strategy

The Covariance Matrix Adaptation Evolutionary Strategy (CMA-ES) was first proposed in 1996 by Hansen and Ostermeier [82, 83]. Similar to PSO, CMA-ES was proposed as a method to solve global and non-linear optimisation problems. The CMA-ES algorithm falls within a class of algorithms known as Evolutionary Algorithms (EAs) which are inspired by the biological process of evolution and uses evolutionary operators such as crossover, mutation and selection. CMA-ES is initialised by randomly sampling a number of solutions to the optimisation problem. The solutions are then ranked according to their fitness. The mean (m), covariance matrix (C) and step size (σ) are then updated by increasing the likelihood of sampling good solutions (based on the pseudocode in Algorithm 4). The covariance matrix is an \( n \times n \) matrix that determines the variation around the mean, where \( n \) is the number of dimensions in the problem. In Algorithm 4, \( p_\sigma \) is the path for \( \sigma \), \( p_c \) is the path for \( C \), \( \omega \) are the recombination weights such that \( \omega_1 \geq \omega_2 \geq ... \geq \omega_\mu > 0 \) sum to 1, \( y_\omega \) is the move of the population mean, \( \lambda = 10 \) is the population size, \( \mu \) is the number of samples selected for the update, \( I \) is the identity matrix and \( N(m, \sigma^2C) \) is the multivariate normal distribution. Parameter tuning found that \( \sigma_0 = 0.1 \), \( \mu = 0.5 \) and \( \lambda = 5 \) gave the best performance. Standard values for other constants were implemented: \( c_c = 4/n \), \( c_\sigma = 4/n \), \( c_1 = 2/n^2 \), \( c_\mu = \mu \omega / n^2 \), \( d \approx 1 \) and \( \alpha = 1.5 \).

```
Initialize: Sample \( \lambda \) random solutions
Initialize: \( C_0 = I, p_c = 0 \) & \( p_\lambda = 0 \)

while Evaluation \( e < E_{\text{max}} \) do
    \( \bar{x}_i = m + \sigma y_i \sim N(m, \sigma^2C) \)
    \( f_i = \text{fitness}(x_i) \)

end

Sort solutions \( x \) according to fitness
\( m \leftarrow m + \sigma \sum_{i=1}^{\lambda} \omega_{p(i)} y_i =: m + \sigma y_\omega \)
\( p_\sigma \leftarrow (1 - c_\sigma)p_\sigma + \sqrt{1 - (1 - c_\sigma)^2 \sqrt{\mu} \omega C^{-0.5} y_\omega} \)
\( \sigma \leftarrow \sigma \times \exp(\frac{2n}{\alpha} \frac{||p_\sigma||}{\sqrt{||N(0,1)||}} - 1) \)
\( p_c \leftarrow (1 - c_c)p_c [\frac{1}{n \alpha \sqrt{n}} \{||p_\sigma||^2\}^{1/2} \sqrt{1 - (1 - c_\sigma)^2 \sqrt{\mu} \omega y_\omega} \)
\( C \leftarrow (1 - c_1 - c_\mu)C + c_\mu \sum_{i=1}^{\lambda} \omega_{p(i)} y_i y_i^T + c_1 p_c p_c^T \)

end

Return best solution
```

**Algorithm 4:** Covariance Matrix Adaptation - Evolutionary Strategy (CMA-ES) Algorithm
3.5.1.3 Differential Evolution

Similar to CMA-ES, Differential Evolution (DE) is also a global optimisation algorithm that falls under the heading of evolutionary computing. The DE algorithm was first developed in 1997 by Storn and Price [173]. DE does not rely on any gradient information about the problem and is therefore suitable for optimisation problems that contain noise. DE is a relatively simple optimisation algorithm to implement when compared to CMA-ES.

The simplicity and robustness of DE make it a very popular optimisation algorithm and suitable for optimising network weights. Since its creation, DE has been applied to many real-world problems such as robotics [38] and energy systems [85]. The differential evolution optimisation algorithm has been applied to optimisation problems within the domain of cloud computing. Tsai et al. use an improved differential evolution algorithm for multi-objective task scheduling and resource allocation in a cloud environment [185]. Their improved differential evolution algorithm performed better than state of the art multi-objective optimisation algorithms at obtaining Pareto-optimal solutions. There are many examples applications of DE to neural network weight optimisation [94, 1, 54, 143, 133].

At each iteration, the current agent’s position (i.e., solution) is combined with three other distinct agents’ positions to produce a new position $\vec{y}_i$. The functionality of DE is outlined in detail in Algorithm 5. If the new position of an agent has a better fitness than that agent’s previous position, the agent moves to the new position. This is then repeated for each of the agents until a predetermined number of problem evaluations has been conducted. In Algorithm 5 the parameter $CR = 0.9$ is the crossover probability, $F = 0.5$ is the differential weight, and $N = 20$ is the number of agents.
Initialise N agents with random positions

while Evaluation e < Emax do

for Agent = 1 to N do

Select 3 other agents A,B and C

Select random dimension index R

for dimension = 1 to D do

generate random number \( r \in [0,1] \)

if \( r < CR \) Or \( i = R \) then

new position \( y_i = a_i + F \times (b_i - c_i) \)

else

\( y_i = x_i \)

end if

end

if fitness(y) < fitness(x) then

replace \( \vec{x} \) with \( \vec{y} \)

end if

end

Return best solution

Algorithm 5: Differential Evolution (DE) Algorithm

3.6 Comparative Forecasting Methods

In order to measure the performance accuracy of each of the neural network model’s in this thesis, the following forecasting methods implemented are used as a accuracy benchmark:

1. Previous walk (RWF).
2. Moving Average (MA).
3. Linear Regression (LR).
4. ARIMA Model.

3.6.1 Previous Walk

Previous Walk forecasting is the most basic forecasting method that will be implemented. This approach predicts the next utilisation value of CPU \( CPU_{t+1} \) by setting predicted value to the previous value \( CPU_t \), shown in Equation 3.37.

\[
CPU_{t+1} = CPU_t
\] (3.37)
3.6.2 Moving Average

The moving average method is a commonly used forecasting approach. This method consists of predicting a future value by averaging \( n \) previous values. In this thesis the moving average algorithms take the input of current CPU value \( CPU_t \), and the previous CPU value \( CPU_{t-1} \), which are average to predicted the future CPU value \( CPU_{t+1} \) of a host as shown in Equation 3.38:

\[
CPU_{t+1} = \frac{CPU_t + CPU_{t-1}}{n}
\]

(3.38)

where \( n \) is the number of CPU values being evaluated.

3.6.3 Linear regression

Linear regression (LR) is a popular statistical approach to estimate the relationship between one or more input variables and the output variable [140]. The case of one input variable is called simple regression. More than one input variable is called multiple regression. In all cases, regression approximates a function (regression function) that it can be considered as linear or non-linear. If the independent variable is \( x = [x_1, x_2, \ldots, x_m] \), and the corresponding dependent variable is \( y \), then the LR model is as shown in Equation 3.39:

\[
y_t = \beta_0 + \sum_{i=1}^{m} \beta_i x_i
\]

(3.39)

The parameters \( \beta_0 \) and \( \beta_i \) are regression coefficients. A measure of goodness of fit, that is, how well it predicts the output variable \( y \) is the magnitude of the residual \( e_i \) at each of the \( n \) data points as shown in Equation 3.40:

\[
e_i = y - \hat{y}
\]

(3.40)

\( e_i \) is the difference between the prediction output \( \hat{y} \) and the real output \( y \) for data point \( i \).

3.6.4 ARIMA Modelling

ARIMA modelling is one of the most popular and frequently used forecasting approaches in time series analysis [92]. At its simplest, a time series can be described as a collection of observations over successive time intervals from which future values may be predicted. Time series data is often composed of several fundamental
components such as long-term trends, seasonal fluctuations and correlations between sequential observations. An ARIMA model aims to identify and describe the underlying components and systematic variations in the time series data in order to forecast future values. An ARIMA model is defined by three terms denoted as: $p$ is the number of autoregressive terms, $d$ is the number of nonseasonal differences needed for stationarity, and $q$ is the number of lagged forecast errors in the prediction equation. The identification of a valid model is the process of finding suitable values for $p$, $d$ and $q$ which best capture the fundamental patterns in the data.

The integrated component of the model ($d$) is identified before determining the values of $p$ and $q$. One of the fundamental principles in applying an ARIMA methodology is the time series data must be stationary. The concept of stationarity plays a crucial role in the process of fitting an ARIMA model. A non-stationary series is often unstable and can result in false correlations in the series making it extremely difficult to model. In general, a stationary series is one whose statistical properties such as mean and variance remain constant over time. To transform a non-stationary series into a stationary one the series must be differenced, this is achieved by subtracting the value of an earlier observation from the value of a later observation. The number of times the time series data is differenced determines the value of the component ($d$).

The autoregressive (AR) term ($p$) represents the lingering effect of preceding observations on current values in the series. For example, an $AR(1)$ model forecasts future values based on the value of the preceding observation $y_{t-1}$ as denoted below in Equation 3.41. Where $\phi$ is a parameter of the model known as the autoregressive coefficient which represents the magnitude of the relationship and $\varepsilon_t$ represents the random variation at the current time period $t$, as defined in Equation 3.41.

$$y_t = \phi(y_{t-1}) + \varepsilon_t .$$  \hspace{1cm} (3.41)

The moving average (MA) term ($q$) represents the effects of previous random variation on the current period’s random error. For example, an $MA(1)$ model forecasts future values based on a combination of the current random variation and previous error as defined in Equation 3.42 :

$$y_t = \theta(\varepsilon_{t-1}) + \varepsilon_t .$$  \hspace{1cm} (3.42)

Where $\varepsilon_{t-1}$ is the value of the previous random shock, $\theta$ is the correlation coefficient of the model which defines the extent of the relationship and $\varepsilon_t$ represents the random
variation at the current time period \( t \). The combined model assuming differenced data is denoted in Equation 3.43:

\[
y_t = c + \phi_1(y_{t-1}) + \ldots + \phi_p(y_{p-1}) + \theta_1(\varepsilon_{t-1}) + \ldots + \theta_q(\varepsilon_{t-q}) + \varepsilon_t. \tag{3.43}
\]

ARIMA models are also capable of modelling a variety of highly seasonal data. Seasonal ARIMA models are classified by including the following additional seasonal terms \((p, d, q)_m\), where \( m \) defines the number of periods per season. The seasonal portion of the model operates across previous seasonal periods as opposed to previous observations which occur in the standard model introduced above. However, in practice both models are often combined in order to capture all of the fundamental characteristics of a seasonal time series. In this Thesis, the Box-Jenkins methodology was employed in order to fit only the bandwidth model [26]. This methodology consists of several steps: model identification, estimation and diagnostic checking and lastly forecasting and validation.

### 3.7 Summary

This chapter discussed the fundamentals of both RL and the Neural Networks, with both being implemented in subsequent chapters as part of our research. Two of the Reinforcement learning algorithms Q-learning and SARSA algorithm detailed in this chapter will be implemented in experiments in Chapter 4 for the selection VMs from over-utilised hosts. The reason for selecting the RL algorithm for this problem, is that they have shown to be quite useful in decision making domains such network routing [27], grid world [29], robotics [7], allocation of resource [68], VM consolidation [63].

The second part of this chapter detailed the architecture of a Recurrent Neural network and areas it has been applied to. RNN has the ability to store memory of past predictions enabling it to predict sequences of values, making it a promising candidate to predict utilisation levels for a host machine for the next 30 minutes, which are implemented in this Thesis. Back-propagation through time (BPTT) [193] has been chosen to train the recurrent network as tends to be significantly faster for training an RNN than other approaches [169]. This chapter also discussed swarm and evolutionary algorithm which are used in this thesis to train the neural network.

Finally, this chapter details the four traditional comparative forecasting techniques. These methods are utilised as benchmarks to evaluated the performance of each of the neural network optimisation algorithms as they are state of the art.
forecasting algorithms. Each of the forecasting models explained in this chapter are implemented in chapters 5 and 6, except for ARIMA modelling which is only implemented in chapter 6.
Chapter 4

An Autonomous Approach for the selection of Virtual Machines for Live Migration

4.1 Introduction

Analysing and predicting cloud network traffic volumes in real time is becoming more prevalent in efficiently utilising limited resources during peak hours in cloud data centres. Live VM migration is commonly used to transfer VMs across the cloud network, due to its ability to maintain high system performance under dynamic workloads. However, VM migration requires a considerable amount of cloud network resources. Migration policies are often too rigid and do not take into account attributes such as bandwidth. Traditional scheduling policies usually lack essential functionality, such as failing to anticipate the long-term consequences of its scheduling decisions. The goal of live migration is to balance the networks resources along with the network traffic load while satisfying VMs and host resource constraints.

However, once a host transitions into an over-utilised (i.e. depleted resources) or under-utilised (i.e. too much resources available that a host should not be running) state, a single VM or a group of VMs must be relocated to different host machines. These host must have sufficient resources, to ensure Quality of Service (QoS) guarantees, for the end user.

Managing multiple migrations from a host in a given time period where network traffic demand is high, provides a challenging and complex problem, as the cloud computing system may not have sufficient resources freely available to meet these increases in resources required. As a result, processes such as live migration will take longer to complete thus causing saturation to network resources for a sustained period...
of time. In this situation, a host or hosts will spend longer periods transitioning from
over-utilised or under-utilised states and will have a negative impact on service level
agreement (SLA).

4.2 Motivations and Aims

- **H1**: Through the interaction with the environment, a Reinforcement Learning
  agent can learn to select optimal VMs to be migrated between hosts while
  utilising bandwidth availability.

This chapter proposes the development of an autonomous network aware live
migration strategy, that observes the current demand level of a network and performs
appropriate actions to utilise cloud computing resources efficiently at a given time.

A Reinforcement Learning (RL) approach is implemented to control the scheduling
of VMs for live migration. The RL approach learns optimal behaviours even in
partially observable environments. A single learning agent is utilised to determine
the most opportune time to migrate different sized VMs, depending on the current
saturation of network bandwidth links within a data centre. This RL approach has
the ability to learn and adapt to current network congestion and select VMs for
migration that will have the least impact on network resources.

Having an agent learn and adapt to changing workloads will better utilise signifi-
cant resources in depleted network resources at peak times. This chapter presents an
adaptive agent who has the ability to decide which sized VM to migrate depending
on bandwidth available, thus realising bandwidth for other processes.

4.3 Learning Methods

This section describes two reinforcement learning algorithms that has been imple-
mented in this chapter; Q- Learning and SARSA learning.

4.3.1 Q-Learning

In the absence of a complete environment model, model free reinforcement learning
algorithms such as Q-learning can be used to generate good policies in difficult cir-
cumstances [192]. Q-learning belongs to a group of algorithms known as Temporal
Difference (TD) methods. These methods do not require a complete model of the
environment and have the capability of being able to make predictions in an online
fashion.

The update rule for Q-learning is defined as follows:

\[ Q(s, a) \leftarrow Q(s, a) + \alpha [r(s, a) + \gamma \max_{a'} Q(s', a') - Q(s, a)] \]

This value \(Q(s,a)\) is calculated after each time a non-terminal state is reached and
is known as a Q-value. The actions which are chosen are based on a policy \(\pi\) which
is being followed. The action policy employs an \(\epsilon\) policy to decide the optimal action
while in a particular state. This means the action which the agent selects should
present the greatest amount of reward.

Estimated action values of each state action pair \(Q(s,a)\) are stored in a look-
up table known as the Q-matrix. The goal of the learning agent is to maximise
the long-term rewards. By introducing a discount factor \(\gamma\), \((0 < \gamma < 1)\) the agent’s
degree of how it perceives future reward can be controlled. A value close to 1 for \(\gamma\)
assigns a greater weight to future rewards, while a value close to 0 considers only the
most recent rewards. This represents a key benefit of policies determined through
reinforcement learning compared with threshold based policies. The reinforcement
learning based approaches are capable of reasoning over multiple actions, choosing
only those which will yield the greatest cumulative reward over the entire duration
of a learning episode.

4.3.2 SARSA

As referred to in chapter 3, SARSA requires a quintuplet consisting of values \(s_t, a_t, r, s_{t+1}, a_{t+1}\)
in order to calculate its Q-value. This is where the design of the VM selection algo-
rithms differs. Although both algorithms accept the same input, the order in which
they process this input must be altered appropriately. This alteration is evident in
the following SARSA algorithm, post observation of the new state.

\[ Q(s, a) \leftarrow Q(s, a) + \alpha [r + \gamma Q(s', a') - Q(s, a)] \]

4.4 Experimental Set Up

The following section describes the agent’s state-action space, the reward the agent
will receive and the simulator used to conduct the experiments.
4.4.1 State-Action Space & Reward

For an agent to learn, a state space representation must be sufficiently informative to allow optimal actions to be made concerning the environment. However, the agent’s state space also should be sufficiently concise to guarantee the problem remains within the bounds of computational tractability. Commonly referred to as the ‘Curse of Dimensionality’ it refers to the exponential increase in the size of the problem as additional states and actions are considered. Accurately determining an appropriate state space representation for a given domain is a challenging problem which has a direct impact on the performance of the learning system.

In this research a multi-state action space has been implemented. For an agent to interact with the cloud environment in the VM selection domain, there are two major factors which should be considered. Firstly, a agent should know the utilisation level of the host machine and secondly the available bandwidth at the current time step. Based on the current host utilisation and the bandwidth available the agent will select a VM to be migrated (action). Below is a more detailed explanation of the state action space.

The first state \( s \) is defined as the current host utilisation returned as a percentage which therefore confines the first state space in a range of 0-100% in increments of 1% and is obtained through the following Equation,

\[
    s = \frac{\sum_{vnu=1}^{n} vmu}{hu} .100
\]  

(4.1)

where \( hu \) is the total CPU available, \( vmu \) is defined as the VM utilisation, and \( n \) is all possible VMs on a host machine.

The second state space is defined as bandwidth currently available \( bw_t \) returned as a percentage of utilisation, which ranges from 0-100%.

\[
    bw_t = \frac{bw_c}{bw_e} .100
\]  

(4.2)

where \( bw_c \) is the current bandwidth and \( bw_e \) is the maximum amount of bandwidth available.

The action \( a \) space is represented as the VM’s size \( vms \) (RAM). There is three different VM sizes: 512MB, 1024MB and 2048MB. The three sized VMs were chosen based on the constraints applied by the cloud simulator.

The main reason the state space is ranged from 0-100%, as it discretises the environment as Q-learning and SARSA can better operate in discrete state and action
spaces [176]. Both state spaces from host utilisation and bandwidth are in increment of 1%, as previous experiments conducted did not show improvements in performance when the increments were lower or higher than 1%.

The reward signal guides the decision making process. Often represented as single scalar value, it is the benefit of an executed decision. The expected reward which is achievable is dependent on the current action $a$, current state $s$ and next state $s'$. The goal is to maximise overall return in the long run even at the expense of short term gains [176]. The scalar reward received by the agent is defined as follows, in Equation 4.3:

$$vm_{bn} = vms / max_{bw}$$ (4.3)

where $vms$ is the size in-terms of MB (megabytes) of the current select VM and $current_{bw}$ is the current bandwidth available (0-100%). This is defined in Equation 4.4.

$$r_t = current_{bw} - vm_{bn}$$ (4.4)

The following Figure 4.1 illustrates the state action space the agent will perceive at each iteration.

Figure 4.1: Illustration of How an Agent Perceives it’s State Action space.

For simplicity we have implemented a state space for both bandwidth and CPU from 0 to 100% in increments of 1%. However, from preliminary results shown in Table 4.1 increments of 1%, 2% and 3% show similar results when comparing energy usage, however a more extensive experiment would need to be carried out, which is left for future work. A RL agent was trained with Q learning $\epsilon$-greedy algorithm on a single day workload in the cloud simulator cloudsim for the 1%, 2% and 3% increments in the state space and the energy results are shown in Table 4.1.

Table 4.1: Different increments of the State Space

<table>
<thead>
<tr>
<th>State Space Increment</th>
<th>Energy Usage</th>
</tr>
</thead>
<tbody>
<tr>
<td>1% (i.e. 1, 2, 3..)</td>
<td>134.65 kWh</td>
</tr>
<tr>
<td>2% (i.e. 2, 4, 6..)</td>
<td>134.03 kWh</td>
</tr>
<tr>
<td>3% (i.e. 3, 6, 9..)</td>
<td>136.23 kWh</td>
</tr>
</tbody>
</table>
In this thesis a discrete state action space with no function approximator was chosen based on the simplicity of the implementation of the discrete action space. In later chapters in this thesis we explore function approximators (i.e. neural networks) for CPU predictions. In future work function approximators will be considered for estimating state-action spaces.

4.4.2 Simulator Model

The CloudSim toolkit was chosen as an appropriate simulation platform as it allows for the modelling of the virtualized IaaS environment and is the basis of much leading research in the area of cloud computing capabilities, particularly for energy conservation and resource allocation [121] [194] [166] [32].

CloudSim is a framework which allows for the modelling, simulating and experimenting of a cloud computing infrastructure. The CloudSim toolkit is developed in the GRIDS laboratory at the University of Melbourne. CloudSim facilitates switching between space shared and time shared allocation of cores to virtualised services. Cloudsim simulations is a calculation model that generates possible outcomes with given inputs. For this reason, this made it a prime candidate for conducting simulations with as it will produce the same output from a given input, making it possible to measure the RL performance with other standard heuristic algorithms on the same workload.

Another key reason for selecting Cloudsim is the fact that a bandwidth model can be implemented and changed. CloudSim’s network contains a flow model when migrating VMs [69]. They use point to point communication for data from source \( u \) to destination \( k \), entity which is called the flow and is represented as \( f = size_f; u; v_n \) where \( size_f \) is the number of bytes in the flow and \( v_n \) is the VM being migrated. The bandwidth that is available between two entities is represented as \( bw \), \( lat \) is the network latency, the duration of a single network flow can be calculated as follows, in Equation 4.5:

\[
delay = lat + size_f/bw
\]  

(4.5)

The times at which migration occurs, depends on an over-utilisation detection strategy the cloud system is following. Over-utilisation is determined by 5-minute intervals (industry standard). Five minutes intervals are chosen due to the fact that if a shorter period is set (for example 2 minutes) a greater number of migration would be
initiated increasing the energy consumption significantly. Cloudsim’s Local Regression (LoR) approach which is used to estimate the CPU utilisation is implemented to predict when a host will become over-utilised.

Cloudsim has several built-in algorithms (Minimum Migration, Random Selection, Maximum correlation) which we utilise in this chapter. Utilising Cloudsim’s performance metrics we compare Cloudsim’s algorithms with the most promising of the RL algorithm (Q-learning or SARSA). These metrics will show the effectiveness of each algorithm in terms of Energy, Service Level Agreement Violations SLAVs, total live migration time and ESV (Energy * SLAV). Finally this section will detail a bandwidth data model used in this cloud environment.

Figure 4.2 illustrates how the agent will interact with the cloud simulated environment. The agent will first perceive its state space, which is made up of current bandwidth available between hosts and the CPU of the current host selected. From this structure, the agent will learn to select a VM based on the reward it is receiving from the cloud environment.

4.4.3 Algorithms compared

Beloglazov et al. research in 2011 remains one of the most highly cited and accepted pieces of research in relation to the consolidation of VMs while maximising performance and efficiency in cloud data centres [15]. Beloglazov models SLAs as a key
component for performance of a cloud data centre and is also a key metric for our work. Building on past research Beloglazov suggests an adaptive selection policy known as Local Regression (LoR) for determining when VMs require migration from host in order not to violate SLAs [13]. Local Regression first proposed by Cleveland allows for the analysis of a local subset of data in this case, hosts [44]. The Local Regression algorithm decides if a host is likely to become over utilised if their current CPU utilisation is larger than the maximum possible utilisation. Beloglazov implements four different VM selection policies: Minimum Migration Time, Maximum Correlation and The Random Selection Policy.

4.4.3.1 Minimum Migration Time

Minimum Migration Time (Mmt) policy first decides which VMs are placed on a migration list based the lowest utilised RAM, which is divided by spare bandwidth for the host \( h \), the policy chooses the appropriate \( v \) through the following Equation (Equation 4.6),

\[
x \in V_h \mid \forall y \in V_h, \quad \frac{\text{RAM}(x)}{\text{NET}_h} \leq \frac{\text{RAM}(y)}{\text{NET}_h}
\]

(4.6)

where \( x \) and \( y \) are two comparing VMs based on RAM, and \( \text{NET}_h \) is the spare network bandwidth available on host \( h \).

4.4.3.2 Maximum Correlation

Maximum correlation (Mc) policy is based on the premise that the stronger the inter-relationship of applications running on an over utilised server, the higher the probability the server will overload as highlighted by Verma et al. [188]. The multiple correlation policy finds a VM \( v \) that satisfies the following policy.

\[
v \in V_h \mid \forall a \in V_h, \quad R^2 x_v, x_1, x_\ldots, x_{v+1}, x_{v-1}, x_n \geq R^2 x_v, x_1, x_\ldots, x_{v+1}, x_{v-1}, x_n
\]

(4.7)

4.4.3.3 The Random Selection Policy

The Random Selection Policy (Rs) is another simple method to select VMs from overloading hosts. The policy chooses a VM randomly to migrate. The policy is repeated until the host is no longer considered as being overloaded [186].

This chapter focuses on selecting a VM to be migrated, so the LoR algorithm is implemented to detect over-utilisation.
4.4.4 Performance Metrics

The following section discusses metrics to measure Clousim’s performance as outlined by Beloglazov et. al. [18].

4.4.4.1 Energy Consumption

The total energy consumed by the data centre per day in relation to computational resources (i.e. host). Although other energy draws exist, such as cooling and infrastructural demands, this area was deemed outside the scope of this research.

4.4.4.2 Average VM Migration Time

The average VM migration time (AVMT) is calculated by accumulating migration times for each VM migrated, as defined in Equation 4.8 and 4.9.

\[ VMT = \sum_{v=1}^{n} vm_{migtime} \]  
\[ (4.8) \]

where \( vm_{migtime} \) is determined for each VM based on the delay calculation in equation 4.5.

\[ AVMT = \sum \frac{VMT}{n_{vm}} \]  
\[ (4.9) \]

where \( n_{vm} \) is the number of VMs to be migrated during a time step.

4.4.4.3 Service Level Agreement Metrics

The importance of maintaining a high standard of QOS and SLAs is imperative for a cloud provider to uphold. Their importance is highlighted by the three stages of measurement used to accurately report SLA violations.

4.4.4.4 SLATAH, PDM & SLAV

Service level agreement time per active host (SLATAH) is calculated as the time \( Ts_i \) where active hosts have experienced 100% utilisation of their CPU. As a result, this restricts access to VMs upon the host \( i \) to any further processing energy should they request additional CPU utilisation, thus forcing violations. Where \( N \) represents the number of hosts and \( Ta_i \) represents the time host \( i \) is actively serving VMs. Performance degradation due to migration, as defined in Equation 4.10.

\[ SLATAH = \frac{1}{N} \sum_{i=1}^{N} \frac{Ts_i}{Ta_i} \]  
\[ (4.10) \]
PDM calculates the overall reduction in performance experienced by VMs. \( M \) is the number of VMs, \( C_{sv} \) is the estimated performance degradation of VM \( v \) with \( C_{av} \) representing the total requested CPU capacity by a VM over its lifespan, as defined in Equation 4.11.

\[
PDM = \frac{1}{M} \sum_{v=1}^{m} \frac{C_{sv}}{C_{av}}
\]  
(4.11)

Due to the equal importance of both SLATAH and PDM, a combined metric, service level agreement violation (SLAV) is used to measure the combination of both metrics as defined in Equation 4.12.

\[
SLAV = SLATAH \cdot PDM
\]  
(4.12)

### 4.4.4.5 Energy and SLA Violations

In order to ensure that the implementation of an energy saving policies does not negatively impact SLAs, one must measure the co-related effect [18]. This is done by combining the Energy and SLA Violations to create the ESV metric and is calculated in Equation 4.13.

\[
ESV = ENERGY \cdot SLAV
\]  
(4.13)

The lower the ESV, the better the performance of the data centre.

### 4.5 Data Models

The following section describes two data models (bandwidth and CPU) which are used in the cloud simulator.

#### 4.5.1 CPU Model

The CPU workload comes from a real world IaaS environment. PlanetLab files within the CloudSim framework contains data from CoMon project which represents CPU utilisation values of over a 100 VMs from servers located in 500 locations worldwide. In order to produce an accurate and reliable experiment the algorithms were deployed to represent a one month time period. In order to achieve this, the PlanetLab files
were utilised through random selection to create a 30 day workload. Each PlanetLab file contains 288 values representative of CPU workloads. VMs are assigned these workloads on a random basis in order to best represent the stochastic characteristics of workload allocation and demands within an IaaS environment. Each VM corresponds to that of the Amazon EC2, other than that they are single core, representing the fact the workload was retrieved from single Core VMs. The 288 CPU values when used with CloudSims default monitoring interval represents 24 hours of data centre capacity.

4.5.2 Bandwidth Model
The bandwidth model implemented as part of this study is based on TCP bandwidth measurements collected from Amazons EC2 cloud [162]. In particular, the bandwidth values were taken from measurements of the network performance within Amazons EU region. This benchmark study provides a measurement of the available bandwidth on the network links at 4 points over a single day. In order to generate a bandwidth model with a sampling distribution of 10-minute intervals, the values were interpolated resulting in a time series model composed of 10-minute intervals over a single day (24 hours).

In this study, the initial bandwidth values over 24 hours were sampled at each interval and the corresponding values were inputted into a Gaussian distribution in order to produce a bandwidth model, shown in Figure 4.3. The Gaussian distribution served to introduce uncertainty into the bandwidth values on the network links, thus resulting in a more random and scientifically valid model. The resulting model consisted of 300 values for each day (represented as time on the x-axis in Figure 4.3). Using the same procedure as above, a test set was also generated from the initial distribution and used to validate the selected models.

4.6 Results
The following section details the results of each experiment conducted. First, a comparison analysis of two of the training algorithms for RL (Q-learning and SARSA) is conducted to determine the policy which will best suit the selection of VMs. The second half of this chapter conducted experiments which compared the best RL approach with each of the Cloudsim’s migration algorithms.
4.6.1 Q-Learning vs SARSA Results

Two RL optimisation functions known as softmax and $\epsilon$-greedy will be examined. Both of these functions will determine the most optimal action to take (which VM to select) in a given state (current bandwidth availability and host utilisation). Whether softmax or $\epsilon$-greedy is better depends on the task or the environment in which they are deployed, and an intrinsic link between the action selection choice and the update function [176].

For this reason, the following experiment was conducted to determine the optimal combination for the update function and selection policy. There are four possible combinations;

1. Q-Learning / $\epsilon$-greedy
2. Q-Learning / Softmax
3. SARSA / $\epsilon$-greedy
4. SARSA / Softmax

The following experiment in this section decides which combination we will use in later experiments. Both algorithms will be analysed over a single workload for a hundred iterations. The reason for examining the policies over a single workload for
multiple iterations, is that it will show how quickly an agent can learn and how good their policies are.

4.6.2 Algorithm Learning Performance

The convergence of each of the RL VM selection policies can be seen in Figure 4.4. This figure shows the speed of convergence of each policy over the 100 iterations on a single day workload. One hundred iterations was chosen to give the agent enough time to discover a good policy for the problem it is solving. The graph also shows which policies achieved the lowest average migration time.

It is evident from the graph the two worst performing algorithms are the Q-learning softmax and the SARSA Softmax. Q-learning - $\epsilon$-greedy converges to a policy that yields the highest return from all of the algorithms. However, the SARSA $\epsilon$-greedy performance is very similar.

There is an interesting difference in the rate of convergence to an optimal policy for the algorithms trained by $\epsilon$-greedy than when trained by softmax. Figure 4.4 shows after 70 iterations both Q-learning -$\epsilon$-greedy and SARSA $\epsilon$-greedy have achieve low migration times, which neither of the softmax algorithms could achieve.

![Figure 4.4: Training Convergence of the RL Agent.](image)

This graph shows the convergence rate of each of the RL approach over 100 iterations on the same workload.
4.6.3 Average Time for Migration

The first simulation metric which we will examine is the average time for VM migration. The RL agent must select a VM that will ensure fast migration when different levels of network traffic are occurring in a data centre. Thus, balancing the size of a VM to be migrated with the bandwidth available is essential to ensure quick migration. In this regard, it is essential to examine the average time for live migration for all approaches.

Table 4.2 presents the average times in terms of seconds for live migration.

Table 4.2: Average Migration Times for Each of the RL Policies

<table>
<thead>
<tr>
<th>Policy</th>
<th>Average Time (s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Q-learning-(\epsilon)-greedy</td>
<td>0.656</td>
</tr>
<tr>
<td>Q-learning-Softmax</td>
<td>0.701</td>
</tr>
<tr>
<td>SARSA-(\epsilon)-greedy</td>
<td>0.669</td>
</tr>
<tr>
<td>SARSA-Softmax</td>
<td>0.711</td>
</tr>
</tbody>
</table>

Table 4.2 shows the Q-learning - \(\epsilon\)-greedy algorithm has the lowest migration time. SARSA \(\epsilon\)-greedy outperforms both algorithms trained by softmax. Q-learning - \(\epsilon\)-greedy achieved an average migration time of 0.656 seconds with SARSA softmax having the worst performance of 0.711 seconds. One reason for Q-learning-\(\epsilon\)-greedy achieving a low migration time, is that the epsilon function allows for a degree of randomness to occur, allowing for the agent to discover more of its state space, whereas SARSA takes a safer approach when deciding which VMs to migrate. A t-test, Q-learning - \(\epsilon\)-greedy shows to have a statistical improvement in to the next best performing algorithm SARSA \(\epsilon\)-greedy, with a P-value of \(<0.234\) with a 95% confidence interval (1.6965, -2.346).

4.6.4 Average Energy Consumption

The rapid growth in cloud data centres has led to a surge in demand for computational power which in turn has led to enormous consumption of electrical power. Modern data centres must balance both performance and energy consumption.

Table 4.3 shows the energy consumption of all algorithms in the simulation. As shown from the results, the Q-learning - \(\epsilon\)-greedy has the lowest energy consumption of all approaches with an average of 156.57 kWh. SARSA-Softmax had the worst energy consumption of all approaches with the average energy consumption of 164.53 kWh.
Table 4.3: Energy Consumption for Each of the RL Policies

<table>
<thead>
<tr>
<th>Policy</th>
<th>Energy (kWh)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Q-learning-(\epsilon)-greedy</td>
<td>156.57</td>
</tr>
<tr>
<td>Q-learning-Softmax</td>
<td>163.89</td>
</tr>
<tr>
<td>SARSA-(\epsilon)-greedy</td>
<td>160.54</td>
</tr>
<tr>
<td>SARSA-Softmax</td>
<td>164.53</td>
</tr>
</tbody>
</table>

SARSA-Softmax high migration count case it to have the worst energy consumption. The initiation of live migration itself consumes energy thus, increasing the energy consumption of a host machine. A t-test, Q-learning - \(\epsilon\)-greedy shows to have a statistical improvement in to the next best performing algorithm SARSA \(\epsilon\)-greedy, with a P-value of <0.0034 with a 95% confidence interval (-5.9434, -30.94).

4.6.5 Average Service Level Agreement Violation

The last metric that was examined was the Service Level Agreement Violation (SLAV). In this thesis SLAV is the percentage of time, during which active PMs have experienced the CPU utilisation of 100% Table 4.4 presents the average SLAV’s each algorithm occurred during the simulations. The results again show that Q-learning - \(\epsilon\)-greedy has the best performance for SLAV. Q-learning-\(\epsilon\)-greedy achieved such a low level of SLAV as it has the lowest average live migration Time, leaving hosts in an over-utilised state for short periods of time. A t-test, Q-learning - \(\epsilon\)-greedy shows to have a statistical improvement in to the next best performing algorithm SARSA \(\epsilon\)-greedy, with P-value of <0.253 with a 95% confidence interval (2.045, -3.944).

Table 4.4: SLAV for Each of the RL Policies

<table>
<thead>
<tr>
<th>Policy</th>
<th>SLAV (Time %)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Q-learning-(\epsilon)-greedy</td>
<td>0.00505</td>
</tr>
<tr>
<td>Q-learning-Softmax</td>
<td>0.00665</td>
</tr>
<tr>
<td>SARSA-(\epsilon)-greedy</td>
<td>0.00565</td>
</tr>
<tr>
<td>SARSA-Softmax</td>
<td>0.00685</td>
</tr>
</tbody>
</table>

The Q-Learning-\(\epsilon\)-greedy based model consistently out performed other selection policies. Based on these results, the Q-Learning-\(\epsilon\)-greedy will be used in the following section, comparing it’s performance to other heuristic algorithms.
4.7 Cloud Simulation Performance Results

This section will present the performance of each of the heuristics (detailed in section 4.4.3) compared to the RL approach in a simulated data centre environment known as Cloudsim. Local regression is used to detect over-utilisation and the LoR acronym will be used before each approach, e.g. LoR-RL, LoR-Mmt.

4.7.1 Simulation Energy Consumption

Figure 4.5 shows the energy consumption of each approach. It is clear the RL agent consumes the least amount of energy, with Lor-Mmt consuming the highest amount. The Lor-RL agent selects a VM to migrate based on the available BW and size of the VM (RAM). The Lor-Mmt algorithm selects the VM with least utilisation of RAM. However, this in turn results in a host being in an over-utilised state for longer periods and more migration occurring. A paired t-test shows that there is a statistically significant difference in the consumption of energy when utilising Lor-RL and Lor-Mmt resulting in a P-value < 0.0041 with a 95% confidence interval (-7.8685, -39.8715). As a result over the 30 day period, the Lor-RL algorithm consumes 523.22 kWh less energy overall.

![Energy consumed each day for a 30 day workload](image)

Figure 4.5: Simulation Energy Consumption.
This graph illustrates the energy consumption of each of the approaches over a 30 day workload.
4.7.2 Simulation SLAV

Lowering energy usage within a data centre can have a parallel negative effect on SLAV. For instance, one can lower the number of active servers through the extreme consolidation of VMs to a fewer number of servers. However, this results in a greater possibility of hosts reaching 100% utilisation of their CPU and VMs access to computational processing is restricted, resulting in violations.

The SLA violations are displayed in Figure 4.6. Lor-Mmt achieves the best SLA, with Lor-RL having a similar performance. The results of a T-test shows there is no statistical difference, with a P-value of $<0.2751$ with a 95% confidence interval $(1.1365, -3.9669)$. Lor-MC achieves the worst performance out of all approaches.

![Figure 4.6: Simulation Service Level Agreement Violations.](image)

This graph shows the SLAVs for each of the approaches over a 30-day workload.

4.7.3 Simulation Average Migration Time

Table 4.5 presents the overall average migration times for each approach. Both the Lor-Mmt and Lor-RL approaches produce very similar results. The Lor-RL approach achieves a low migration time based on migrating small size VMs when little BW is available and migrating bigger size VMs when more BW is available. On the other hand, Lor-Mmt always migrates the VM with the least RAM usage thus ensuring low migration time no matter the bandwidth utilisation. The results of a t-test show
there is no statistical difference with a P-value of <0.1751 with a 95% confidence interval (0.1365, -3.454).

Table 4.5: Average Migration Times

<table>
<thead>
<tr>
<th>Policy</th>
<th>Average Time (ms)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Lor-RL</td>
<td>0.056</td>
</tr>
<tr>
<td>Lor-MMT</td>
<td>0.046</td>
</tr>
<tr>
<td>Lor-RS</td>
<td>0.196</td>
</tr>
<tr>
<td>Lor-MC</td>
<td>0.256</td>
</tr>
</tbody>
</table>

4.7.4 Simulation ESV

The final metric to be evaluated is the ESV, results of which can be seen in Figure 4.7, the results overall the Lor-RL approach achieves the best performance, with Lor-RS second. On carrying out a t-test, Lor-RL shows to have a statistical improvement in overall performance, with a P-value of <0.0001 with a 95% confidence interval (-0.0021, -0.0037).

![Graph showing ESV performance for each day of a 30-day workload](image)

Figure 4.7: Simulation ESV Performance

This graph presents the ESV performance for each of the approaches over a 30-day workload.
### Table 4.6: Summary of all Performance Metrics for Each Algorithm

<table>
<thead>
<tr>
<th></th>
<th>AVT (ms)</th>
<th>Energy</th>
<th>SLAV(Time %)</th>
<th>ESV</th>
</tr>
</thead>
<tbody>
<tr>
<td>Lor-RL</td>
<td>0.056</td>
<td>156.57</td>
<td>0.00505</td>
<td>0.722</td>
</tr>
<tr>
<td>Lor-Mc</td>
<td>0.046</td>
<td>161.80</td>
<td>0.00507</td>
<td>0.822</td>
</tr>
<tr>
<td>Lor-Rs</td>
<td>0.196</td>
<td>161.39</td>
<td>0.00506</td>
<td>0.815</td>
</tr>
<tr>
<td>Lor-Mmt</td>
<td>0.256</td>
<td>173.45</td>
<td>0.00503</td>
<td>0.871</td>
</tr>
</tbody>
</table>

### 4.8 Discussion

The results from the experiment show that the reinforcement learning approach has the capabilities to adapt to changing network loads and decide the most suitable VM to migrate based on bandwidth conditions. The time an RL agent schedules a migration of particular sized VM can have a detrimental impact on SLAs.

In the first experiment, both the Q-learning and SARSA were both trained with $\epsilon$-greedy and softmax policies. The results showed that the Q-learning - $\epsilon$-greedy algorithm outperformed all other Reinforcement learning approaches on each of the metrics, which is in line with Sutton & Barto’s cliff walking example [176]. Their results show SARSA as accumulating greater reward similar to the cliff walking. This is the result of SARSA’s online policy nature, which considers the action selection policy, therefore not letting the agent fall off the cliff or in the context of this chapter move an unrewarding machine.

In contrast Q-learning chooses to ignore the action selection policy and attempts to converge to the optimum policy, even though on occasion, this can cause an agent to fall off the cliff, or migrate an inappropriate VM, but will explore more of the state action space, and converging on an optimal policy faster. The Q-learning - $\epsilon$-greedy algorithm converges onto an optimal faster as it explores its state-space and migrates a VM based on the bandwidth.

The second set of experiments involved determining which algorithm would produce the most efficient data centre, regarding energy, SLAVs and average migration times.

From the results, the RL approach achieved the lowest ESV value. The Lor-RL algorithm achieved the lowest energy consumption and the second lowest SLAV. The Lor-RL approach determines which VMs to be migrated based on the current bandwidth available, the host utilisation and the size of the VM (RAM usage). When the bandwidth is low, and the host utilisation is high, the RL agent learns to select the best VM candidate (small sized VM) to ensure SLAs are not violated. However,
when there is no saturation in the network load, and the host is of high utilisation, the agent learned to migrate larger VMs, which were migrated quickly because of the amount of bandwidth available while also providing fast relief to over-utilised hosts.

4.8.1 Limitations

The RL agent approach showed how an adaptive virtual machine selection strategy has the ability to improve a simulated cloud data centre performance. However, there are limitations when applying the reinforcement learning algorithm in this problem. For example, consider the time take for the RL agent to learn an optimal or sub-optimal policy for the current problem. It takes up to 40 episodes for the agent to start to converge upon a policy.

Also one has to consider the state space used in this research, which is discrete. In real world problem state spaces are continuous, which has infinite number of states and actions. Standard RL algorithms do not perform well in continuous state spaces. For example, for the given problem the bandwidth is set at a specific range, however, if these values were to drastically change due to the installation of a better quality fiber optic connection the agent would not generalise well to new unseen data, causing performance degradation of host machines.

4.8.2 Comparison with the State of the Art

A learning algorithm known as Reinforcement Learning was implemented as a decision support system to select the correct VM to migration at critical times in a data centre. The agent’s goal was to move a VM quickly and efficiently while also maintaining a high level performance for the data centre. In comparison Beloglazov et al. proposed a method that selects VMs based on the lowest RAM usage, comparing this with our approach we consider RAM, bandwidth and host utilisation when deciding on a VM to migrate [17].

Sohrabi et al. proposed two algorithms, the first being MedianMT, which selects a given VM that requires the average time to complete a migration relative to other VMs [170]. Their second approach known as Maximum Utilisation selects a VM based on the largest CPU usage on a host machine. Our approach builds upon their work by firstly only selecting large CPU usage VMs when there is enough bandwidth available to migrate the VM across the network quickly. Secondly, our approach selects VMs based on size relative to the network bandwidth available.
The difference between our approaches and previous works such as Beloglazov et al. [15] is that they focus on determining policies for live migration of VMs by selecting a VM which utilise the least amount of RAM. Our approach considers both the size of RAM of a VM and the current network traffic. Our approach also facilitates learning and reasoning of performance variance associated with the cloud to best match the bandwidth levels and the current VM to be migrated.

4.9 Summary

The contribution of this chapter can be summarised as follows:

- Q-Learning-\(\epsilon\)-greedy showed to outperform other variants of the Q-learning and SARA algorithm to produce a more efficient data centre over a single day workload.

- The findings from this research show that a RL VM selection policy provides a more robust and energy efficient approach while also showing significant improvements in the quality of service provided.

- The RL algorithm out-performs heuristic-based approaches proposed by [15] to produce a more efficient simulated data centre over a 30-day workload.
Chapter 5

Predicting Host CPU utilisation in Cloud Computing Using Recurrent Neural Networks

5.1 Introduction

Predicting resource utilisation has been listed as one of the ten biggest obstacles facing the growth of cloud computing [6]. One of the major difficulties for prediction algorithms in cloud computing is that cloud resources are continually changing and exhibit complex dynamic behaviour. A practical resource prediction solution must be able to produce accurate predictions to enable cloud management systems to provide the resources required to VMs before they suffer performance degradation. The solution itself must be able to predict resource usage of a host for short time scales such as 5-minute intervals to longer time scale such as 30 minutes, to best provision resources such as CPU, RAM or disk, required by VMs. However, many prediction approaches are not capable of functioning when unexpected spikes occur in resource usage.

This chapter proposes the power of a machine learning algorithm known as a neural network to predict host CPU utilisation. The neural network will be trained with 5 different training algorithms: Backpropagation, Backpropagation through time, PS0, CMA-ES, and DE, each of which are discussed in greater detail in chapter 3.

The following chapter utilises the CRISP-DM (cross-industry process for data mining model [197] to provide a structure for the planning and experimentation of CPU forecasting. The CRIPS-DM model has 6 different phases: business understanding, data understanding, data, preparation, modelling, evaluation and deployment. The following sections in this chapter discuss each of these topics in detail how they will
help to develop a prediction model which can be deployed in the cloud environment (Refer to figure 5.1).

Figure 5.1: CRISP-DM [197]

5.2 Business Understanding

The first stage in the CRIPS-DM model is to understand what you want to achieve, we do this by stating the motivations and aims.

Motivations and Aims

- H2: Employing a predictive algorithm trained with traditional Back-propagation algorithms and evolutionary methods which are able learn from historical data to improve upon traditional non-linear and linear algorithm’s prediction accuracy of future host resource utilisation.

In recent years, machine learning algorithms have received a great amount of attention and are a very active area of research due to the increased power of modern computers. Cloud computing has been adapting the power of machine learning to improve performances of data centres. In this chapter, we focus on applying a Recurrent Neural Network to predict CPU utilisation, as it is one of the most critical metrics for measuring the performance of host machines and is a popular metric for researchers in predicting future host performance [214, 52, 22].
Many studies have examined one-step ahead forecasting methods such as locally Weighted Scatterplot Smoothing and Feed-forward Neural Network (FFNN) for predicting CPU utilisation at the next time step [214] [52] [22]. However, this one step ahead prediction time frame (usually 5 minutes into the future) leaves little time for the cloud resources to be adjusted.

In this thesis we define the short time scale as 5 minutes to 1 hours. Predicting resources in these short time scales. Research has shown resource prediction can be difficult on shorter time scales as they can be extremely random [19]. The greater the accuracy of predictions of an algorithm on resource demand for short-term time scale is critical to a data centre’s efficiency and performance.

Therefore, it would be advantageous for the cloud management system to have information of when the host machine will become over-utilised in advance to enable to migrate VMs to others host machine or to start up another host to help manage current levels of traffic.

The main aim of this chapter is to employ a recurrent neural network trained with both traditional algorithm and evolutionary methods to predict CPU utilisation of a host machine.

![Figure 5.2: Histogram of the Planet Lab CPU Training Data-set.](image)

### 5.3 Data Understanding & Data Preparation

The second and third phase of the CRISP-DM model is to acquire the data needed and process the data. The data used in this chapters experiments is from the PlanetLab
files [34]. Each of the PlanetLab files contain CPU utilisation values measured every 5 minutes in PlanetLab’s VMs. Each file contains 288 values, and each value relates to five minutes in a 24 hour day. There are 10 folders, each folder contains between 898 to 1516 files.

Three data sets were generated from the PlanetLab files. The first data set generated was the training data which contained 2296 values or 8 days’ worth of CPU data. The second data set contains 288 CPU values (1-day workload) and was used to test the Neural Networks after the training phase had been completed. The training and the testing data sets are from the same host. However, the third data set which was used to also test the Neural Networks has come from different host CPU values. The reasoning behind having two test data sets from two different hosts was to test the generality and adaptability of the Neural Networks on entirely new host CPU data.

Figure 5.2 shows the distribution of the Training data, with 10 bins. The distribution of the training CPU data is skewed to the right, favouring the higher CPU utilisation values. Interestingly the lower values from 0-10 occur frequently. Having a data-set which is not of a normal distribution could potentially affect accurate forecasting of the CPU data.

![PlanetLab Testing CPU Dataset](image_url)

**Figure 5.3: Histogram of the Planet Lab CPU Testing Data-set.**

Figure 5.3 shows the distribution of the Testing data, with 10 bins. The distribution of the testing CPU data is skewed to the right, favouring the higher CPU utilisation values similar to the testing data. However, the lower values from 0-10 occur the most frequently in the testing data set. Both the training and testing data have similar distributed data, favouring more frequent higher CPU values. In theory, having
similar distributed data sets should aid each forecasting model, as sufficient examples should be available from the training data to help improve the testing data accuracy. Table 5.1 presents the summary metrics for both the training and testing data sets, with both data set values being similar.

Table 5.1: Summary Metrics of the Training and Testing CPU Data-sets.

<table>
<thead>
<tr>
<th>Data set</th>
<th>Min</th>
<th>1st Qu.</th>
<th>Median</th>
<th>Mean</th>
<th>3rd Qu.</th>
<th>Max</th>
</tr>
</thead>
<tbody>
<tr>
<td>Training</td>
<td>0.00</td>
<td>35.15</td>
<td>71.71</td>
<td>59.69</td>
<td>84.68</td>
<td>99.99</td>
</tr>
<tr>
<td>Testing</td>
<td>0.00</td>
<td>17.39</td>
<td>62.02</td>
<td>52.72</td>
<td>80.41</td>
<td>99.91</td>
</tr>
</tbody>
</table>

5.4 Modelling

The fourth phase for the CRIP-DM model is to identify and select a machine learning model. We chose a Neural Network model which is one of the most prominent fields of Machine Learning research. Neural Networks are function approximators that are inspired by nature, in particular, the biological brain [24, 84].

Since they were first proposed in the 1960’s, neural networks have been applied to a wide range of problem domains including classification, regression, forecasting, control, learning and robotics. The standard feed forward network consists of an input layer of neurons, one or more hidden layer of neurons and an output layer of neurons. A neuron is a signal processing unit that reads in a number of signals as input and then outputs a signal using the sigmoid (or logistic) function. The network senses its environment by reading a normalized signal into the input layer. The neural network implemented in this chapter takes a signal input which corresponds to the CPU demand on the host from previous time steps ($t - 1$). This signal is propagated forward through the connected layers of neurons via weighted connections. Once the signal is propagated through the network, the network outputs the signal through the output layer of neurons. Regarding the host utilisation prediction problem, this output signal corresponds to the CPU demand on the host in future time steps.

This chapter implements a particular type of neural network known as a Recurrent Neural Network (RNN), illustrated in Figure 5.4. Recurrent networks differ from the standard feed forward networks due to the recurrent connections between hidden neurons. These recurrent connections differ from the other network connections as they connect neurons within the same hidden layer. This gives the neural network
‘memory’ which makes it particularly well suited to the problem of predicting CPU demand.

![Fully Connected Recurrent Neural Network](image)

Figure 5.4: Fully Connected Recurrent Neural Network [129].

This figure illustrates a Recurrent Neural Network. Neurons are connected to the weighted synapses that pass signals between neurons. The recurrent synapses can be seen in the hidden layer of neurons. This gives the recurrent network the abilities to retain information.

### 5.5 Experimental Setup

The following section details how the network parameters were selected to determine the most accurate architecture to predict CPU and Bandwidth utilisation and determine the length of the training phase. This section also describes the data models for CPU used to train and test both each algorithm used in this chapter. Finally, in this section, a description of the forecasting algorithms used to compare the performance of each trained neural networks is discussed.

#### 5.5.1 Network Parameter Selection

A recurrent neural network (RNN) is implemented in this chapter to predict CPU utilisation. The reason for an RNN being selected over a feed forward neural network is that the recurrent connections give the system memory which is advantageous for the task of CPU utilisation prediction. The network has three hidden neurons and two inputs for the current and previous CPU utilisation. Parameter sweeps revealed that selecting any more than three hidden neurons did not lead to an increase in performance and resulted in a longer training time due to the additional number of weights to be trained. Parameter sweeps also showed that more than two inputs
did not lead to any increase in performance. The network has one output which corresponds to the network’s prediction of future CPU utilisation.

5.5.2 Network Training

The weights of the network correspond to a potential solution for each of the optimisation algorithms outlined in the previous section. A potential solution is referred to a particle’s position in PSO, a candidate solution in CMA-ES and an agent’s position in the DE algorithm. Each algorithm evaluates 10,000 network weight configurations before the network with the highest prediction accuracy is returned and evaluated on the test data. This process is repeated over 10 runs to ensure statistically significant results.

5.5.3 Metrics

The following describes the metrics used to compare the accuracy of each algorithm mentioned in this chapter.

Mean Absolute Error (MAE) measures the difference between the predicted value and the actual value by the mean of the absolute error. MAE tells us how big of an error we can expect from the forecast on average, as shown in Equation 5.1.

\[
MAE = \frac{1}{n} \sum_{t=1}^{n} |y - \hat{y}|
\]  

(5.1)

Mean Squared Error (MSE) is the measure of how close a fitted line is to data points. Each of the data points takes the distance vertically from the point to the corresponding y value on the curve fit (the error), and square the value. Then take the sum of all these values for all data points and divide by the number values n. The squaring of each error is done to prevent negative values. The smaller the MSE, the closer the fit will be to the data (see Equation 5.2).

\[
MSE = \frac{\sum_{t=1}^{n} (y - \hat{y})^2}{n}
\]  

(5.2)

5.5.4 Experiments Conducted

The following experiments were conducted as part of our research:
1. Compared Performance: The first experiment involves comparing the performance of each optimisation on the training data. All evolutionary neural networks are judged based on their rate of convergence, prediction accuracy, performance consistency and ease of implementation.

2. Performance of Each Trained Network: The second experiment evaluates the performance of each trained network on previously unseen data. The purpose of this is to test if the trained networks are capable of giving a good general performance and do not only perform well on the training data.

3. Future Prediction: The third experiment evaluates how far into the future the network can predict. It would be advantageous in data centres to know well in advance how much CPU will be required of a given host. This experiment will evaluate the accuracy of the network for predicting CPU utilisation further than one step into the future.

4. Unseen Host: Finally, the fourth experiment examines the generality of the trained networks. Once the network has been trained on a host’s CPU utilisation data set, the network is then tested on new data from a completely separate host. Testing the performance of the network on a completely new host gives insight into how general the networks are at predicting CPU levels and also establish if there are commonalities in the CPU utilisation data from different hosts.

5.6 Results

The fifth phase of the CRISP-MP is to evaluate the performance of the model. We do this by examining the results of the experiment we have conducted.

This section presents the results of each of the experiments outlined above followed by a discussion. A two tailed t-test with a significance level of $\alpha = 0.05$ is conducted to determine significant performance differences when comparing algorithms. All values are rounded to 4 decimal places.

5.6.1 Training Data

The convergence of each of the training algorithms for the recurrent networks evaluated can be seen in Figure 5.5. This figure illustrates the best solution for each algorithm as they evaluate the training data over 10,000 evaluations. It is shown in
the graph that CMA-ES performs best and PSO performs the worst of the 4 algorithms. However, BPTT and DE provide a very similar level of performance regarding both rates of convergence and final fitness.

![Convergence Rate of Neural Networks](image)

**Figure 5.5: Convergence Rate of Neural Networks.**

This graph presents the convergence of each neural network training algorithm.

In Figure 5.6 (training time steps 0-500) there is an interesting difference in the prediction values for the network trained with PSO than those trained with CMA-ES or DE. The network trained using PSO performs significantly worse at predicting the lower CPU values than either BPTT, CMA-ES or DE, as seen in Figure 5.6. When predicting the higher CPU values, neither CMA-ES nor DE trained networks predict CPU utilisation values higher than approximately 0.8. It is thought that the reason for this is that when the CPU utilisation is high, the CPU values oscillate rapidly between 0.6 and 1.0. Each of the networks trained appears to struggle with this rapid change in CPU utilisation and as a result, found that predicting approximately 0.8 when the CPU utilisation is high is the best overall policy. By time-step 1000-1500 phase, shown in Figure 5.8 all algorithm have converged on to a solution and also highlights when the CPU utilisation changes dramatically each algorithm perform poorly with predictions. It appears that the PSO trained network in parts does match these changing high CPU values. Table 5.2. However, shows that the PSO trained network performs worse overall.

Table 5.2 displays the Mean Absolute Error (MAE) and Mean Squared Error (MSE) for each of the algorithms tested. It is evident from Table 5.2, the Previous
Figure 5.6: Host CPU Training Prediction (Time Steps 1 - 500).
This graph shows the host utilisation predictions for first 500 steps of training data.

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>MAE (Std Dev)</th>
<th>MSE (Std Dev)</th>
<th>RMSE (Std Dev)</th>
<th>MAPE (Std Dev)</th>
</tr>
</thead>
<tbody>
<tr>
<td>PSO</td>
<td>0.1521 (0.0021)</td>
<td>0.0481 (0.0003)</td>
<td>0.2193 (0.0003)</td>
<td>0.194 (0.004)</td>
</tr>
<tr>
<td>DE</td>
<td>0.1422 (0.0008)</td>
<td>0.0454 (0.0004)</td>
<td>0.2130 (0.0004)</td>
<td>0.1842 (0.002)</td>
</tr>
<tr>
<td>CMA-ES</td>
<td>0.1414 (0.0008)</td>
<td>0.0452 (0.0005)</td>
<td>0.2126 (0.0005)</td>
<td>0.1833 (0.005)</td>
</tr>
<tr>
<td>BPTT</td>
<td>0.1492 (0.0011)</td>
<td>0.0454 (0.0004)</td>
<td>0.2130 (0.0004)</td>
<td>0.1911 (0.008)</td>
</tr>
<tr>
<td>Previous Walk</td>
<td>0.1746 (0.0000)</td>
<td>0.0594 (0.0000)</td>
<td>0.2437 (0.0000)</td>
<td>0.2165 (0.000)</td>
</tr>
<tr>
<td>Moving Avg</td>
<td>0.1715 (0.0000)</td>
<td>0.0572 (0.0000)</td>
<td>0.2391 (0.0000)</td>
<td>0.2134 (0.0000)</td>
</tr>
<tr>
<td>Linear Regression</td>
<td>0.1643 (0.0000)</td>
<td>0.0498 (0.0000)</td>
<td>0.2231 (0.0000)</td>
<td>0.206 (0.000)</td>
</tr>
<tr>
<td>Back-propagation</td>
<td>0.1505 (0.0010)</td>
<td>0.0462 (0.0004)</td>
<td>0.2149 (0.0004)</td>
<td>0.1928 (0.004)</td>
</tr>
</tbody>
</table>

Walk, moving average, linear regression and Back-propagation neural network perform significantly worse than all recurrent neural network algorithms. Previous Walk performed the worst overall followed by moving average, linear regression and then Back-propagation. This further validates the choice of recurrent networks for CPU utilisation time series prediction. When comparing the performance of each of the neural networks, statistical testing reveals that CMA-ES performs significantly better than DE, BPTT and PSO. PSO performs statistically worse than BPTT, CMA-ES and DE. CMA-ES provides the best solution and converges fastest on the training data. CMA-ES and DE also have the lowest standard deviation meaning that they give the most consistent prediction accuracy. The box plots in Figure 5.9 illustrate the spread in the MAE produced by each algorithm. From the result, it is clear that the RNNs here perform significantly better than moving average, previous Walk, linear
regression and back-propagation in this scenario.

5.6.2 Test Data

The next set of experiments involve evaluating the performance of the networks trained with each algorithm on previously unseen test data. The goal of this experiment is to establish how each network performs on new data from the same host as it was trained on. Figure 5.10 compares the predictions of each network with the actual CPU values. Similar observations can be made from this graph as can be made from the training data in Figure 5.6, i.e. BPTT, CMA-ES and DE are better at predicting lower CPU values. The ability of each network to perform well on the unseen test data confirms the generality of the trained neural networks.

The accuracy of each the algorithm for the test data can be seen in Table 5.3. As with the training data, the non recurrent neural network algorithms perform statistically worse. Back-propagation performed the worst overall followed by moving average, previous Walk and finally linear regression. Interestingly Back-propagation performs the worst on the test data but performs third worst on the training data in the previous section. This indicates that the network trained using Back-propagation does not generalise well to unseen data. Compare the Back-propagation results to the BPTT algorithm where it had the lowest MSE value and performs the third best
for MAE. Of the evolutionary neural networks, the network trained using PSO also performs worst of the three optimisation algorithms. A more interesting observation, however, is the performance of DE and CMA-ES. When evaluated on the training data CMA-ES performs statistically better than DE, however when evaluated on the test data DE performs statistically equal to CMA-ES. The reason CMA-ES performs significantly better than DE on the training data but equal on the test data is that CMA-ES over-trains the network on the training data to the point where gains in accuracy on the training data does not correspond to gains in accuracy in the test data. This is an interesting result because CMA-ES appears to be the superior optimisation algorithm for training the neural network, this well trained neural network does not perform any better on new data. In this case, it may be more advantageous to implement DE to train the network as it is a simpler algorithm to implement and perform equally well on test data.

The final observation to be made from Table 5.3 is that all of the standard deviations for each of the networks are higher than they were for the training data. This is to be expected however due to the fact that the networks are being evaluated on unseen data, more deviation in the accuracy is to be expected.
Figure 5.9: Forecasting Accuracy Spread.

Table 5.3: Test Data Accuracy

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>MAE (Std Dev)</th>
<th>MSE (Std Dev)</th>
<th>RMSE (Std Dev)</th>
<th>MAPE (Std Dev)</th>
</tr>
</thead>
<tbody>
<tr>
<td>PSO</td>
<td>0.1564 (0.0023)</td>
<td>0.0483 (0.0002)</td>
<td>0.2197726098 (0.002)</td>
<td>0.1983 (0.002)</td>
</tr>
<tr>
<td>DE</td>
<td>0.1495 (0.0018)</td>
<td>0.0465 (0.0013)</td>
<td>0.2156385865 (0.015)</td>
<td>0.1883 (0.008)</td>
</tr>
<tr>
<td>CMA-ES</td>
<td>0.1498 (0.0013)</td>
<td>0.0468 (0.0008)</td>
<td>0.2163330765 (0.007)</td>
<td>0.1917 (0.0010)</td>
</tr>
<tr>
<td>BPTT</td>
<td>0.1598 (0.0016)</td>
<td>0.0466 (0.0008)</td>
<td>0.2158703314 (0.008)</td>
<td>0.2017 (0.0016)</td>
</tr>
<tr>
<td>Previous Walk</td>
<td>0.1733 (0.0000)</td>
<td>0.0602 (0.0000)</td>
<td>0.2489979992 (0.000)</td>
<td>0.2175 (0.000)</td>
</tr>
<tr>
<td>Moving Avg</td>
<td>0.1756 (0.0000)</td>
<td>0.0577 (0.0000)</td>
<td>0.2402082434 (0.000)</td>
<td>0.2159 (0.000)</td>
</tr>
<tr>
<td>Linear Regression</td>
<td>0.1704 (0.0000)</td>
<td>0.0509 (0.0000)</td>
<td>0.2256102835 (0.000)</td>
<td>0.2123 (0.000)</td>
</tr>
<tr>
<td>Back-propagation</td>
<td>0.1892 (0.0123)</td>
<td>0.0558 (0.0050)</td>
<td>0.2362202562 (0.005)</td>
<td>0.2311 (0.02)</td>
</tr>
</tbody>
</table>

5.6.3 Multi-step Ahead Prediction

This experiment aims to evaluate how far into the future the neural network can predict CPU utilisation and to establish how much the accuracy of the prediction decreases. Since CMA-ES trained neural network demonstrated the best accuracy in the two previous results sections, the CMA-ES neural network was implemented to predict CPU utilisation for multiple steps into the future in this experiment. This experiment involved predicting the CPU utilisation at 1, 2, 3 and 4 steps into the future where each step corresponds to 5 minutes. Table 5.4 presents the accuracy of the prediction at each of the future points in time. As is expected, the accuracy of the future predictions decreases steadily the further into the future the network attempts to predict. This is true for both the training and testing data.
Figure 5.10: Host Utilisation Predictions for Test Data.

Table 5.4: Multi-Step Prediction Accuracy

<table>
<thead>
<tr>
<th>Number of Steps</th>
<th>Training MAE (Std Dev)</th>
<th>Training MSE (Std Dev)</th>
<th>Training RMSE (Std Dev)</th>
<th>Testing MAPE (Std Dev)</th>
<th>Testing MAE (Std Dev)</th>
<th>Testing MSE (Std Dev)</th>
<th>Testing RMSE (Std Dev)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 Step Ahead</td>
<td>0.1414 (0.0008)</td>
<td>0.0452 (0.0005)</td>
<td>0.2126029163 (0.0005)</td>
<td>0.3031501278 (0.0005)</td>
<td>0.1498 (0.0013)</td>
<td>0.0468 (0.0008)</td>
<td>0.2289 (0.0029)</td>
</tr>
<tr>
<td>2 Step Ahead</td>
<td>0.1793 (0.0013)</td>
<td>0.0909 (0.0009)</td>
<td>0.2505642731 (0.0009)</td>
<td>0.3384 (0.0009)</td>
<td>0.187 (0.0020)</td>
<td>0.0927 (0.0013)</td>
<td>0.2510 (0.0029)</td>
</tr>
<tr>
<td>3 Step Ahead</td>
<td>0.2033 (0.0011)</td>
<td>0.1016 (0.0008)</td>
<td>0.2890731273 (0.0008)</td>
<td>0.3583 (0.0009)</td>
<td>0.207 (0.0020)</td>
<td>0.107 (0.0011)</td>
<td>0.2772 (0.0029)</td>
</tr>
<tr>
<td>4 Step Ahead</td>
<td>0.2206 (0.0011)</td>
<td>0.1107 (0.0008)</td>
<td>0.3096547591 (0.0009)</td>
<td>0.3894 (0.0009)</td>
<td>0.259 (0.0020)</td>
<td>0.1209 (0.0011)</td>
<td>0.3015 (0.0029)</td>
</tr>
</tbody>
</table>

Figure 5.11 displays two different graphs. The top graph shows the actual CPU demand for the host. The bottom graph displays the absolute error of the prediction at each time step for both 1 and 4 steps ahead on the test data. The purpose of this graph is to illustrate at what points in the CPU utilisation data does the accuracy of the network increase and decrease. Figure 5.11 reveals that the largest prediction errors occur when there are large and instantaneous changes in the CPU utilisation of the host. An example of this is just before the 100th time step. At this point on the top graph, it can be seen that the utilisation drops from 1 to 0 very suddenly. At the corresponding point on the bottom graph, it can be seen that there is a very high spike in the error of both the 1 and 4 step ahead predictions. This is because the network finds it very difficult to cope with these instantaneous and drastic changes in CPU utilisation. At other time steps where the change is less severe or less instantaneous, the corresponding prediction error is much lower. The final observation to note from Figure 5.11 is the difference between the prediction error of the 1 and 4 step ahead predictions. Whenever there is a large error spike such as the spike midway between the 200th and 250th-time step, the 1 step ahead error spike is much shorter in duration.
than the 4 step ahead error. The reason for this is merely that for the 1 step ahead prediction, the large error is discovered at the next time step and the network can adjust its prediction to accommodate for the sudden change in CPU utilisation. For the 4 step ahead prediction, however, the sudden change in CPU utilisation is not discovered for another 4 time steps, and therefore the prediction error stays larger for a longer duration. This is to be expected however given the nature of the CPU utilisation data.

Figure 5.11: Prediction Residuals on Test Data.

5.6.4 New Host Evaluation

The final experiment involved testing the performance of a network on separate host data than that which it was initially trained on. This experiment aimed to test if the
trained network is general enough to accurately predict new and unseen CPU values from a completely different host.

The accuracy obtained testing the trained network on CPU utilisation data from a new host was: $\text{MAE} = 0.1442 \pm 0.0009, \text{MSE} = 0.0475 \pm 0.0007$. When these results are compared to those in Table 5.3, it is evident that the network trained on one host is general enough to be applied to other hosts and still achieve a similar level of accuracy. This result implies that there are similarities between different hosts in a data centre. The CPU demand profile for the various hosts has similar characteristics that the network can exploit these when making predictions on a separate host than the host on which it was trained. This is beneficial from a practical point of view. It would be advantageous to not to train a neural network for each host in the data centre to make accurate predictions. The results presented in this section indicate that training individual neural networks for each host may not be necessary.

### 5.7 Discussion

The results for each experiment demonstrate that recurrent neural networks have the capabilities to improve upon traditional prediction methods to predict CPU utilisation with a high degree of accuracy. This is true for both one-step and multi-step prediction.

The first experiment demonstrated how an RNN trained with various algorithms can outperform traditional forecasting methods such as previous Walk, moving average, linear regression and Back-propagation neural networks. The results show that despite a significant amount of noise present in the CPU utilisation data, the BPTT, CMA-ES, PSO and DE trained neural networks could produce more accurate forecasts when compared to the traditional models. The CMA-ES had the best accuracy when evaluated on the training data.

The second experiment established the generality of each method by applying the forecasting models to previously unseen data. In this experiment, the evolutionary methods again provided the best forecasting accuracy. The best performing algorithms here were both DE and CMA-ES which performed equally optimally.

The third experiment conducted establishes how much the prediction accuracy of the recurrent network decreases as they attempt to predict further into the future. The results demonstrate that it is indeed possible to predict multiple time steps into the future while still retaining a reasonable degree of accuracy. The CMA-ES trained neural network obtained a reasonably high level of accuracy when predicting 4 time
steps into the future ($4^5 = 20$ minutes). As expected, the further into the future the neural network tries to predict, results in a significant decrease in prediction accuracy.

Multi-time step ahead forecasting has proven to be a difficult area in time series research. The additional noise present in the CPU utilisation data makes forecasting far into the future even more difficult. The recurrent neural network forecasting method presented in this chapter could potentially be incorporated with many of the other subfields of cloud computing that involve host migration and VM scheduling to improve overall performance.

The results from the final experiment highlighted the generality of the proposed evolved neural networks. When evaluated on previously unseen data from a completely new host, the evolved neural network performed with an accuracy comparable to the original host. This is advantageous as it implies that it may not be necessary to evolve a neural network for every host in the data centre to make accurate forecasts. Evolving just a single neural network can provide accurate forecasts for future CPU utilisation for each host.

The primary aim of this research was to investigate if neural networks are capable of accurately predicting CPU utilisation through the CRISP-DM model. The results obtained indicate that it is, in fact, possible to predict CPU utilisation with a high degree of accuracy for short time periods and on data sets that have sudden extreme changes. All four algorithms (BPTT, PSO, DE and CMA-ES) were able to train a network to accurately predict CPU utilisation within 10,000 evaluations of the training data. Of the four algorithms evaluated CMA-ES performed the best on the training data followed by DE then BPTT and lastly PSO. On the test data, however, CMA-ES and DE performed equally well measure with the MAE metric with BPTT outperforming the other four algorithms for the MSE metric.

5.8 Limitations

Even though results from this chapter show the recurrent neural network can improve resource prediction of a host machine, it’s limitation must too be considered.

For example, in this research, a univariate forecasting approach was considered, only predicting CPU utilisation of hosts. However, having more inputs for the neural network, such as considering RAM and disk space along with CPU utilisation would in theory, improve the prediction accuracy of the networks. However, for this
research, a univariate approach was sufficient to highlight the recurrent neural networks capability.

This research only two CPU workloads were considered and implemented. The first workload was split into training and testing data sets and was from the same host. The second data set was from a different host to examine the networks ability to forecast CPU on an unseen workload. Even though these workloads were extremely fluctuating, more workloads could be considered with different characteristics such as random CPU usage, sinusoidal patterns and low CPU usage, to examine thoroughly the abilities of the RNN to adapt to changing workloads.

5.9 Comparison with the State of the Art

The results demonstrated in this chapter contributed to improving the state of the art methods. The RNN approach predicts CPU for both one time step to multi-time steps into the future. Research such as Farahnakian et al. implements a one step ahead linear regression model to detect host over-utilisation based on a predicted CPU value and a defined threshold [62]. The RNN approach furthers this research by comparing a linear regression model with more advance RNN models to achieve higher prediction accuracy results. Our model also predicts CPU usage multiple time steps into the future.

Khatua et al. propose an approach predicting resource multiple times steps into the future, utilising an Auto-regressive Integrated Moving Average (ARIMA) model [112]. An ARIMA model is useful at predicting sequences of values (such as 24 hour CPU load). Our approach was able to achieve reasonable accuracy for both one-step ahead and multi-step ahead CPU predictions.

Duy et al. used a neural network predictor for the optimisation of servers power consumption in a data centre [57]. Similar to the approach we undertook, the Duy et al. trained their neural network on historical data to predict future demands. However, they implemented a standard feed-forward network. Our work implemented a RNN architecture to enable the network to store memory to improve CPU predictions.

Janardhanan et al. implement a long-short-term-memory (LSTM) neural network to predict day worth CPU of host machines [102]. The approach proposed in this chapter utilised both traditional neural networks training algorithms such as Back-propagation and evolutionary algorithms such as PSO, CMA-ES and DE to adequately train a neural network to outperform traditional approaches. The RNN
approaches also produced accurate results for both for sequences prediction of CPU and with one step ahead predictions.

5.10 Summary

The contribution of this chapter can be summarised as follows:

- Neural Networks are capable of accurately predicting CPU utilisation for short time scales and multi-time steps when compared to traditional linear and non-linear models.

- All of the Neural Network training algorithms (BPTT, PSO, DE and CMA-ES) were able to train a network to accurately predict CPU utilisation within 10,000 evaluations of the training data.

- The CMA-ES provides the best approach for predicting CPU when compared to other evolutionary technique, traditional neural network algorithms and traditional linear prediction models.
Chapter 6

A Multi-Time Steps Ahead Prediction Approach for Scheduling Live Migration in Cloud Data Centres

6.1 Introduction

Scheduling the precise time for live migration to occur is a challenging problem for resource planning in a data centre. The algorithm must first have the ability to anticipate when live migration will be required as a host will become over-utilised and secondly determine how congested network traffic is when live migration is scheduled to occur. As server consolidation continues to attract attention from an energy perspective, more research will be dedicated to the cause and effects of long migration times, due to the direct impact on performance. Recently, many studies have been looking closely into how network traffic contributes to long delays when migrations of VMs occur.

6.2 Business Understanding

Motivations and Aims

- H3: Implementing a recurrent predictive algorithm which stores previous predictions as memory, will aid in developing an efficient scheduling live migration to best utilise resources at critical times better when compared to heuristic approaches.
This chapter focuses on applying forecasting algorithms to optimise times of migration, decrease service level agreement violations (SLAVs) and reduce energy consumption. A recurrent neural network (RNN) is implemented to predict CPU and network bandwidth and compare results to traditional one step ahead nonlinear and linear forecasting algorithms. The reasoning for implementing the RNN is that the RNN has the ability to retain information and accurately make predictions for time series problems, making it a promising candidate to predict cloud resources for given time periods with greater accuracy when compared to traditional approaches.

It is estimated by 2020, there will be 51,974 GB of internet traffic generated per second [97] and with the trend of technology companies offering computational resources as a service via cloud computing will generate massive volumes of data from hosts and Virtual Machines (VMs). The more information that is available from these hosts and VMs will provide a better understanding of how they function and will allow a more in-depth monitoring of cloud resources. CPU is one of the most critical metrics for measuring the performance of a host machine and is a popular metric for researchers to test when predicting host performance [214, 52, 22]. These studies are examples of one-step-ahead forecasting methods such as LOESS and feed-forward Neural Networks used to predict CPU utilisation. However, one-step-ahead prediction models (usually predicting on a timescale no longer than 5 minutes ahead) give insufficient time for the cloud resources to adjust, to sudden high demands when they occur. For example, if an algorithm can predict further into the future, there will be a greater chance of preventing a host becoming over-utilised. However, previous studies have shown predicting a workload on a time scale such as 5-10 minute intervals is more difficult than predicting for long-term timescales (i.e. timescales of days or weeks) [19], due to the fact that cloud resources in these short timescales can be extremely unpredictable. In this thesis we refer to the time span of 5 minutes to 1 hour for short time scales.

The further into the future an algorithm can accurately predict resources usage is critical to how well a data centre can perform and is one of the key ideas that has motivated this research. Along with CPU resource prediction, the analysing and predicting of cloud network traffic volumes is becoming more prevalent in efficiently utilising limited resources during peak hours in cloud data centres.
6.3 Data Understanding and Data Preparation

Two data models (bandwidth and CPU) were used in this study as data for the algorithm to predict and also as an input to our simulator.

*Bandwidth Data.* The bandwidth model implemented as part of this study is based on TCP bandwidth measurements collected from Amazon's EC2 cloud [162]. In particular, the bandwidth values were taken from measurements of the network performance within Amazon's EU region. This benchmark study provides a measurement of the available bandwidth on the network links at four points over a single day. In order to generate a bandwidth model with a sampling distribution of 10-minute intervals, the values were interpolated resulting in a time series model, composed of 10 minute intervals over a 24 hour period. In general, the more data that is available to fit a predictive model provides a greater opportunity to generate better predictions. Two datasets were generated from the TCP bandwidth measurements [162]. The first was the training set, in order to generate the training data the initial bandwidth values over 24 hours were sampled at each interval, and the corresponding values were inputted into a Gaussian distribution in order to produce a bandwidth model over seven consecutive days as shown in Figure 6.1. The Gaussian distribution served to introduce uncertainty into the bandwidth values on the network links. The resulting model consisted of 144 values for each day (6 data points per hour x 24 hours) or 1008 values in total. Using the same procedure as above, a test set was also generated from the initial distribution and used to validate the selected models. The test set contained 432 values (3 days workload) and was used to test the accuracy of the predictive models in this study. In the forecasting literature, there is generally no principled approach to dividing data into training and test sets. Rob Hyndman (2014) [92] suggested an 80/20 split between the training and test sets, where roughly 80% of the data is used to train the model, and the remaining 20% is used to test the model. Broadly speaking a commonly occurring ratio within the community is to use 60-80% of the data as the training set and 20-40% as the test set. Based on this we divided the bandwidth model using 70% of the data as the training data which corresponded to the first seven days’ worth of time series data. The remaining 30% of the data was used as a test set which corresponded to the next three days. By dividing the data using these ratios it helps to ensure that the predictive model can generalise to unseen data.
**CPU Data.** The CPU data model implemented as part of this study is Google’s cluster data trace [154]. This trace data is the resource usage information of machines in a cluster for a 29-day trace period. This data set is over 300GB in size and contains information of over 12,000 host machines in Google’s data centres. All of the algorithms mentioned in Section 2.3 are trained on a CPU data set that contains 7623 values and tested on a data set that contains 144 CPU values. The testing data contains 144 values representing each 10-minute intervals in 24 hours. The training data is 53 days in total. The reason for having such a large training set is that there is a huge amount of variation in the Google cluster data trace and this will give each algorithm an opportunity to discover patterns within the data. Figure 6.2, shows the CPU training data used.

![Bandwidth Model for Intra Cloud Network Links Within the EU Region](image)

**Figure 6.1: Simulator Bandwidth Model**
Figure 6.2: CPU Training Data Set

Figure 6.3 presents the distribution of the bandwidth data, with a maximum of 10 bins, utilised in this chapter. The graph shows the data set with a focus mainly around the medium bandwidth values. Interestingly, the bandwidth values range from 500 MB to 700 MB.

Figure 6.3: Histogram of the Bandwidth Data-set.

Figure 6.4 shows the distribution for the Google cluster trace CPU data, with maximum of 10 bins. The graph shows the distribution to be somewhat normal, with
nearly all CPU values falling between 50% to 75% CPU utilisation. Interestingly, from the statistical summary table 6.1 the lowest CPU value is 31.91% compared to that of the lowest value in chapter 5 CPU data of 0. From the distribution in Figure 6.4 and summary statistics in table 6.1 compared to the planet lab data used in chapter 5, the Google trace CPU data seems to be less volatile and the forecasting approaches should achieve a reasonable accuracy when predicting the future CPU.

![Google Cluster Trace CPU Dataset](image)

Figure 6.4: Histogram of the CPU dataset.

<table>
<thead>
<tr>
<th>Dataset</th>
<th>Min</th>
<th>1st Qu.</th>
<th>Median</th>
<th>Mean</th>
<th>3rd Qu.</th>
<th>Max</th>
</tr>
</thead>
<tbody>
<tr>
<td>Bandwidth</td>
<td>492.7</td>
<td>562.4</td>
<td>620.7</td>
<td>612.2</td>
<td>651.1</td>
<td>704.8</td>
</tr>
<tr>
<td>CPU</td>
<td>31.91</td>
<td>56.23</td>
<td>62.56</td>
<td>62.35</td>
<td>65.95</td>
<td>96.32</td>
</tr>
</tbody>
</table>

### 6.4 Modelling

*Neural Networks* are function approximators that are inspired by the biological neural networks that constitute the human brain [24]. Some of the applications of neural networks include: power generation [129] [130] where Neural Networks were trained which could accurately estimate the required power and watershed management [134] where a Neural Network was used to improve the water distribution.
The standard feed-forward network consists of an input layer of neurons, one or multiple hidden layers of neurons and an output layer. The neural networks receive information in the form of a signal (normalised between 0 and 1) through the input layer neurons and then outputs a signal using the Sigmoid function. The signal or input that the network receives is in the form of two CPU utilisation values from a host machine or two bandwidth utilisation values. Both sets of data are normalised between 0 and 1.

The two input values (CPU or Bandwidth) are propagated forward through the hidden layers of neurons via synapses (weighted connections). Then the network calculates an output at the output layer neuron or neurons. The Neural network models implemented in this chapter have multiple outputs, and the output signals correspond to future CPU or bandwidth values of a host machine. An error signal is calculated by finding the difference between the actual and the predicted value. This error is then propagated back through the network, and the weights (synapses) are adjusted to correct the error of the prediction.

This research implements a Recurrent Neural Network, illustrated in Figure 5.4 (Section 5.3). Recurrent networks are different from the standard feed-forward networks as the hidden layer neurons have recurrent connections. These connections allow the hidden layer neurons to connect to itself. Thus, this architecture allows the neural network to store memory of previous predictions and makes it well suited to the problem of predicting CPU utilisation or bandwidth demand.

The recurrent network is trained using the popular Back-propagation-Through-Time (BPTT) algorithm [193]. A more significant discussion of the state of the art of recurrent neural networks is discussed in Section 5.3.1.

6.5 Experiment Details

The following section describes metrics that were used to determine the most accurate algorithms for predicting CPU and Bandwidth utilisation. This section also describes the data models for CPU and Bandwidth used to train and test both non-linear and linear models in this chapter. Similar to Chapters 6 experiments we train test and compare traditional linear and non linear models, however each algorithm will be trained on the Google cluster training data set along with the Amazon EC2 bandwidth data set. Even though the CMA-ES algorithm outperformed all other algorithms in Chapter 6, we have chosen to implement the BPTT algorithm as it results in less overhead when configuring and training the networks.
Finally, in this section a detailed description of the simulated cloud environment implemented to test each algorithm performance, regarding service level violations, bandwidth usage and energy consumption is given.

6.5.1 Metrics

The following describes the metrics used to compare the accuracy of each algorithm mentioned in section 2.3. In the following descriptions of the metrics $y$ denotes actual, $\hat{y}$ denotes forecast and $n$ is the number of values.

Mean Absolute Error (MAE) measures the difference between the predicted value and the actual value by the mean of the absolute error. MAE tells us how big of an error we can expect from the forecast on average, as shown in Equation 6.1.

$$MAE = \frac{1}{n} \sum_{t=1}^{n} |y - \hat{y}|$$  \hspace{1cm} (6.1)

Mean Absolute Percentage Error (MAPE) calculates the average % the forecasted values deviate from the actual values observed in the test set, as shown in Equation 6.2.

$$MAPE = \frac{100}{n} \sum_{t=1}^{n} \left| \frac{y - \hat{y}}{y} \right|$$  \hspace{1cm} (6.2)

Both the MAPE and MAE methods are based on the mean error and are likely to underestimate the impact of large infrequent errors (incorrect predictions by the algorithms). For this reason both the MSE and RMSE are also used in this chapter to measure prediction accuracy.

Mean Squared Error (MSE) is the measure of how close a fitted line is to data points. For each of the data points you take the distance vertically from the point to the corresponding $y$ value on the curve fit (the error), and square the value. Then you sum all these values for all data points and divide by the number values $n$. The squaring of each error is done to prevent negative values. The smaller the MSE, the closer the fit will be to the data (see Equation 6.3).

$$MSE = \frac{\sum_{t=1}^{n} (y - \hat{y})^2}{n}$$  \hspace{1cm} (6.3)

Root Mean Squared Error (RMSE) is the square root of the mean square error. By squaring the errors before calculating the mean and then taking the square root
of the mean, we arrive at a measure of the size of the error that gives more weight to
the large infrequent errors (see Equation 6.4).

\[
RMSE = \sqrt{\frac{\sum_{t=1}^{n} (y_t - \hat{y}_t)^2}{n}}
\]  

(6.4)

6.5.2 Simulator Model

The simulation in this chapter is a large-scale data centre consisting of a cluster of
600 host machines represented as \( H(h_1, h_2, ..., h_n) \). Each host \( h_n \) contains a list of
VMs \( V(v_1, v_2, ..., v_n) \) and has a capacity of \( a_h \). Each VM is of size 1024MB. Each VM
is allocated \( a_v \) of CPU. Therefore the maximum number of VMs allocated to a host
is represented as \( m = a_h / a_v \). In the interest of simplicity, we assume that the all of
the host machines and VMs in the simulated environment are homogeneous. The
tasks processed by each VM is driven by the Google cluster trace data-set detailed
in Section 4.2. Any host’s CPU utilisation greater than 85% is deemed to be over-
utilised. The over-utilised host detection policies will be continuously monitoring
each host machine’s CPU utilisation. A host will stay in an over-utilised state until
necessary VM migrations take place. Live migration occurs once a host becomes over-
utilised and VMs will be moved between hosts. Live migration has a negative impact
on the performance of the host machine and VMs. Voorsluys et al. [190] have shown
that a VMs performance degradation and downtime during live migration depends on
the application’s behaviour (i.e. how many memory pages the application transfers
during live migration). The average performance degradation including the downtime
can be estimated as approximately 10% of the CPU utilisation.

Moreover, in the simulations, we modelled that the same amount of CPU capacity
is allocated to a VM when transferred to the destination host. This means that each
migration may cause service level agreement (SLA) violations, thus, it is crucial to
minimise the number of VM migrations and select a host which will not become
over-utilised if a VM is placed on it. The length of a live migration depends on the
total amount of memory used by the VM and the total network traffic in the cloud
environment which will vary at each time \( t \). Once it has been decided that a host is
over-utilised, the next step is to select particular VMs to migrate from the host. We
have implemented the minimum migration time policy, which selects a VM that will
require the minimum time to complete a migration relative to the other VMs allocated
to the same over-utilised host. VMs are selected based on their RAM utilisation. If
no suitable host is available for in the current time step then no migration will take
place, and the host will remain over-utilised. Each VM will be assigned resources from the assigned host’s RAM and CPU. A sequential migration transfer policy of VMs is implemented in this simulator, which means that VMs will be transferred one after another. Due to workload changes, resources used by the VM will vary, possibly leading to SLA violations and increased energy consumption. SLA violations will occur in host machines when the total demand for the CPU performance exceeds the available CPU capacity $a_h$.

Power consumption by host machines in data centres is determined by the CPU, memory, disk storage, power supplies and cooling systems [137]. Recent studies [61], [119] have shown that the power consumption by a host machine can be accurately described by a linear relationship between the power consumption and CPU utilisation. The power model is similar to the power model used by Beloglazov, where only the HP ProLiant G4 host is used [15]. Table 6.2 represents the power usage at each of the 10% CPU utilisation intervals of the host.

<table>
<thead>
<tr>
<th>Server</th>
<th>0%</th>
<th>10%</th>
<th>20%</th>
<th>30%</th>
<th>40%</th>
<th>50%</th>
<th>60%</th>
<th>70%</th>
<th>80%</th>
<th>90%</th>
<th>100%</th>
</tr>
</thead>
<tbody>
<tr>
<td>HP ProLiant G4</td>
<td>86</td>
<td>89.4</td>
<td>92.6</td>
<td>96</td>
<td>99.5</td>
<td>102</td>
<td>106</td>
<td>108</td>
<td>112</td>
<td>114</td>
<td>117</td>
</tr>
</tbody>
</table>

We implemented the CloudSim’s (2011) network flow model for calculations of the current delay in the network when migrating VMs [69]. They use point to point communication for data from source $u$ to destination $d$ entity which is called the flow and is represented as $f = size_f; u; v$, where $size_f$ is the number of bytes in the flow and $v_n$ is the current VM being migrated from source host $u$ to destination host $d$. The bandwidth that is available between two entities is represented as $bw$, the latency is denoted as $lat$. The duration of a single network flow can be calculated as shown in Equation 6.5:

$$ delay = lat + size_f / bw $$ (6.5)

where $v_n$ is the current VM being migrated from source host $u$ to destination host $d$. Migration from an over-utilised host will stop automatically when the host is under the threshold set at 85% of the host CPU utilisation, and the total delay for the network at each time step is measured as shown in Equation 6.6.

$$ totalDelay = \sum_{v=1}^{n} delay(u, d, v_n) $$ (6.6)
To calculate the impact that one VM migration would have on the network link bandwidth, we defined $bw_l(v_n)$ as the network bandwidth consumed by one VM. The total traffic $Tr$ generated by a group of VMs $V$ from the host selected $h_n$ is as shown in Equation 6.7:

$$Tr(h_n) = \sum_{v=1}^{n} bw_l(v_n) \quad (6.7)$$

Where $v$ is the set of VMs to be migrated. The latency model has a direct effect on the bandwidth model. The higher the latency $lat$, the lower $bw_l$ that will be available. When $v_n$ is migrated from $h_n$ then its utilisation is recalculated and is then compared to a over-utilised threshold. We only consider the host’s CPU utilisation as an indicator to measures if a host machine is over-utilised.

6.5.3 Experiments

The first set of experiments involve comparing the accuracy of performance for each of the prediction algorithms on the bandwidths training and testing data. All algorithms will be judged based on their performance on the metrics outlined in section 4.1. As the RNN is designed to make sequence prediction, it will be applied to the multi-step ahead predictions for both the CPU and bandwidth data.

The next set of experiments will evaluate how each algorithm performs when predicting CPU data. This section is divided into three separate experiments for CPU prediction. The first experiments will examine how each algorithm performs when predicting CPU utilisation on a time scale of one step ahead. The second experiment will examine the performance of the RNN for multi-time step ahead prediction. The purpose of the multi-step ahead experiments is to determine how far into the near future a recurrent network can predict and still produce accurate results. This experiment would be advantageous in real-world data centres as to know in advance if a host will become over-utilised or when bandwidth will become saturated so resources can be re-configured.

The final experiment in this section will measure the prediction accuracy for each algorithm for a single time step and then for the RNN’s multi-time step ahead CPU predictions when trained on one host’s CPU data and tested on a different host’s CPU data. Again this experiment would be advantageous in real-world data centres as when a host is predicted to become over-utilised a destination host must be selected for VMs to be moved onto without causing SLAV’s on the new host.
Finally, the last experiment will evaluate the performance of each one step ahead algorithm with the RNN and how well their predictions can improve a simulated cloud data centres efficiency. Each algorithm’s effectiveness will be measured in SLAs, energy and bandwidth usage.

6.6 Results

This section presents the results of each of the experiments outlined above. First, the results of the bandwidth experiment are presented, followed by the CPU result and lastly the overall simulator results.

6.6.1 Bandwidth Prediction Results

This section presents the results from the range of models that were implemented to predict bandwidth resources. The results for predicting bandwidth a single time step and multiple time steps ahead are presented below.

6.6.1.1 Single Time Step Ahead

A comparison of the accuracy of each of the bandwidth forecasting models for one step ahead predictions is presented below for both the training data and test data sets. The calculated metrics on the training data allows us to measure the performance of the fitted models while the results generated from the test data demonstrates the predictive accuracy and ability of each model to generalise to unseen data.

Table 6.3 evaluates the models across each of the selected performance metrics. The results overall show that non-linear models perform best, this is primarily due to the characteristics of the bandwidth model as displayed in Figure 6.5. As shown the bandwidth data shows significant seasonal patterns with daily peaks and troughs evident throughout. However, the data also remains largely stochastic between successive time-steps. In particular, the Back-propagation (sliding window size 20) algorithm performed best followed by Back-propagation (sliding window size 30) while ARIMA narrowly outperformed BPTT on the training data. Although Linear Regression is a standard prediction method, the experiments results showed it performed the worst out of all of the selected models with a MAPE of 7.7215 on the training data. This is not surprising as the bandwidth data does not display a linear trend, it is stochastic displaying various fluctuations over time and as a result fitting a straight line through the data results in significant errors on both sides of the fitted line as the model struggles to capture any of the variations in the data.
The true accuracy of a predictive model can only be determined by considering how well it performs on new data which was not used to train the model. Table 6.4 presents the results of each predictive model relative to the test set. The results showed that BPTT yielded the highest accuracy when compared to the remaining algorithm with a MAPE of 1.430. In these experiments, the BPTT algorithm shows its capacity to learn complex relationships in the bandwidth time series data but it also highlights its ability to generalise well to unseen data, thus indicating the reliability of the model for future predictions. Similar to what was observed in the training results the Back-propagation (Sliding Window 20) algorithm performed best out of the remaining approaches. Our empirical evaluation found that by adjusting the size of the input sequence from 10 to 20 values, it provides more specific knowledge about the underlying structure of the data resulting in an improved MAPE of 2.3798. Unlike the training results, ARIMA achieved a slightly better result on the test data in comparison to the Back-propagation (Sliding Window 30).

In terms of the more simplistic models, they perform similarly. In particular, they show their inability to model the fundamental characteristics of the bandwidth data with MA performing least best overall with a MAPE of 7.7112. Overall BPTT, ARIMA and Back-propagation with different sized sliding windows resulted in acceptable performance. Figure 6.6 shows the predictive performance of these algorithms relative to the actual values observed in the test data over a single day.
Predicted and Actual Bandwidth Values

Figure 6.5: BPTT and BP (Sliding Window 20)

Predicted and Actual Bandwidth Values

Figure 6.6: BP (Sliding Window 10) and ARIMA
6.6.1.2 Multi-Time Steps Ahead

In the previous experiment, the recurrent neural network trained with the BPTT algorithm outperformed all other approaches. In this experiment, the BPTT algorithm was evaluated regarding its ability to accurately predict bandwidth availability for multiple time steps into the future. In particular, the performance of the BPTT algorithm was assessed based on its ability to predict bandwidth from 1 to 6 time-steps ahead. Each time step represents a 10-minute interval, thus equating to 1 hour of future bandwidth availability which is adequate for predicting resources over a short time span.

Similar to the previous experiment the results for both the training and test sets are presented in Table 6.5 and Table 6.6. As shown, BPTT is capable of achieving significant predictive accuracy when forecasting multiple time steps into the future with a MAPE of between 1.8379 and 3.502 for time steps 1 to 6 for the training data and 1.430 and 2.967 for the test data. This result indicates the reliability and overall robustness of the recurrent neural network for estimating bandwidth resources over a longer time horizon. Also evident as shown in Figure 6.7 is the approximately linear growth in the error of the algorithm the further out into the future it attempts to predict.

Table 6.5: BPTT Multi-step Ahead Training Data Prediction Accuracy

<table>
<thead>
<tr>
<th>Step Ahead</th>
<th>MSE</th>
<th>RMSE</th>
<th>MAE</th>
<th>MAPE</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 Step Ahead</td>
<td>184.9968</td>
<td>13.6014</td>
<td>11.0208</td>
<td>0.0184</td>
</tr>
<tr>
<td>2 Step Ahead</td>
<td>436.0588</td>
<td>20.8820</td>
<td>16.7036</td>
<td>0.0277</td>
</tr>
<tr>
<td>3 Step Ahead</td>
<td>500.8841</td>
<td>22.3804</td>
<td>17.8305</td>
<td>0.0295</td>
</tr>
<tr>
<td>4 Step Ahead</td>
<td>552.9954</td>
<td>23.4967</td>
<td>18.8824</td>
<td>0.0313</td>
</tr>
<tr>
<td>5 Step Ahead</td>
<td>592.6458</td>
<td>24.3443</td>
<td>19.4938</td>
<td>0.0322</td>
</tr>
<tr>
<td>6 Step Ahead</td>
<td>718.8461</td>
<td>26.8113</td>
<td>21.1520</td>
<td>0.0351</td>
</tr>
</tbody>
</table>

Table 6.6: BPTT Multi-step Ahead Test Data Prediction Accuracy

<table>
<thead>
<tr>
<th>Step Ahead</th>
<th>MSE</th>
<th>RMSE</th>
<th>MAE</th>
<th>MAPE</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 Step Ahead</td>
<td>121.9398</td>
<td>11.0426</td>
<td>8.5788</td>
<td>0.0143</td>
</tr>
<tr>
<td>2 Step Ahead</td>
<td>237.5089</td>
<td>15.4113</td>
<td>12.0718</td>
<td>0.0200</td>
</tr>
<tr>
<td>3 Step Ahead</td>
<td>257.1531</td>
<td>16.0360</td>
<td>12.5853</td>
<td>0.0208</td>
</tr>
<tr>
<td>4 Step Ahead</td>
<td>318.3459</td>
<td>17.8422</td>
<td>14.1399</td>
<td>0.0233</td>
</tr>
<tr>
<td>5 Step Ahead</td>
<td>407.0782</td>
<td>20.1762</td>
<td>16.2011</td>
<td>0.0208</td>
</tr>
<tr>
<td>6 Step Ahead</td>
<td>488.2690</td>
<td>22.0968</td>
<td>17.9570</td>
<td>0.0297</td>
</tr>
</tbody>
</table>
6.6.2 CPU Prediction Results

This section presents the results for the three CPU prediction experiments, carried out.

6.6.2.1 Single Time Step ahead

First, we examined the training prediction accuracy and then the testing prediction accuracy for all algorithms for one-time step ahead. Table 6.7 displays MSE, RMSE, MAE, MAPE, for each of the algorithms. Table 6.7 shows that each algorithm performs similarly across all metrics on the training data. Based on the MAPE, BP had the least accuracy, and BPTT has the highest accuracy.

Table 6.7: Train CPU Data Prediction Accuracy

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>MSE (Std Dev)</th>
<th>RMSE (Std Dev)</th>
<th>MAE (Std Dev)</th>
<th>MAPE (Std Dev)</th>
</tr>
</thead>
<tbody>
<tr>
<td>BPTT</td>
<td>0.00761</td>
<td>0.0872</td>
<td>0.0639</td>
<td>0.1395</td>
</tr>
<tr>
<td>Random Walk</td>
<td>0.0078</td>
<td>0.0887</td>
<td>0.0604</td>
<td>0.2644</td>
</tr>
<tr>
<td>Moving Avg</td>
<td>0.0089</td>
<td>0.0947</td>
<td>0.0649</td>
<td>0.3238</td>
</tr>
<tr>
<td>Linear Regression</td>
<td>0.0075</td>
<td>0.0867</td>
<td>0.0608</td>
<td>0.2843</td>
</tr>
<tr>
<td>ARIMA</td>
<td>0.1719</td>
<td>0.0505</td>
<td>0.1498</td>
<td>0.1498</td>
</tr>
<tr>
<td>Back-propagation (Sliding Window 2)</td>
<td>0.0079</td>
<td>0.0893</td>
<td>0.0660</td>
<td>0.3621</td>
</tr>
</tbody>
</table>

Table 6.8 shows the results of each algorithm on the testing data. Again for one step ahead prediction each algorithm performs similarly across all metrics on the
testing data. Based on the MAPE metric BP with sliding window 30 had the worst performance. BPTT performed the best out of all. Figure 6.8 shows the prediction results for both the BPTT and LR algorithm. Both algorithms perform similarly when compared to the actual tested CPU data-set. The recurrent neural network is usually used for sequence prediction, and in the next section, it will be examined on how far into the future it is accurately able to predict CPU data.

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>MSE (Std Dev)</th>
<th>RMSE (Std Dev)</th>
<th>MAE (Std Dev)</th>
<th>MAPE (Std Dev)</th>
</tr>
</thead>
<tbody>
<tr>
<td>BPTT</td>
<td>0.0144</td>
<td>0.1202</td>
<td>0.0818</td>
<td>0.1237</td>
</tr>
<tr>
<td>Random Walk</td>
<td>0.0188</td>
<td>0.1370</td>
<td>0.0843</td>
<td>0.1498</td>
</tr>
<tr>
<td>Moving Avg</td>
<td>0.0228</td>
<td>0.1509</td>
<td>0.0988</td>
<td>0.1516</td>
</tr>
<tr>
<td>Linear Regression</td>
<td>0.0144</td>
<td>0.1201</td>
<td>0.0828</td>
<td>0.1248</td>
</tr>
<tr>
<td>ARIMA</td>
<td>0.1719</td>
<td>0.0565</td>
<td>0.1498</td>
<td>0.1498</td>
</tr>
<tr>
<td>Back-propagation (Sliding Window 2)</td>
<td>0.0146</td>
<td>0.1208</td>
<td>0.0859</td>
<td>0.1278</td>
</tr>
</tbody>
</table>

6.6.2.2 Multi Time Step ahead

The aim of the multiple time steps ahead prediction experiment was to evaluate how far into the future the recurrent neural network could accurately predict CPU utilisation of a host and to determine by how much does predictions decrease the further into the future the network attempts to predict. This experiment involved predicting the CPU utilisation of a host machine from one to six-time steps into the
future. Each of these time steps corresponds to 10 minutes, for instance, time step six relates to a prediction 1 hour into the future.

Table 6.9 and Table 6.10 present the accuracy of the prediction on the training and testing datasets at each of the six-time steps. It is clear by the results, each performance metric shows a linear increase in the error of prediction the further into the future the recurrent network tries to predict. Up to six time steps ahead was chosen as the max future time-step as it corresponds to one hour of CPU information and this is a sufficient time into the future for the planning of resources.

Table 6.9: Multi-step Ahead Trained Data Prediction Accuracy

<table>
<thead>
<tr>
<th>Step Ahead</th>
<th>MSE (Std Dev)</th>
<th>RMSE (Std Dev)</th>
<th>MAE (Std Dev)</th>
<th>MAPE (Std Dev)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 Step Ahead</td>
<td>0.0144</td>
<td>0.129</td>
<td>0.0818</td>
<td>0.1237</td>
</tr>
<tr>
<td>2 Step Ahead</td>
<td>0.013</td>
<td>0.126</td>
<td>0.084</td>
<td>0.179</td>
</tr>
<tr>
<td>3 Step Ahead</td>
<td>0.015</td>
<td>0.126</td>
<td>0.088</td>
<td>0.191</td>
</tr>
<tr>
<td>4 Step Ahead</td>
<td>0.019</td>
<td>0.137</td>
<td>0.106</td>
<td>0.220</td>
</tr>
<tr>
<td>5 Step Ahead</td>
<td>0.021</td>
<td>0.144</td>
<td>0.112</td>
<td>0.232</td>
</tr>
<tr>
<td>6 Step Ahead</td>
<td>0.024</td>
<td>0.154</td>
<td>0.122</td>
<td>0.248</td>
</tr>
</tbody>
</table>

Figure 6.9 highlights the MAE for both 1-time-step-ahead and 6-time-steps-ahead for the BPTT algorithm on the test data. Figure 6.9 shows that the errors increase the further into the future the algorithm predicts. The graph also highlights where the BPTT algorithm struggles when making predictions. As shown in Figure 6.8 at time step 22 when there is an extremely sudden change of CPU utilisation, Figure
Table 6.10: Multi-step Ahead Test Data Prediction Accuracy

<table>
<thead>
<tr>
<th>Step Ahead</th>
<th>MSE (Std Dev)</th>
<th>RMSE (Std Dev)</th>
<th>MAE (Std Dev)</th>
<th>MAPE (Std Dev)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 Step Ahead</td>
<td>0.0144</td>
<td>0.1202</td>
<td>0.0818</td>
<td>0.1237</td>
</tr>
<tr>
<td>2 Step Ahead</td>
<td>0.0259</td>
<td>0.1608</td>
<td>0.1200</td>
<td>0.1890</td>
</tr>
<tr>
<td>3 Step Ahead</td>
<td>0.0301</td>
<td>0.1734</td>
<td>0.1332</td>
<td>0.2140</td>
</tr>
<tr>
<td>4 Step Ahead</td>
<td>0.0333</td>
<td>0.1824</td>
<td>0.1447</td>
<td>0.2320</td>
</tr>
<tr>
<td>5 Step Ahead</td>
<td>0.0356</td>
<td>0.1887</td>
<td>0.1555</td>
<td>0.2443</td>
</tr>
<tr>
<td>6 Step Ahead</td>
<td>0.0377</td>
<td>0.194</td>
<td>0.1617</td>
<td>0.255</td>
</tr>
</tbody>
</table>

6.9 shows at the same time step that the BPTT MAE accuracy also decreases, this is a result of the extreme variance of the CPU workloads. However, considering this the algorithm performs well when predicting six times steps into the future.

6.6.2.3 Unseen Host Data-Set

The next set of experiments involved evaluating the performance of each algorithm on a host’s CPU data-set which it was not trained on. The results from this experiment showed how well each generalises when applying to new unseen data. Table 6.11 shows how well each of the metrics that BPTT outperforms all other algorithms, with BP with 10 sliding windows having the worst performance.

Table 6.11: Test Prediction of New Host CPU Data

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>MSE (Std Dev)</th>
<th>RMSE (Std Dev)</th>
<th>MAE (Std Dev)</th>
<th>MAPE (Std Dev)</th>
</tr>
</thead>
<tbody>
<tr>
<td>BPTT</td>
<td>0.012</td>
<td>0.107</td>
<td>0.081</td>
<td>0.122</td>
</tr>
<tr>
<td>Random Walk</td>
<td>0.015</td>
<td>0.121</td>
<td>0.085</td>
<td>0.132</td>
</tr>
<tr>
<td>Moving Avg</td>
<td>0.014</td>
<td>0.109</td>
<td>0.082</td>
<td>0.127</td>
</tr>
<tr>
<td>Linear Regression</td>
<td>0.013</td>
<td>0.108</td>
<td>0.081</td>
<td>0.123</td>
</tr>
<tr>
<td>ARIMA</td>
<td>0.1719</td>
<td>0.0505</td>
<td>0.1498</td>
<td>0.1498</td>
</tr>
<tr>
<td>Back-propagation (Sliding Window 2)</td>
<td>0.013</td>
<td>0.107</td>
<td>0.082</td>
<td>0.125</td>
</tr>
</tbody>
</table>

Similarly, in this experiment, the recurrent neural network was examined on its ability to predict another host’s CPU usage based on training it completed on another host’s CPU data. Table 6.12 shows how far into the future the recurrent neural network could accurately predict CPU utilisation. The results show a linear increase in the error of prediction the further into the future the recurrent network tries to predict. However, the predictions produced for even 6-times steps into the future are relatively accurate. These predictions help the RNN approach to decide in advance if a host will become over-utilised so to not place another VM on it, preventing further degradation. Also, it is important from a Data Centre perspective, that a prediction model can generalise well to unseen data as, it would not be possible to train models for every host (as there are just too many), so how well an approach can generalise across unseen hosts is an important evaluation.
### 6.6.3 Simulation Results

This section presents the performance of each algorithm in a simulated data centre environment. BPTT has the abilities to predict multi-times into the future (60 minutes) for both CPU and Bandwidth while the rest of the algorithms are one step ahead prediction (i.e. 10 minutes) for CPU only. As Back-propagation (2 sliding windows) has outperformed each of the other BP sliding windows approaches, it will be the only network evaluated in the simulation experiment. The goal of this experiment was to investigate how well a multi-time step ahead approach can improve data centre efficiency by determining when a host is over-utilised and when is the best time to initiate live migration.

The first metric that was examined was the Service Level Agreement Violations (SLAV). Maintaining low occurrences of SLAV’s is an essential factor in the delivery of reliable quality assured cloud-based services. In this regard, it is imperative to consider the number of SLAVs incurred by all approaches throughout the simulation. Table 6.13 presents the average SLAV’s each algorithm occurred during the simulations. The lower the SLAV value is, the better the data centre is performing. The BPTT multiple-step ahead predictions enabled it to anticipate well in advance if any over-utilisation was to occur in any of the hosts. Being able to make accurate predictions multiple time steps into the future allows the cloud management system to start live migration before a host becomes over utilised and select a suitable host for a VM which would not become over-utilised in the next hour. BPTT averaged a SLAV of just 1.80776E-06. This is nearly 87% better than the next lowest value from Previous Walk (PW). BP had the worst SLAV of all approaches. A T-test was performed between the best algorithm and the second best algorithm for SLAV. The two-tailed P value is less than 0.0001 and the results are deemed statistically significant with a 95% confidence interval of this difference from -15.82188702780 to -8.95795613255, BPTT had a standard deviation of 5.53272815845 compared to 20.17874430023 for PW. Figure 6.11 presents the results for all algorithms.

<table>
<thead>
<tr>
<th>Step Ahead</th>
<th>MSE (Std Dev)</th>
<th>RMSE (Std Dev)</th>
<th>MAE (Std Dev)</th>
<th>MAPE (Std Dev)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 Step Ahead</td>
<td>0.012</td>
<td>0.107</td>
<td>0.081</td>
<td>0.122</td>
</tr>
<tr>
<td>2 Step Ahead</td>
<td>0.016</td>
<td>0.127</td>
<td>0.098</td>
<td>0.15</td>
</tr>
<tr>
<td>3 Step Ahead</td>
<td>0.021</td>
<td>0.145</td>
<td>0.112</td>
<td>0.169</td>
</tr>
<tr>
<td>4 Step Ahead</td>
<td>0.026</td>
<td>0.161</td>
<td>0.126</td>
<td>0.189</td>
</tr>
<tr>
<td>5 Step Ahead</td>
<td>0.03</td>
<td>0.172</td>
<td>0.134</td>
<td>0.199</td>
</tr>
<tr>
<td>6 Step Ahead</td>
<td>0.032</td>
<td>0.177</td>
<td>0.14</td>
<td>0.204</td>
</tr>
</tbody>
</table>
Figure 6.10 shows the energy consumption at each of the time steps. As can be seen from the graph BP had the lowest energy consumption of all approaches with an average of 340.67 kWh per host. BPTT had the second lowest energy consumption of 457.72 kWh per host and PW had the worst energy consumption of all approaches with the average energy consumption of 764.22 kWh per host. One reason for PW having the worst energy consumption is that it too has the highest average migration count in the simulation with 197 and both neural network approaches had the lowest migrations with 175 for BPTT and 120 for BP. BPTT achieved a relatively low average energy consumption within the data centre. One reason being is that it selected optimal times to migrate when more bandwidth was available and had better accuracy of prediction also, thus providing faster migration and ensuring the host never enters an over-utilised state.

The BPTT ability to store past information makes it an ideal algorithm for sequence prediction of cloud resources. BPTT produced the most accurate predictions for the bandwidth utilisation both for one and multi-time steps ahead. BPTT achieves an average bandwidth usage of 610 MB per time step where PW achieved the worst overall results of 646 MB per time step. Migrating at times when bandwidth is available led to the BPTT having lower migration times for VMs. By choosing a specific time to schedule a migration allowed the BPTT algorithm to utilise available resources at critical times better.

<table>
<thead>
<tr>
<th>Metric</th>
<th>SLAV</th>
<th>ESV</th>
</tr>
</thead>
<tbody>
<tr>
<td>BPTT</td>
<td>1.80776E-06</td>
<td>0.000827449</td>
</tr>
<tr>
<td>Previous Walk</td>
<td>1.41977E-05</td>
<td>0.010850105</td>
</tr>
<tr>
<td>Moving Average</td>
<td>1.89143E-05</td>
<td>0.013299876</td>
</tr>
<tr>
<td>Linear Regression</td>
<td>2.60094E-05</td>
<td>0.016313468</td>
</tr>
<tr>
<td>Back-propagation</td>
<td>2.7531E-05</td>
<td>0.009379006</td>
</tr>
</tbody>
</table>

The SLA and energy metrics combine to create a metric known as ESV (i.e. ESV = energy*SLAV). This metric measures the overall data centre performance, in-terms of minimising energy and reducing SLA’s. It is desirable to obtain a method which will consume less power and still incur less SLA violation.

The lower the ESV, the better the performance the data centre is achieving. Overall the BPTT algorithm achieved the best ESV performance of 0.000827449 compared to the second best, which is PW of 0.010850105. This is an improvement of 61% of data centre efficiency. Again a T-Test was conducted between the two best performing algorithms. The two-tailed P value is less than 0.0001 and the results are
Figure 6.10: Energy Consumption of all Algorithms

deemed statistically significant with a 95% confidence interval of this difference from -0.01458090588 to -0.00688034372, BPTT had a standard deviation of 0.00424366520 compared to 0.02314924312 for PW.

6.6.4 Discussions

The results of the experiments show that the recurrent neural network has the capabilities to improve upon traditional prediction methods for predicting host CPU utilisation and bandwidth to determine better when live migration should be scheduled. Even though the bandwidth data-set does contain a learnable pattern, the CPU data-set used has varying demands which makes it much harder to predict CPU utilisation of host machines.

The first experiment determined how a Recurrent Neural Network could outperform tradition forecasting methods in predicting the next bandwidth value. The first part of this experiment indicated from the MSE, MAE, RMSE and MAPE that a Recurrent Neural Network trained with the BPTT algorithm achieved the highest accuracy as shown in table 6.4. The second part of this experiment examined how far into the future the recurrent network could predict bandwidth with a high degree of accuracy. The results indicate that the recurrent neural network can produce a reasonable degree of accuracy when predicting multiple time steps into the future even though it only needs two previous inputs to achieve this.
The second experiment involved determining which algorithm would produce the most accurate results in predicting CPU utilised. From the results, BPTT achieved the highest results with an MAE accuracy of 0.0818 on the test data. BPTT also produced the most accurate predictions when trained on one host’s CPU utilisation data and tested on a new host CPU data.

The results from the final experiment highlight the efficiency of the BPTT algorithm. Each of the forecasting methods was tested in a simulated environment on the following metrics; energy, SLAV and bandwidth usage metrics. The BP algorithm achieved the lowest energy, but the highest SLAVs. The PW algorithm achieved the second lowest SLAV but the highest energy consumption. However, the BPTT achieved a statistically significant lower SLAV and ESV values, improving the data centres efficiency by 61% when compared to the next best algorithm. One of the main reasons BPTT could achieve the most efficient results is because it can identify when host machines will become over-utilised and the best time to schedule migration based on the network bandwidth.

Multi-time step ahead prediction is a problematic area in time series research. The additional noise present in both the CPU and bandwidth utilisation data makes forecasting far into the future even more difficult. The BPTT forecasting method presented in this research could potentially be incorporated with many of the other sub fields of cloud computing that involve host prediction, VM migration and task scheduling to improve overall performance.
The results presented demonstrate that recurrent neural networks are capable of predicting CPU data 1 hour ahead with a reasonably high degree of accuracy despite the noise in the data set. It is known that instantiating a new virtual machine takes between 5-15 minutes [98]. In real-world data centres the recurrent network could be implemented to inform the cloud management system as to when a host is going to become over/under utilised so the management system can take action and boot up new VM instances on a different host before initiating live migration from the over-utilised host. This, in-turn, would lead to smoother transitions of VMs being moved from a source host to a destination, reducing live migration times and decreasing the occurrences of SLAs on host machines.

6.7 Limitations

Even though results from this chapter show the recurrent neural network can improve resource prediction for live migration, its limitation must too be considered.

For example, one has to consider the amount of time for training a neural network, selecting the correct number of layer and neurons while also tuning each hyper-parameter too. One also has to consider the different types of neural networks which could be applied to improve results. In this chapter we implemented a one layer deep network with produce reasonably accurate predictions. However, a more extensive network with multiple layers could improve the results further but would lead to longer training times.

Other limitations of this approach could be the fact that if the neural network did not generalise well across other host machines, then a network would have to be trained for each host machines, which could be upwards of 10,000 machines in one data centre. In this work, the results show from table 6.11 that the RNN generalised well across different host machines. However, to test the network’s ability more workloads with varying patterns from extreme heavy or light loads, to non-fluctuating workloads on a wide range of different host machines could be considered.

6.8 Comparison with the State of the Art

The results demonstrated in this chapter match and contributed to improving the state of the art methods. Compared with Ghorbani et al. [72] who utilises upper
and lower thresholds to guarantee network bandwidth, the approach proposed in this chapter predicts when a host will first become over-utilised and then decides the best times to migrate a VM, utilising available bandwidth.

Piao et al. present a network-aware VM placement and migration approach. Their algorithm places a VM on a host machine considering network link between the source and destination host [146]. The work in this chapter utilises an RNN architecture to build upon Piao et al. work by predicting future bandwidth before migrating a VM.

6.9 Summary

The contributions of this chapter can be summarised as follows:

- A recurrent network produces more accurate results for predicting both CPU and bandwidth utilisation when compared to other linear and Non-linear approaches.

- An RNN has the abilities to predict bandwidth and CPU with reasonably high accuracy up to 30 minutes into the future.

- The predictions of the RNN can help the cloud management systems to be more efficient for scheduling live migration while reducing energy and SLAV within a simulated cloud environment.
Chapter 7

Conclusion

7.1 Introduction

The research presented within this Thesis has broadly focused on applying techniques from a subset of Machine Learning and Artificial Intelligence algorithms known as Reinforcement Learning and Neural Networks, to address two critical issues prevalent within modern day IaaS clouds; performance unpredictability and live migration scheduling. Previously highlighted by Armbrust et al. these are two critical obstacles preventing the greater growth and adoption of cloud computing infrastructures.

This Thesis presented a number of novel solutions, utilising learning techniques to address these two major obstacles. To examine the efficiency of our proposed solutions, we conducted extensive evaluations of the machine learning algorithms on real-world data throughout this thesis, including simulations of host resource usage, bandwidth demands, and scheduling of live migration. Within each of these experiments, a variety of conditions and settings were simulated, including fluctuating incoming network traffic demands and variable resource performance.

The main contributions of this thesis are (1) the application of machine learning algorithms to real world cloud data models and (2) the examinations of how reinforcement learning and neural networks can be applied to solve principal problems such as performance unpredictability and the scheduling of live migration.

7.1.1 An Autonomous Approach for the Selection of Virtual Machines for Live Migration

This chapter proposed an autonomous network aware VM migration strategy that acts as a decision support system. An RL-based approach learned to select VMs to be migrated during varying network traffic loads. The agent observed real time
network resource usage and decided on the optimal action (which VM to select) to perform.

Compared with the current state of the art, where VMs are selected based on the lowest RAM usage [15],[14], the approach in this work considers both RAM and bandwidth. The work presented in this Thesis demonstrated the adaptive nature of reinforcement learning, through the selection of VMs to be migrated. The RL approach showed that it has the abilities to learn and decide which sized VMs to migrate when utilisation of network traffic is low. This in-turn reduced congestion at peak times, and utilises available network resource at on/off peak times. Through the RL agent approach, a cognitive live migration framework is created.

1. We presented an autonomous network aware live migration strategy that has the capabilities to learn the optimal VM to be migrated based on current resource levels and RAM size. We created a novel multidimensional state-action space, represented by the utilisation of bandwidth available, the direction of the network traffic demand (increasing, decreasing) and an action to be performed.

2. From the results, there is a case to consider the size of VM and the current network conditions to decide when to schedule live migration. RL has shown to reduce migration times and overhead cost to ensure cloud resource are available during peak hours applications when compared to Beloglazov et la. proposed VM selection strategies [15].

7.1.2 Predicting Host CPU Utilisation

Cloud Computing Using Recurrent Neural Networks

The primary aim of this research was to investigate the capability of neural networks for accurately predicting CPU utilisation for short time periods. The results obtained indicate that it is in fact, possible to predict CPU utilisation with a high degree of accuracy for short time periods and on data sets that have sudden extreme changes. All four algorithms (BPTT, PSO, DE and CMA-ES) were able to train a network to accurately predict CPU utilisation within 10,000 evaluations of the training data. Of the four algorithms evaluated CMA-ES performed the best on the training data followed by DE then BPTT and lastly PSO. On the test data, however, CMA-ES and DE performed equally well measure with the MAE metric with BPTT outperforming the other four algorithms for the MSE metric. It was discovered however that the prediction of the CPU utilisation is a difficult task due to the occasional sudden extreme change in CPU utilisation. All prediction algorithms struggled to predict
these rapid changes. It was also shown that the evolved networks were also capable of accurately predicting CPU utilisation data from a different host machine. The accurate results of new host data indicates that there are some reoccurring patterns and regularities in the CPU utilisation data that the networks are capable of exploiting to give a good general performance. It was also found that the networks prediction accuracy does decrease as it predicts further into the future however on average the network is capable of predicting with a reasonable level of accuracy 4 steps (20 minutes) into the future.

Compared with state of the art, the trained RNN out-performed traditional prediction models such as linear regression, which was used by Farahnakian et. al. [63]. The RNN presented here also has several advantages when compared to approaches such as the ARIMA model used in [112], as it works well for one time step ahead prediction, multi-step-ahead prediction and the prediction can be adapted as the workload changes.

In summary, the contributions of this research are:

1. It is possible to evolve a network capable of accurately predicting host CPU utilisation.

2. CMA-ES converged fastest to a good solution and gave the best network prediction accuracy on the training data. On the test data, CMA-ES and DE provide a statistically equal network prediction accuracy. CMA-ES would be the algorithm of choice if rapid convergence is desired however DE is far simpler to implement and gives an equal performance on the test data.

3. The recurrent networks are capable of accurately predicting CPU utilisation on unseen data from both the same and also new hosts, demonstrating a high degree of generality.

4. The accuracy of the network predictions decreases linearly as the networks attempt to predict further into the future.

7.1.3 Multi-Time Step Ahead Prediction Approach for Scheduling Live Migration in Cloud Data Centres

To maximise resource usage within a data centre and to ensure SLA are not violated, resource management strategies must be able to predict how each host is going to perform in the future. This chapter conducted a competitive analysis of the traditional nonlinear and linear algorithms against Recurrent Neural Network. Each
algorithm was evaluated through extensive simulations on a large-scale experiment setup, utilising the Google clustering trace data.

The results of the experiments have shown that the memory retention and sequence prediction of a Recurrent Neural Network allowed it to produce the most accurate predictions for both CPU and bandwidth utilisation. The Recurrent Neural Network also was able to decrease bandwidth usage during critical times, reduce the occurrence of SLAVs and improve the overall efficiency of a cloud data centre.

Compared to state of the art the RNN live migration approach builds upon previous work [72, 40, 171, 146] by considering bandwidth before migrating VM but also deciding optimal times to migrate a VM before it will become over-utilised, be it for one or multi-time steps in to the future.

In summary, the main findings of this research are:

1. Recurrent Neural Networks produce better predictions for host CPU and network bandwidth utilisation when compared to traditional models.

2. Recurrent Neural Networks produce high accuracy when predicting multi-times steps into the future for both CPU and bandwidth data. In both cases, the accuracy of predictions decreases linearly the further into the future the network attempts to predict.

3. The Recurrent Neural Networks produce statistically significant performance results when compared to traditional prediction algorithms to reduce SLAVs and decrease energy consumption.

7.1.4 Limitations

7.1.4.1 State-Action-Space of Reinforcement Learning

This research implemented a reinforcement learning approach which had a multi-state action space that was defined by host utilisation, bandwidth availability and virtual machine utilisation of selected host. However, in data centres, there are more factors to consider when migrating VMs, such as RAM utilisation, and the destination host’s future utilisation. If each of these parameters were also factored into the current state-action-space, a better solution for the overall data centre performance could potentially be found. However, this would come at a significant cost to learning times as the state-action space would increase extensively (‘curse of dimensionality’).
7.1.4.2 Uni-variate Forecasting of Cloud resources

This research only focused on CPU when forecasting resource utilisation of a host machine. Other significant resources such as RAM, and disk, would likely increase the prediction accuracy of the neural networks, giving greater insight to which of the host resources will degrade in the future. One of the main aims of this research was to show the effectiveness of the Recurrent Neural Network when predicting sequences of CPU utilisation. However, to show this, we implemented a univariate approach which is a limitation of this thesis.

7.1.4.3 Implementing Resource Predictions in Data Centre Management

With cloud resources being in a constant state of flux, incorporating prediction into the cloud management system could prove to be difficult. For example, consider if a host has been predicted to become over-utilised in the next 30 minutes and a virtual machine is migrated based on this knowledge. If then a task running on a VM suddenly finishes altering the resource consumption, it might have not been necessary for VM migration to have taken place.

7.1.4.4 The Adaption of AI in Data Centres

AI and ML techniques have proven to be useful when applied to a wide range of complex problems and domains such as energy optimisation [130], video gaming [178], helicopter aerobatic [184] and also cloud computing [12]. Companies such as Google have begun to invest heavily in applying AI to improve the efficiency of their data centre. However, one of the major challenges facing the greater adaption of intelligent problem-solving in cloud data centres is the lack of user-friendly solutions. AI needs a more of a ‘plug and play’ solution for service providers instead of investing in specialised teams to produce custom AI solutions.

7.1.5 Future Work

As shown from the each of the result chapters (4,5,6), machine learning has the potential to help increase the efficiency of a data centre, which is also highlighted by Google’s data centre research [59]. It is hoped that more efficient virtual machine migration process can be achieved by utilising a deep neural network for multi-variate resource prediction. Also, it would be advantageous to consider a continuous state-space for the VM selection policy, for this a function approximator would be required. Deep learning can be used to help Reinforcement Learning to generalise its training
knowledge and apply it to unseen states in a continuous state-space [139] [138]. Another method to improve the performance and robustness of RL is to apply potential based reward shaping which give extra incentives to the agent whilst learning [50] [51] [77].

7.1.6 Final Conclusion

Our research into applying Neural Networks and Reinforcement Learning to cloud computing has shown the possible improvements in performance of cloud data centres compared to solutions currently employed by researchers. We have also demonstrated that these proposed improvements, can be financially advantageous to companies, environmentally friendly and advance the research in the cloud computing area. We look forward to carrying out further research and to enhance the cloud computing industry.
Bibliography


[9] Paul Barham, Boris Dragovic, Keir Fraser, Steven Hand, Tim Harris, Alex
Ho, Rolf Neugebauer, Ian Pratt, and Andrew Warfield. Xen and the art of
virtualization. In ACM SIGOPS operating systems review, volume 37, pages

Boutaba. Cqncr: Optimal vm migration planning in cloud data centers. In

scheduling workflow applications in the cloud. In Web Services (ECOWS),

[12] Enda Barrett, Enda Howley, and Jim Duggan. Applying reinforcement learn-
ing towards automating resource allocation and application scalability in the
1674, 2013.

source allocation heuristics for efficient management of data centers for cloud

[14] Anton Beloglazov and Rajkumar Buyya. Energy efficient resource manage-
ment in virtualized cloud data centers. In Proceedings of the 2010 10th IEEE/ACM
International Conference on Cluster, Cloud and Grid Computing, pages 826–

[15] Anton Beloglazov and Rajkumar Buyya. Optimal online deterministic algo-
rithms and adaptive heuristics for energy and performance efficient dynamic
consolidation of virtual machines in cloud data centers. Concurrency and Com-

[16] Anton Beloglazov and Rajkumar Buyya. Managing overloaded hosts for dy-
namic consolidation of virtual machines in cloud data centers under quality of
service constraints. IEEE Transactions on Parallel and Distributed Systems,

dynamic and energy-efficient consolidation of virtual machines in openstack


[56] Xavier Dutreilh, Nicolas Rivierre, Aurlien Moreau, Jacques Malenfant, and Isis Truck. From data center resource allocation to control theory and back.


