Load Balancing and Rate Limiting Based Algorithms for Improving Cloud Computing Performance

Joseph Doyle

A thesis submitted for the degree of
Doctor of Philosophy in Computer Science

University of Dublin, Trinity College
School of Computer Science and Statistics

Supervisor: Prof Donal O'Mahony

September 2012

Declaration

I declare that the work in this thesis has not been submitted for a degree at any other university, and that the work is entirely my own.

Signature

Joseph Doyle

September 2012

Permission to Lend and/or Copy

Acknowledgements

Firstly I would like to thank Professor Donal O'Mahony and Professor Robert Shorten for their wisdom and guidance throughout my postgraduate work. This work would be a pale shadow of itself without their knowledge and advice.

Many thanks to Rade Stanojević and Florian Knorn for their discussions about algorithms in communication networks which greatly influenced this work. I thanks Dimitris Kalamatianos for his discussions about conjugate gradient algorithms.

I thank Andreas Simon-Kajda and Dirk Niemeier for their help with the computational fluid dynamics simulations.

Hearty thanks to all my colleagues in CTVR for their encouragement and for providing such a pleasant working atmosphere. Thanks to all my friends for preventing me from constantly looking at a monitor screen.

I am deeply indebted to my family for their understanding and support. I especially thank Matthew, Lisa and Joe for their help with the enclosure construction. I also thank Pauline for proofreading this work.

Finally, I thank my girlfriend Laura for her immense support and understanding when work had to come before life.

Abstract

The cloud computing paradigm has recently become an increasing popular method for the delivery of internet services. While the increased utilisation of the cloud has made it easier for cloud operators to recover their capital investment further work can be done to maximise profits. The cost of constructing a cloud is considerable (hundreds of millions of Euro) and the operating cost is also significant (tens of millions of Euro annually). Cloud operators can attempt to ensure that their capital investments are recovered by lowering operating costs and using mechanisms which endeavour to maintain high utilisation.

In this dissertation we begin by examining methods which cloud operators can use to control bandwidth utilisation to ensure high utilisation. Fixed cost pricing is more desirable to enterprises and we examine the use of distributed rate limiting to achieve this in the cloud. In addition, some clouds are public environments and as such mechanisms that ensure that each user receives a fair share of the bandwidth available are useful in maintaining quality of service levels. We propose the use of a dropping mechanism to achieve this and compare it with the established token bucket mechanism.

We then shift our attention to investigate methods which cloud operators can use to lower the operational costs. Firstly we propose algorithms to lower cooling costs while ensuring all demand is serviced. Data centres generate a lot of heat and this must be removed to prevent damage to equipment. Some data centres are excessively cooled to cater for worst-case airflows. By equalising the inlet temperature of server racks within a data centre we attempt to lower the cooling cost. Secondly we examine algorithms to lower carbon emissions while maintaining a reasonable Quality of Service (QoS) level. The carbon intensity of electricity suppliers, which is the carbon emitted to produce a given amount of electricity, varies in different geographical regions. Our work formulates the operation of the cloud as an optimisation problem and applies the subgradient method to minimise the combination of average service request time and carbon emissions. Finally we propose an algorithm which uses Voronoi partitions to minimise a function which encompases the electricity cost, carbon emissions and average service request time. Both the electricity cost and carbon emissions are affected by the cooling design used at the data centre and we also incorporate this in our simulations in this section. We gather carbon intensity, electricity price and latency data for a cloud which has data centres in USA and Europe (similar to a popular commercial cloud). Our work then utilises this data and examines how this algorithm can be used to achieve a variety of goals such as minimising carbon emissions, electricity cost and average service request time.

Contents

Declaration **ii**

Permission to Lend and/or Copy **iii**

Acknowledgements **iv**

Abstract **v**

1 Introduction **1**

1.1 Key Contributions 3

1.2 Disertation Outline 3

1.3 Publications arising from this work 4

2 Background **5**

2.1 Introduction 5

2.2 Cloud Computing 7

2.2.1 Models 7

2.3 Hardware 8

2.3.1 Data Centre Architecture 9

2.3.2 Cooling 12

2.3.3 Bandwidth Management 15

2.3.4 Data Centre Costs 18

2.4 Applicable Theory 20

2.4.1 Consensus Algorithms 20

2.4.2 Voronoi Paritions 21

2.5 Load Balancing 23

2.5.1 Current Methods 23

2.5.2 Current Research 27

2.6 Rate Limiting 28

2.6.1 Distributed Rate Limiting 28

2.7 Summary 30

3 DRL Pricing 32

3.1 Abstract 32

3.2 Introduction 32

3.3 Mathematical Framework 34

3.3.1 Cloud Control With Constant Probabilities (C3P) 34

3.3.2 Distributed Deficit Round Robin (D2R2) 35

3.3.3 Improving Resilience to Failure 37

3.3.4 Updating Aggregate Capacity 38

3.4 Experimental Setup 39

3.5 Experimental Results 40

3.5.1 Comparison with Simulation 41

3.5.2 Failure Tolerance 44

3.5.3 Changing Aggregate Capacity 48

3.6 Summary 49

4 Bandwidth Management in Data Centres 52

4.1 Abstract 52

4.2 Introduction 52

4.3 “Fair-share” Dropping Mechanism 54

4.4 Experimental Setup 55

4.5 Experimental Results 56

4.6 Summary 57

5 Thermal Management of Data Centres 59

5.1 Abstract 59

5.2 Introduction 59

5.3 Preliminaries 60

5.3.1 Problem setting 61
5.3.2 Global Demand and Local Temperature Exchange (GDLTE) 63
5.3.3 Local Demand and Local Temperature Exchange(LDLTE) 65
5.4 Simulation Setup 67
5.5 Simulation Results 72
5.5.1 Simulation 1 73
5.5.2 Simulation 2 75
5.5.3 Simulation 3 77
5.6 Experimental Setup using Real Hardware 78
5.7 Experimental Results 80
5.8 Summary 81

6 Carbon Emissions Control **82**
6.1 Abstract 82
6.2 Introduction 82
6.3 Mathematical Preliminaries 83
6.3.1 Problem Formulation 83
6.3.2 Algorithm 84
6.4 Simulation Setup 85
6.5 Simulation Results 89
6.6 Summary 91

7 Unified Global Load Balancing **92**
7.1 Abstract 92
7.2 Introduction 92
7.3 Mathematical Preliminaries 93
7.3.1 Problem Formulation 93
7.3.2 Pairwise Partitioning Rule 95
7.4 Analysis 95
7.4.1 Electricity Cost 96
7.4.2 Carbon Emissions 97
7.4.3 Cooling Cost 99
7.4.4 Average Job Time 102

7.5 Simulation Setup 105
7.6 Simulation Results 108
7.6.1 Overall Performance 109
7.6.2 Request Distribution 110
7.6.3 Performance Variance 111
7.6.4 Balanced Performance 113
7.7 Summary 114

8 Conclusions and Future Work **117**

Bibliography **121**

List of Tables

2.1 Percentage breakdown of cost of cloud [57]. 20

3.1 Mean and Standard deviation of aggregate forwarding rates in C3P and D2R2. . 42

3.2 Jain's Fairness index [73] for C3P and D2R2. 44

7.1 Average round trip time between data centres and sources of requests, carbon intensity of data centres and sources of requests and daily number of requests at source 100

7.2 Average Service Request time, Daily Carbon Emission and Number of Requests Serviced at Each DC for Various Scenarios 108

List of Figures

2.1 Illustration of a parties involved in public cloud computing and the products and services that the parties provide. 7

2.2 A conventional data centre network architecture. 9

2.3 Diagram of raised floor plenum CRAC unit cooling. 13

2.4 Example of how sources of requests are partitioned between two data centres. Colour indicates that the node is part of a particular partition and numbers on the edges are weights. 22

2.5 Example of how NetScaler functions in a cloud with two data centres. 24

2.6 Simple example of DRL. 29

3.1 Pseudo Code of C3P [143]. 34

3.2 Pseudo Code of D2R2 [143]. 36

3.3 Topology of Experiment. 39

3.4 Forwarding rates achieved by 55 concurrent flows. 42

3.5 Loss Rate of C3P. 43

3.6 Aggregate forwarding rate during limiter failure without *good-neighbours*. 45

3.7 Loss rates of operating C3P limiters during limiter failure without *good-neighbours*. 45

3.8 Augment “fair-share” of operating C3P limiters without *good-neighbours*. 46

3.9 Augmented “fair-share” of operating D2R2 limiters during limiter failure with *good-neighbours*. 46

3.10 Loss Rate of C3P limiters during limiter failure with *good-neighbours*. 46

3.11 Aggregate forwarding during failure of limiters with *good-neighbours*. 47

3.12 Loss Rate of C3P limiters with changing aggregate capacity. 49

3.13 Aggregate forwarding rate with varying aggregate capacity. The specified aggregate capacity begins at 40Mbps and changes to 60Mbps at 250s, 15Mbps at 500s and 30Mbps at 750s. 50
3.14 Augmented "fair-share" of D2R2 limiters with changing aggregate capacity. . . . 50

4.1 Pseudo Code of "fair-share" dropping mechanism. 54
4.2 Diagram of the experimental setup. 55
4.3 The forwarding rates of the individual TCP flows using (a) "fair-share" and (b) token bucket. 56

5.1 Illustration of how the thermal management algorithms operate. 61
5.2 Pseudocode for GDLTE. 65
5.3 Pseudocode for LDLTE. 66
5.4 Optional caption for list of figures 68
5.5 Layout of simulation setup. To illustrate the communication graphs used in simulation 3, the dotted lines indicate which other machines the racks 1 and 13 from the left cold aisle can exchange temperature information with. Server racks in adjacent aisles are connected to lower temperature differences between aisles. . 70
5.6 Utility functions. 71
5.7 Performance of GDLTE at three demand levels $D^* = \{40\%, 55\%, 25\%\}$, which are indicated by the dashed line in the top plot, using $\eta = 0.1$, $\mu = 1/3$. 73
5.8 Performance of GDLTE with demand varying in a periodic fashion, using $\eta = 0.1$, $\mu = 1/3$. 74
5.9 Performance of LDLTE with constant demand of 40%, using $\eta = 0.1$. 75
5.10 Performance of LDLTE with a single broken link and constant demand of 40%, using $\eta = 0.1$. 76
5.11 $\eta = 13$. The temperature of inlets of the server racks at each iteration of LDLTE inside a modular data centre. 77
5.12 Diagram for the experimental setup used to evaluate LDLTE. 78
5.13 Utility function for machines in experiment. 80
5.14 Temperature of machines while LDLTE is operating. 81

6.1 Pseudo-code for algorithm 85

6.2 Average Job Time with various Demand. 86

6.3 Power with various Demand. 87

6.4 Overall cost function for different numbers of servers at the two DCs. 88

6.5 Close-up of optimal point of overall cost function for different numbers of servers at the two DCs. 88

6.6 Number of servers at the DCs with $a_k = \frac{1}{\sqrt{k}||g(k)||_2}$. The number of servers at each data centre moves steadily from the initial conditions to the optimal point. 90

6.7 Number of servers at the DCs with $a_k = -\frac{0.1(C(n^*)-C(n))}{||g(k)||_2+1}$. 91

7.1 Pseudocode for pairwise partitioning rule 94

7.2 Peak Daily price of electricity for suppliers in the regions of the three data centres studied. 95

7.3 Price of electricity for suppliers in the regions of the three data centres studied. . 96

7.4 Daily peak carbon intensity of electricity supplier in the region of the Ireland data centre studied. 97

7.5 Carbon intensity and generated wind power of electricity supplier in the region of the Ireland data centre studied. 98

7.6 Layout of cooling cost simulations with (a) cold aisle containment and (b) no cold aisle containment. 99

7.7 Typical model of the Coefficient of Performance (COP) curve for a chilled water CRAC unit. 102

7.8 Cooling cost of various data centre cooling systems at various levels of demand . 103

7.9 Latency between California and different geographical regions at fifteen minute intervals. 104

7.10 Latency between Ireland and different geographical regions at fifteen minute intervals. 104

7.11 Latency between Virginia and different geographical regions at fifteen minute intervals. 104

7.12 Diagram of the simulation setup. The colour of the node indicates that the node is part of a particular partition. 105

7.13 Number of requests serviced at each data centre when the "Time Focused" scenario is used. It shows the UTC+0 time zone. 109

7.14 Number of requests serviced at each data centre when the “Carbon Focused” scenario is used. It shows the UTC+0 time zone. 110
7.15 Number of requests serviced at each data centre when the “Electricity Focused” scenario is used. It shows the UTC+0 time zone. 111
7.16 Carbon Emitted at each time interval under a variety of scenarios. It shows the UTC+0 time zone. 112
7.17 Electricity Cost at each time interval under a variety of scenarios. It shows the UTC+0 time zone. 113
7.18 Average service request time at each time interval under a variety of scenarios. It shows the UTC+0 time zone. 114
7.19 Total carbon output with varying relative price functions. 115
7.20 Total electricity cost with varying relative price functions. 115
7.21 Average service request time with varying relative price functions. 115

Chapter 1

Introduction

Recently the cloud computing paradigm has become an increasingly popular method for the delivery of internet services. Of particular interest is the establishment of large public clouds with data centres in different geographical locations. A cloud service which services a globally distributed user base and operates in different geographical regions has numerous advantages such as reduced latency, increased data transmission rates and improved redundancy. The design of cloud computing architectures gives rise to a number of relatively new research questions which are the subject of attention in the research community. The objective of this work is to consider and provide solutions to some of these.

In order to function economically the cloud operator must ensure that cloud users are receiving a reasonable quality of service (QoS). A basic question is how QoS can be delivered while at the same time keeping costs low. Tuning knobs which are at the disposal of the designer include: bandwidth allocation; geographical distribution of load; cooling; and many others, all of which can be adjusted to keep costs low, subject to meeting a minimum QoS requirement. Two basic tools which can aid the cloud operator to control the cloud are rate limiting and load balancing. Rate limiting is the imposition of an artificial limit on the bandwidth of a user. Rate limiting can be used to offer a fixed cost price for traffic by placing an artificial limit on the rate of traffic. Load balancing is used to determine how to distribute load across multiple resources to achieve various goals. In the following chapters we attempt to show how these tools can be used with mathematical analysis to maintain a reasonable QoS, enforce desirable pricing schemes and lower operational costs to allow the cloud to function economically.

In the first part of the thesis we consider how to enforce desirable pricing schemes for internet traffic. To do this we consider a recently proposed form of rate limiting known as distributed rate limiting (DRL) [143]. This is the imposition of an artificial limit on the bandwidth of users at distributed locations. An algorithm which uses DRL to enforce a fixed-cost traffic service must fairly distribute the fixed bandwidth to the distributed elements. It should also be resilient to failure and able to adapt to the dynamic environment of the cloud. Our work details the experimental evaluation of two DRL algorithms which are Cloud Control Constant Probabilities (C3P) and Distributed Deficit Round Robin (D2R2), the *good-neighbours* enhancement to improve performance in the event of failure and a method for augmenting the performance of the algorithms with a dynamic bandwidth limit. Rate limiting is also an important concept in the maintenance of a reasonable QoS inside the data centre. Some clouds are public environments and as such there is the potential for service interference between users, both malicious and unintentional. We evaluate the dropping mechanism used in a recent proposed rate limiting algorithm as part of a bandwidth management system to fairly distribute bandwidth to the flows of cloud users.

In the second part of the thesis we consider how physical operational costs can be lowered while sustaining a reasonable QoS. Firstly we consider thermal management inside a data centre. Data centres generate a lot of heat and this must be removed to prevent damage to equipment. The rate of equipment failure for servers increases as the temperature rises and as such most data centres attempt to maintain the server inlet temperature below a certain "redline" value to prevent unnecessary equipment failure. In this context we apply recent results of consensus algorithms to develop a distributed control algorithm for heat balancing in certain types of data centres.

In the final part of the thesis we consider electricity costs and carbon costs associated with data centres. In the context of carbon we develop a subgradient algorithm to minimise a combination of average service request time (QoS) and carbon emissions with a relative price function used to reflect the relative importance of the factors to the cloud user. In the context of electricity cost we develop a geographical load balancing tool based on Voronoi Partitioning that allocates load, so as to minimise a combination of electricity price, carbon emissions and average service request time (QoS).

1.1 Key Contributions

The most significant contributions of our work are as follows:

- We design, build, and deploy, an implementation of recently proposed DRL algorithms on a testbed. Some enhancements are also suggested.
- We extend a dropping mechanism in a DRL scheme for use in a data centre network management system and compare its performance to previously applied schemes.
- We develop and evaluate a distributed algorithm to equalise the temperature of data centre components while servicing the total demand of the data centre.
- We develop a framework which balances the trade-off between the carbon emissions and QoS in a distributed manner for application in geographically distributed data centres.
- We detail a model which considers the carbon emissions, electricity price, and time required for the computational and networking components of a service request. We then develop a distributed algorithm which minimizes the combination of average request time, electricity cost and carbon emissions. We present data for the carbon intensity and electricity price of various geographical regions and experimentally determine the round trip time between various geographical regions. We evaluate the performance of the distributed algorithm using the obtained data.

1.2 Disertation Outline

The dissertation is structured as follows.

Chapter 2 presents an overall background on cloud computing, describing the models which can be employed and the hardware of which the cloud is comprised.

Chapter 3 describes the operation of two DRL algorithms used to offer predictable, incremental, fixed cost pricing policies, as well as enhancements to ensure their performance in the dynamic environment of the cloud and in the event of hardware failure.

Chapter 4 details the use of the "fair-share" dropping mechanism for bandwidth management in data centres.

Chapter 5 reports on the use of implicit consensus algorithms to equalise the temperature of server racks inlets for thermal management purposes.

Chapter 6 describes an algorithm to minimise a cost function which represents trade-off between carbon emissions and average service request time.

Chapter 7 presents the Stratus system which uses Voronoi partitions and a pairwise partitioning rule to control a number of factors in the operation of cloud.

1.3 Publications arising from this work

- Joseph Doyle, Robert Shorten and Donal O'Mahony. "“Fair-Share” for Fair Bandwidth Allocation in Cloud Computing". *IEEE Communications Letters*, Vol. 16, No. 4, pp 550-553, 2012.

- Joseph Doyle, Florian Knorn, Donal O'Mahony, Robert Shorten. " Distributed Thermal Aware Load Balancing for Cooling of Modular Data Centres". *Accepted by IET Journal of Control Theory and Applications*, 2011.

- Joseph Doyle, Robert Shorten, Donal O'Mahony. "Stratus: Load Balancing the Cloud for Carbon Emissions Control". *Submitted to Elseiver Computer Communications*, 2012.

- Joseph Doyle, Donal O'Mahony and Robert Shorten. "Server Selection for Carbon Emission Control". *In Proceedings of ACM SIGCOMM Workshop on Green Networking*, pp 1-6, Toronto, 19 August 2011

- Joseph Doyle, Robert Shorten and Donal O'Mahony."An experimental evaluation of distributed rate limiting for cloud computing applications". *In proceedings of ACM ANCS*, pp 27:1-27:2, La Jolla, 25-26 October 2010.

Chapter 2

Background

2.1 Introduction

Cloud computing is a new paradigm for the delivery of internet services. It refers to both the hardware and software used in a data centre to provide a service and the actual program that delivers the service across the internet. One central idea to a user in cloud computing is that computational resources are offered as a utility. Resources can be used for a short period of time and then released. This reduces the need for cloud users to obtain large amounts of capital to launch internet services as computational resources are paid for on a usage basis. In addition, it allows users of the cloud to dynamically alter the level of computational resources they are using to react to the demand for the service, thereby eliminating a large portion of the financial risk associated with launching an internet service. The owners of the cloud are therefore able to provide a cheaper service and make a profit due to economies of scale and more efficient use of resources. In terms of hardware, there are three key aspects to cloud computing [9] namely:

1. *An illusion of on-demand infinite resources* . A user of the cloud should be able to have as much computational, networking and storage resources as they desire. A user should be able to access these resources on-demand so that provisioning plans are not required.

2. *The elimination of fixed cost investment in computational resources.* This allows users of the cloud to start with a small amount of resources and increase as their requirements grow.

3. *Resources should be available on a short-term basis (e.g. processors by the hour and*

storage by the day). Users of the cloud can then use and release the resources as needed allowing them to save on operational costs. This aspect rewards conservation.

The control of a cloud is challenging task as the cloud owner must adhere to the three key aspects while also considering other important factors such as latency, fairness among cloud users, operating costs and the carbon emissions associated with the cloud. When attempting to achieve these goals an understanding of the limitations of the hardware and the tools that can be used to control the cloud are necessary.

In the next sections we further elaborate on the operation of the cloud, the hardware involved, the load balancing and the rate limiting tools used to control the cloud.

In Section 2.2 we discuss the standard model of cloud computing. We then expand upon this to discuss the varieties of cloud computing that can be found and the manner in which their models work.

We describe the most common network architecture used in data centres and the problems that are associated with it in section 2.3. We will then examine proposals which have been published to tackle the problems associated with traditional data centre architecture and routing protocols. Cooling is a significant portion of the operational cost of a data centre. Thus we discuss the various types of cooling hardware and some of the proposals that have been made to lower cooling costs. We then examine the manner in which cloud users can impinge upon each other and the security techniques that are required to ensure that cloud users receive a reasonable Quality of Service (QoS). Finally, we describe the various costs of constructing and running a data centre so that importance of lowering the operational cost is highlighted.

We discuss the mathematical techniques that our work uses to improve the performance of the data centre in Section 2.4. Firstly, we detail consensus algorithms and describe some of the other applications of these techniques. We then examine Voronoi partitions and discuss some of the other areas in which this technique has been used.

In Section 2.5 we describe the current techniques that are used in load balancing for this application. We then detail some of the proposals which have been made to improve the performance of the cloud using load balancing.

We examine the reasons that rate limiting is utilised and some of the techniques used in rate limiting in section 2.6. We then discuss DRL and various algorithms which are used in DRL.

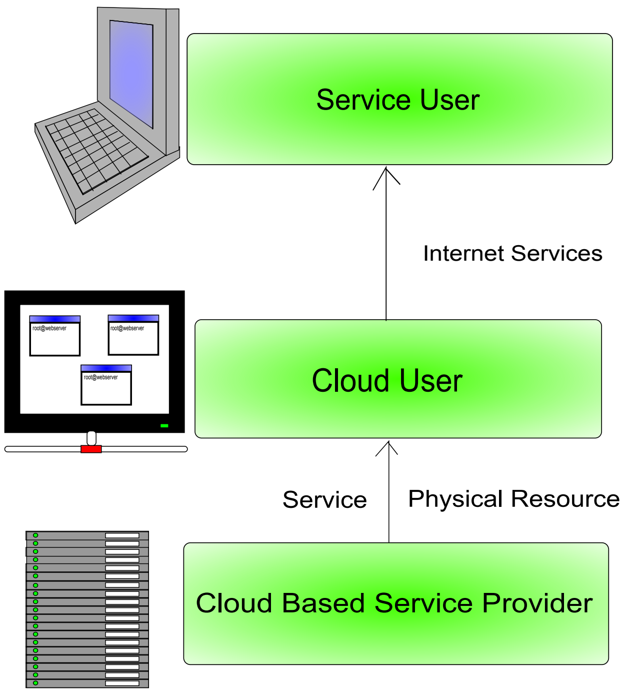


Figure 2.1: Illustration of a parties involved in public cloud computing and the products and services that the parties provide.

2.2 Cloud Computing

2.2.1 Models

The first thing to consider in the cloud computing paradigm is whether the cloud is public or private. A public cloud is one where the resource are sold to the general public on a usage basis. A private cloud is the internal data centre of an organisation or business. In this work we will deal primarily with public clouds but some of the techniques discussed could equally be applied to private clouds. In addition, the barriers between public and private clouds are blurring as some organisations are beginning to use hybrid clouds which consist of at least one private cloud and at least one public cloud. Usually there are three parties involved in the public cloud: namely, the cloud based service provider (CBSP), the cloud user and the service user. The standard cloud service works as follows. A CBSP furnishes the cloud user with the software and hardware so the cloud user can provide the service user with a particular service. This is illustrated in Figure 2.1.

There are a number of cloud computing models. Three of the most popular are: (1)

Software as a Service (SaaS), (2) Platform as a Service (PaaS) and (3) Infrastructure as a Service (IaaS). In SaaS a service is hosted on the cloud and it is accessed through an interface, usually delivered via a web browser. This reduces the complexity of software installation, upgrades, maintenance, and of patching, as all of these tasks can be handled at a central location with the overall goal of reducing the total cost of ownership (TCO).

In PaaS the CBSP offers a software platform on which cloud users can build services [140]. Google's commercial offering (known as "AppEngine" [54]) is an example of PaaS. It is designed as an engine to execute web services, and as such, the specifics of the hardware are abstracted from the cloud user. Services must be written in specific languages, but the scaling of the application is hidden from the user and dealt with by Google. Microsoft's commercial offering known as Azure [97] is another PaaS service which can be used to offer web services. This product offers more freedom when compared with Google's "AppEngine". The application can be written in any language but it must use the Microsoft .NET libraries. The use of a specific language, or application program interface (API), makes it difficult to migrate services into the cloud. PaaS, however, allows a greater degree of control than SaaS for the cloud user.

In IaaS the CBSP offers direct access to a Virtual Machine (VM) that is very similar to physical hardware [140]. Amazon's Elastic Compute Cloud (EC2) [6] is an example of an IaaS service. This provides a lot of control to the cloud user at the cost of increased complexity. For instance an additional tool is required for automatic scaling. This means that without the use of additional software, a service hosted on EC2 cannot quickly alter the level of resources (servers, storage etc) to react to a change in the demand. There is a potential problem that a service could be under-provisioned, or over-provisioned.

2.3 Hardware

One of the focal points of this work is the reduction of operational costs and associated carbon emissions. The specifics of electricity costs and carbon emissions are very dependant on the hardware used in the cloud. In this section we will examine the various aspects of cloud hardware that affect the operation of the cloud. We will examine current data centre architectures. We also examine the limitations of conventional architectures and some new proposed cloud architectures. Secondly, we will consider the cooling hardware used in data centres. Understanding the hardware is necessary to prevent excessive cooling and thereby lower operational

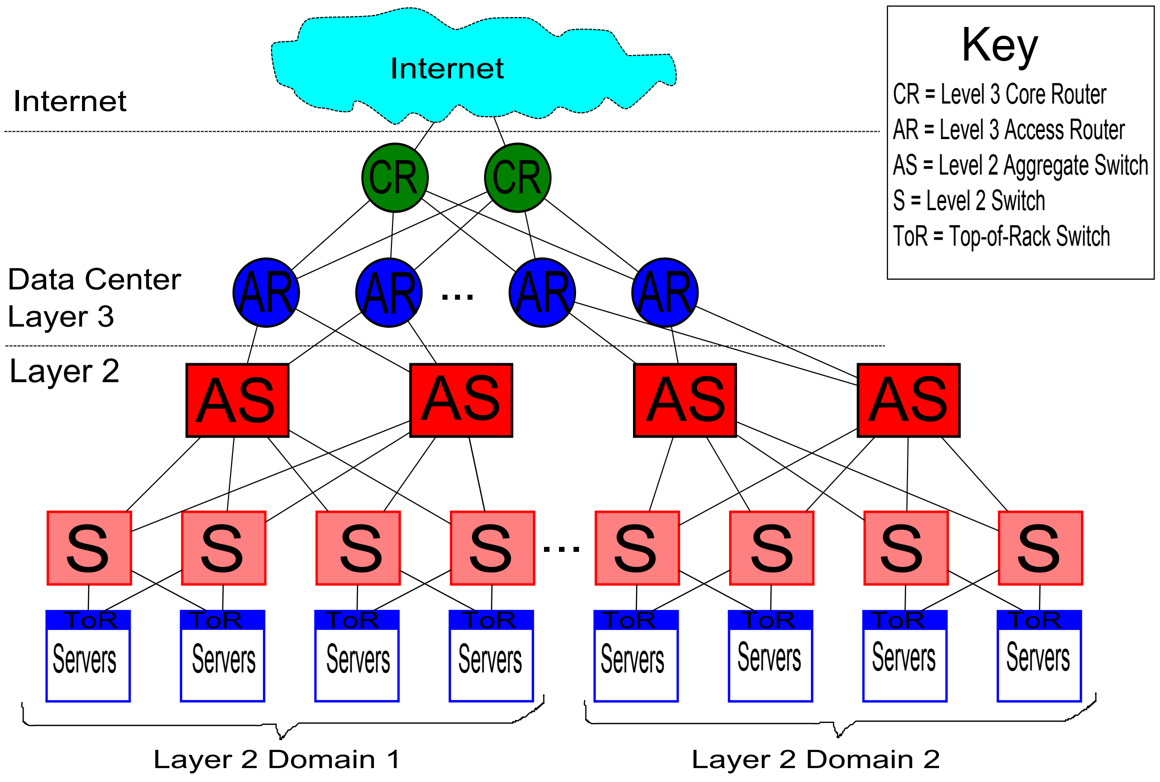


Figure 2.2: A conventional data centre network architecture.

costs. Thirdly, we examine schemes currently used for bandwidth management as bad management of resources can lower QoS. This could discourage utilisation of the cloud and make it difficult for cloud operators to recover capital costs.

2.3.1 Data Centre Architecture

In order to provide cloud services the CBSP creates facilities where huge numbers of computers are concentrated in a single geographical location. There are data centres currently being built with 400,000 to 500,000 physical servers in a single facility [100]. The size of these facilities strains the conventional data centre network (DCN) architecture. An examination of data centre network topology as well as current Routing and Data Link protocols; namely IP, and Ethernet respectively, shows that several performance problems emerge as the number of servers in the DCN increases.

The first of these problem is that the most commonly used topology of current DCNs is such that the interconnective bandwidth between servers is insufficient. This is extremely important in MapReduce applications [36]. This results in under-used servers in some parts of the DCN. The conventional topology consists of a tree like network illustrated in Figure 2.2 with the highest parts of the tree requiring high cost hardware. The data centre network consists of racks with 20 to 40 servers each connected to a Top of Rack (ToR) switch. ToRs are then connected to aggregation switches. They are usually connected to more than one for redundancy purposes. The aggregation switches then connect to access routers. The final connection is between the access routers and the core routers which control traffic in and out of the data centre and route traffic between access routers so that all-to-all communication is possible. In all-to-all communication, such as traffic which occurs in MapReduce applications [36], each server must communicate with every other server in the DCN. When there is insufficient bandwidth oversubscription occurs. The term oversubscription is used to describe situation when the desired level of traffic across a link is greater than the level that can be physically sent across the link. There is no oversubscription in all-to-all communication between servers in the same rack [58] as there are no bottlenecks for traffic. There can, however, be oversubscription of up to 1:20 in all-to-all communication between servers using the same aggregate switch on the links between the ToR switch and the aggregate switch [58]. The level of oversubscription increases as the traffic moves up the hierarchy where it can reach levels of 1:240 at the paths connecting to the core routers [58]. This would mean that a server is only obtaining $\frac{1}{240}$ of its desired level of traffic.

Another problem is that Ethernet will not function properly in today's very large DCNs for the following reasons:

- Ethernet bridges learn about the network topology by establishing entries for medium access control (MAC) addresses in forwarding tables. These tables can grow very large. The number of entries in a table is proportional to the number of hosts in the DCN as flat addressing is used. By this we mean that each individual address must be stored in the table with an associated port rather than a range of addresses being associated with a port. This can result in the network bridges running out of memory and dropping hosts from the tables. Missing entries give rise to unnecessary broadcasts as the address resolution protocol (ARP) must be used to determine which port should be utilised.

- Ethernet creates a spanning tree between two network nodes to prevent loops in the network [122]. This works well for small networks but introduces latency and availability problems in large networks. The spanning tree protocol does not consider the bandwidth available on links when selecting a path from a source to a destination. As a result some paths might be underused and thus bandwidth inefficiencies are introduced. Equal-cost Multi-path (ECMP) can be used to alleviate these but there can be significant problems in its deployment in practice [151].

- Other protocols associated with Ethernet such as the ARP and the Dynamic Host Configuration Protocol (DHCP) use broadcasting which cause significant traffic in networks of this size [76].

These problems can be alleviated somewhat by dividing the DCN into IP subnets and thus achieving scale by assigning each server an IP addresses based on this logical topology. The DCN is divided into smaller networks that are connected together to avoid the problem of scale. VMs which are software implementations of physical machines are now being used in some data centres. The use of virtualisation technology creates a problem with the subnet division approach as virtual machines cannot migrate out of their original IP subnet while maintaining the same IP address. VM Migration, which is the movement of a VM to a different physical machine, can be viewed as a form of load balancing and consequently the subnet division approach is somewhat limited as it constrains how load can be dynamically shared. Further, the setup and maintenance of the subnets can be a long and laborious task. This is sometimes referred to as the “Ethernet scaling” problem as the performance of Ethernet will not scale to the number of hosts required in a large DCN.

Thirdly, there is nothing to prevent one service inflicting damage on other services which share the same resources in a DCN. Users can attempt to consume all the bandwidth on shared resources and current data centre architectures do not have a mechanism to prevent this [58]. This is referred to as the “Traffic flood” problem.

Lastly, the cost of the most oversubscribed parts of the DCN tree accounts for a large portion of the cost of the DCN networking equipment. The reason for the high cost is that these switches use highly specialised application specific integrated circuit (ASIC) hardware with high development costs.

There have been a number of proposals to counteract the problems associated with conventional data centre architecture and routing protocols [3, 58, 63, 62, 76, 107, 59, 48] and it is likely that some of these will be, or have been, adopted in the industry. An understanding of these new architectures and associated protocols is advantageous in large data centre management as any new control schemes implemented will have to adapt to these new architectures and protocols to be useful in the near future.

2.3.2 Cooling

Cooling is crucial to the operation of a data centre and can represent a significant operational cost to the cloud operator. Insufficient cooling can result in equipment failure which will lead to further expense for the cloud operators. It has been reported that every 10 °C increase above 21 °C decreases the long-term reliability of electronics by 50% [148, 120]. In addition, it has been shown that a 15 °C rise increases the failure rates of hard disk drives by a factor of two [7], although recent studies suggest that the increase in the rate of failure may not be as severe as this [42]. To prevent equipment damage, the inlets of all servers in the data centre must be maintained in a safe operating range below the "redline" value of approximately 25 °C. There is considerable debate over what exactly this "redline" value should be [11] as some proponents suggest it should be considerably higher than current guidelines. The fact remains that there is an upper limit to the safe operating range for the temperature of inlets to servers.

Cooling systems take a number of forms. The most popular is the use of computer room air conditioning (CRAC) units and a raised floor plenum. In this system cold air is blown into the plenum by CRAC units pressurising the plenum. Cold air exits through perforated tiles that are placed in front of server racks. The cold air then flows through the server cooling the components. As the cold air passes through the server it is warmed. This warm air exits the back of the server where it is recirculated back to the intakes of the CRAC unit to cool the air. The cycle then repeats. This is illustrated in Figure 2.3. Each CRAC unit consists of a number of coils through which liquid coolant is pushed and a fan which pushes air through the coils cooling the air. A pump circulates the coolant to a chiller which removes the heat from the coolant so that the supply temperature of the CRAC unit is maintained at a constant level.

In this setup some of the hot air generated from the server outlets can mix with the cold air from the CRAC units which causes complex air flow patterns leading to hotspots. This is problematic as it requires the entire data centre be cooled sufficiently to prevent equipment damage at the hotspots. This leads to a large portion of the data centre being over-cooled. A variation of the previous system known as aisle containment is used to alleviate some of these problems [121, 99]. In aisle containment the cold or hot aisles are segregated from the rest of

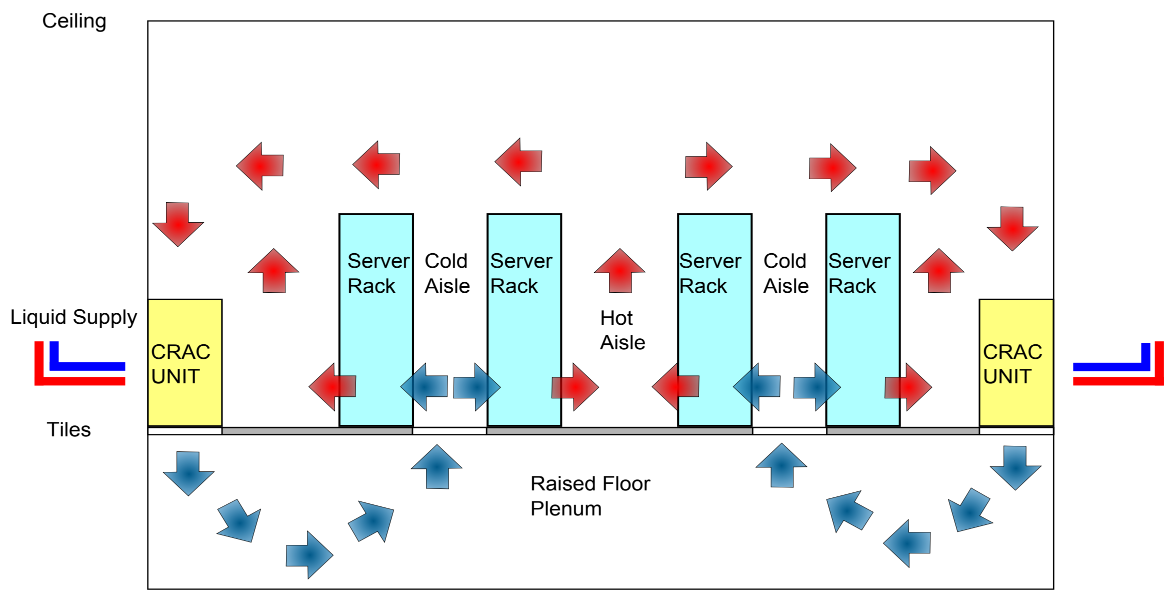


Figure 2.3: Diagram of raised floor plenum CRAC unit cooling.

the data centre which prevents the airflows from mixing and consequently prevents hot spots. The containment is constructed using physical barriers such as PVC curtains and Plexiglas [17]. There are two main variations of aisle containment namely “hot aisle containment” and “cold aisle containment”. There is evidence that “hot aisle containment” may be a more efficient design but it is recognised [121] that there are difficulties in retrofitting this solution which may make “cold aisle containment” the more popular design in the short-term. In the models we use in Chapters 5 and 7, we examine how the variation in cooling systems affect the cooling costs.

Another variation is the use of air economizers instead of CRAC units [12, 17] or “Free Air Cooling”. Air economizers draw in cool air from outside the data centre and expel hot air from the servers back into the environment. It is also possible to use “Water-based Free Cooling”. In this case the coolant supplied to the CRAC units also runs through a heat exchanger where water absorbs heat from the coolant. Both of these methods depend on the ambient temperature being sufficiently cold so that the supply temperature of the air entering the inlet of the servers is below the “redline” value. In addition “Water-based Free Cooling” is difficult to use in very cold climates as additional steps must be taken to prevent the formation of ice on the storage towers. “Free Air Cooling” on has been shown to function for extended periods of time at extremes of $-20\ ^\circ\mathrm{C}$ [98]. A major disadvantage of “Free Cooling” is that it must be supported by CRAC units and chillers if there is a chance that the ambient temperature will exceed the “redline” value as the data centre cannot operate without cooling for more than a few minutes. While it is possible to redirect load to another data centre and shutdown the data centre during peaks of ambient temperature this is a challenging prospect for most cloud operators.

Liquid based cooling [130, 5] can also be used to provide cooling for data centres. In this method a heat exchanger is typically attached to the back of the rack so that hot air from the server outlet flows over coils that are cooled by water. Essentially this method involves placing a small heat exchanger at the back of each server rack. This form of cooling can be used as the only source of cooling or in conjunction with the larger CRACs where only some of the heat is removed by the liquid based cooling solution. In the former case it must completely remove all the heat from the server outlets and thereby replace the large CRACs. In addition, it is also possible to liquid cool the Central Processing Unit (CPU) directly and greater efficiencies can be achieved through this method [130]. All forms of liquid cooling increase the cost of plumbing as coolant must be brought to each rack and cause concern over the risk of leaks.

In additional to hardware, software can also have a large effect on cooling costs. There has been some recent research which quantify this effect and we will examine their work in further detail here. Sharma *et al.* [134] presents an algorithm to lower the difference between the outlet temperature of the server racks to prevent hot spots. This was done by adjusting the power that the server rack consumes (by adjusting the number of requests the server services) so that it is inversely proportional to the difference between the exhaust temperature and a reference temperature (i.e. the supply temperature of the air). This techniques results in server racks with hot exhausts being given less work which will cool the exhaust of these servers and prevent hot spots.

Moore *et al.* [103] use a calibration phase to determine which server racks cause the most recirculation. In this calibration phase the power consumed by the servers and cooling equipment is recorded. The power for a pod (a group of servers) is adjusted and the power consumed by the servers and cooling equipment is recorded. This is repeated for each server rack. A Heat Recirculation Factor (HRF) is then calculated for each server. The HRF is then used to determine where to assign load to minimise the cooling costs. Moore *et al.* [102] also propose a method for determining the temperature of a server inlet without a sensor to allow this software to function without further instrumentation.

Parolini *et al.* [116] examines controlling the supply temperature of the CRAC units, the power state of the servers and assignment of jobs to servers as a Markov decision process (MDP) with the goal of minimising the integrated weighted sum of the power consumption and computational performance . The power states of the servers can be adjusted using the TEAPC [155] system. Parolini *et al.* also consider the problem of excessive cooling as a cyber physical problem [117]. These problems consider cyber resources (the computational resources) and physical resources (the cooling resources) which are joined via a network model. A control strategy to lower cooling costs while considering load distribution is also presented.

Weissel *et al.* propose a system which determines the temperature of servers using information from event monitors which are embedded in modern processors and throttles requests to meet the thermal requirements of the system [161]. Das *et al.* consider controlling the fan speed of CRAC units in order to control a utility function which incorporates the energy consumed and the temperature of the inlets of the rack servers [35]. There are also a number of tools [28, 67, 34] which have been proposed to simulate the performance of these algorithms and examine how the layout of a room affects the temperature of the inlets of servers. Heath *et al.* propose the Freon tool to prevents equipment damage by shutting off hot servers in the event that they cross the "redline" value [67]. Faraz *et al.* [47] considers a combination of the cooling power for CRAC units and the idle and dynamic power of servers in a load balancing scheme to lower overall energy costs.

2.3.3 Bandwidth Management

A public cloud is an environment where resources are shared by multiple users. If these resources are not carefully managed, service interference (both malicious and unintentional) can occur between cloud users. Service interference is already affecting cloud users on current clouds. The throughput of medium *instances* on Amazon's EC2 can vary by 66% [87, 159] and it has been postulated, based on anecdotal evidence, that the cause of this is a lack of bandwidth management algorithms between users [136].

There are numerous techniques for the control and management of computational, memory and disk resources [27, 61]. It is, however, generally accepted that current techniques for network management cannot prevent "service interference". We define "service interference" as the actions of one user on the cloud negatively affecting another user. One of the most widely deployed network management techniques is Transmission Control Protocol (TCP) congestion control. It is largely responsible for the "fair" allocation of network resource in addition to its other functions. It manages the bandwidth of all flows passing through networking equipment. A malicious user can circumvent this to prevent the "fair" allocation of network resources and cause "service interference" by using non-compliant protocol implementations or by opening more flows than normal. For example, attacks could be carried out that are similar to Denial of Service (DoS) by malicious users. By seeking out a server on the same switch as the server they wish to attack, flooding the link with traffic thereby blocking traffic from other sources, malicious users can carry out attacks on victim's servers. This attack is difficult to detect as no traffic is sent to the victim's server.

The utilisation of non-compliant protocol implementations can be defined as manipulation attacks [80]. The aim of these attacks is to improve the performance experienced by a malicious user to the detriment of honest users. There are a variety of manipulation attacks which can be employed. Three of these attacks are described by Savage *et al.* [131]. The first type is ACK division. In this attack the receiver of a data segment of N bytes divides the resulting acknowledgements into M acknowledgements, where $M \leq N$. Each acknowledgement is a distinct piece of the received data segment. The congestion window increases in size upon receipt of an acknowledgement and ACK division results in the sender growing their congestion window at a rate that is M times faster than usual and the congestion window of honest users is smaller as packet loss occurs faster than normal. M can be set to an arbitrary level by the receiver up to one acknowledgement per byte received ($M = N$) allowing the congestion window to grow extremely quickly.

The second type of attack is DupACK spoofing. Some TCP congestion control algorithms such as TCP Reno [81] use *fast retransmit*. Fast retransmit is used to retransmit the missing segment only if loss is detected. Thus, everything from the last correctly received packet does not have to be retransmitted. When a TCP receiver acknowledges a packet it sends the sequence number of the package it expects to receive. If it receives a packet with a different sequence number it sends a duplicate acknowledgement requesting the correct packet again. Loss is detected by observing three duplicate acknowledgements. Fast retransmit operates by controlling the congestion windows. Upon detection of loss the control window is set to the threshold of the congestion avoidance stage plus three times the maximum segment size. Fast retransmit then increases the congestion window by the maximum segment size for each additional acknowledgement packet received. The DupACK spoofing attack exploits this by sending a long stream of acknowledgements for the last sequence number received upon receipt of a data segment thereby artificially increasing the congestion window. This attack allows the receiver to dictate the speed at which the sender transmits as the rate at which the receiver sends acknowledgements is direct proportional to this.

The final type of attack described by Savage *et al.* [131] is optimistic ACKing. In this attack the receiver acknowledges data it has not yet received. Normally the growth of the TCP congestion window is dictated by the round trip between the sender and the receiver. The receiver emulates a shorter round-trip time and increases the data transfer rate by optimistically acknowledging data it has not received. This attack allows the receiver to increase the congestion window to the bottleneck rate immediately and hold it there in spite of losses. It can then use other application layer retransmissions available in the HTTP-1.1 protocol [50] to obtain lost segments. These attacks are detrimental to honest users as the sum of the rates can be no greater than the limits of the physical medium and if one flow rate is disproportionately large, segments from other flows will be dropped earlier and their transfer rate will be less than it should be.

There have been a number of proposals to prevent manipulation attacks which we will discuss here. One of these by Shieh *et al.* at Microsoft Research is Seawall [136]. In this system local entities (VMs, processes, etc) are assigned a network weight and each local entity receives a share of the bandwidth proportional to its network weight along all network links. The network weight functions as a metric for the priority rating of the traffic of the local entity. A local entity with a higher network weight will obtain a greater bandwidth. The Seawall system uses a rate limiting tool known as token bucket to ensure that each local entity obtains a fair allocation of bandwidth.

Another proposal is Gatekeeper [128] by Rodrigues *et al.*. It is similar to Seawall but in Gatekeeper the bandwidth of a link is divided among the tenants of the link rather than among local entities which wish to send traffic across the link. We define the tenants of a switch as the servers which are connected to the switch by a single link. A local entity can be anywhere in the data centre while a tenant must be connected to a switch by a single link. Gatekeeper divides bandwidth among tenants rather than local entities so that it prevents users with a large number of VMs in the cloud from obtaining an unfair proportion of the bandwidth by having a large number of servers send data to a specific tenant on a switch.

Wilson *et al.* proposed the D^3 system [166]. In this proposal deadlines are assigned to some flows. For example an online service which has a specified latency target in the service level agreement (SLA) would be given a deadline. The rate required to reach the deadline is calculated by the sender and this is sent with the packets. A greedy algorithm is used by switches to assign rates to the flows. The router will also divide any remaining bandwidth fairly among the flows. In the event that the router does not have enough bandwidth available to meet a deadline it assigns all that remains to the request when the next flow is received.

Ghowdhury *et al.* proposed a system called Orchestra [29]. In this, the operation of MapReduce [36] and similar systems is improved by considering the length of data transfer and assigning more TCP connections to longer data transfers. In doing this all transfers complete simultaneously. Another proposal is Oktopus by Ballani *et al.* [16]. In this system a greedy algorithm is used to attempt to find the smallest number of links required to form a virtual network of N nodes with bandwidth B connecting all the nodes in a tree structure. Kothari *et al.* proposed a tool which examines a protocol implementation to check if it is vulnerable to manipulation attacks [80].

There are a number of ways non-compliant protocols can be used to exploit the underlying hardware and obtain an unfair share of bandwidth. There have also been a number of proposals to prevent this. Each of these proposals is designed to prevent a specific type of "service interference" and each has its advantages. For example Orchestra is particularly useful if MapReduce type applications are the focus of the cloud while it is not as strong if content delivery is the focus. Some of these programs will use dropping mechanisms to ensure a fair share of bandwidth is received by users. It is important that the dropping mechanism used also ensure that the fair share of bandwidth is divided reasonably among the flows that use this share. In Chapter 4 we will propose a mechanism which builds upon the Seawall system [136] by ensuring fairness between the flows of the local entities and lowering the overhead cost associated with the system.

2.3.4 Data Centre Costs

Throughout this work we propose algorithms which lower the operational costs and increase the utilisation of the cloud. In this section the breakdown of the cost of the cloud will be examined. We will illustrate the importance of optimising the operational cost of the cloud so owners can recover their capital investment. Greenberg *et al.* present such a breakdown [57]. The first interesting thing to note about this breakdown is that operational staff costs are so low ($< 5\%$) and can omitted from the calculation. This is unusual as they are usually the leading cost in enterprises [57]. The breakdown assumes that the cloud consists of a data centre with 50,000 servers. Each server costs \$3,000. At a 5% cost of money and a 3 year amortisation regime servers have an annual cost of \$52.5 million dollars. The data centre requires facilities for power delivery and evacuating heat. This requires investment in large scale generators, transformers, uninterruptible power supply (UPS) systems and cooling equipment. The breakdown assumes a cost of \$200 million for these facilities. At a 5% cost of money and a 15 year amortisation regime the facilities have an annual cost of \$18.4 million. The capital cost of networking is significant as a large number of switches, routers and load balancers need to be purchased for the data centre network to function effectively. In addition to the capital expenditure, there are additional networking costs such as peering where traffic is handed off to internet service providers (ISPs), inter-data centre links which carry traffic between geographically distributed data centres and regional facilities needed to reach wide area network interconnection sites. These costs are difficult to calculate as they vary from site to site and with time, but we can assume an annual networking cost of approximately \$9.3 million [57]. This includes the amortised cost of the capital equipment such as switches, routers and load balancers.

Finally a cloud needs power to operate. A metric to describe the total power required for the facility is the relation between the power required for the faculty and power required for the IT equipment is provided by the Green Grid [56]. Power Usage Efficiency (PUE) is calculated as (Total Facility Power)/(IT Equipment Usage). The PUE of inefficient enterprise facilities ranges from 2.0 to 3.0 [152]. A state of the art facility typically attains a PUE of ∼1.7 and leading facilities such as Facebook data centres can obtain a PUE of 1.08 as of 2012 [44]. To estimate the cost of power the breakdown we assume a PUE of 1.7, a reasonable electricity price of \$.07/kWh and that each server draws an average of 180 W. This results in an annual electricity cost of \$9.3 million which coincidentally is the same as the network cost.

The percentage breakdown of the cost of the cloud is depicted in Table 2.1. From this breakdown we can establish that the greatest cost is the capital cost of the servers. This illustrates the need to encourage utilisation as substantial revenue must be obtained to recover the capital investment. There are high capital costs that need to be covered and running costs are relatively low as servers have a high idle power which is between 60% and 75% of the peak power [25, 45]. Owners are therefore motivated to keep the system running in order to generate revenue.

Component	Annual Cost over Lifetime
Servers	$\sim$45%
Infrastructure	$\sim$25%
Network	$\sim$15%
Power	$\sim$15%

Table 2.1: Percentage breakdown of cost of cloud [57].

2.4 Applicable Theory

In this section we discuss the mathematical techniques we will use to control the operation of the cloud. In the Section 2.4.1 we examine various consensus algorithm and how they are used. In Section 2.4.2 we examine Voronoi partitions and the applications of this technique.

2.4.1 Consensus Algorithms

We use two types of consensus algorithms and the particulars of the systems are described in Chapters 3 and 5. In consensus algorithms a group of distributed nodes attempt to reach a common goal by exchanging information with their neighbouring nodes and adjusting their state according to an algorithm. This can be represented as a graph where edges exist between nodes if the nodes exchange information. Consensus and agreement algorithms were first examined in the context of management science and statistics [41, 108, 167, 37]. While they have been applied in a number of disciplines such as the fusion of sensor data [92, 20, 43, 114], medicine [162], decentralised estimation [85, 106, 111], simulation of flocking behaviour [127, 158] and clock synchronisation [132, 24], it is the application of these ideas to dynamic networked systems that is of primary interest to us.

The initial work in this area [22, 154, 127, 158] had a variety of goals and focused on a system in which bi-directional information exchange between neighbouring nodes occured (leading to undirected communication graphs). Rigourous convergence proofs for such systems were given in [71]. This work was then extended to consider variations on these problems which include whether the topology of the graph representing node communication remains fixed or changes over time, whether there are delays in the information exchange and if all nodes update in a synchronous fashion or if nodes update at their own pace. Other variation include if the graph is directed or undirected, if the node's state is scalar or multidimensional, whether the node can manipulate the state directly or only implicitly and if the node can manipulate the state on which to reach consensus instantly or only within certain dynamics. A system with directed communication graphs is examined in [19, 113, 104, 126, 46]. The effect of asynchronous updates, where each node does not have to update its state at the same time, is examined in [113, 66, 21, 46, 23]. A changing graph topology is considered in [71, 149, 19, 126, 110]. Generalisations of the problem which allow the inclusion of node dynamics in the consensus problem are investigated in [150, 149, 112, 110]. A system where the state of a node can only be changed implicitly is examined in [143, 144]. In Chapters 3 and 5 we examine the use of such algorithms in the context of controlling the resources of the cloud.

In addition to this work, which focuses on unconstrained consensus applications, there has been much research in applications of consensus to systems which should fulfil external conditions. By this it is meant that the goal the distributed nodes are attempting to achieve is constrained in some way. An example of such a system can be found in behaviour based animation. A group of nodes (birds in a flock) should have a common heading and the heading must also be in a particular direction. There are three approaches which are usually taken in such a system namely, leader-following [160, 96, 137, 49, 74], virtual structure based [86, 18, 135] and behaviour based [14, 8, 82, 115, 26, 157]. In the leader-following approach one of the nodes is designated the "leader" node and other nodes adjust their state to follow this "leader" node. In the virtual structure based method the graph of nodes is treated as a single structure and the desired behaviour is assigned to the virtual structure relative to each node which controls its own behaviour. In the behaviour approach the node's behaviour is based on a combination (e.g. weighted sum) of desired behaviours. In Chapter 5 we investigate the application of a behaviour based consensus algorithm [77] to the thermal management of the data centres that contain cloud resources. Consensus algorithms can be used to represent a large variety of systems and the related work described in this section allows us to analyse the various systems described in Chapters 3 and 5.

2.4.2 Voronoi Paritions

Voronoi partitions are the decomposition of a set of points into subsets. These subsets are centred around points known as sites, generators or seeds. Each point in the set is added to a subset consisting of a site and all other points associated with this site. An abstract notion of distance between a point and the sites is used to determine with which subset a point is associated. A point is assigned to a subset if the distance to the site is less than or equal to the distance to the other sites. In our work the set of points consist of sources of requests

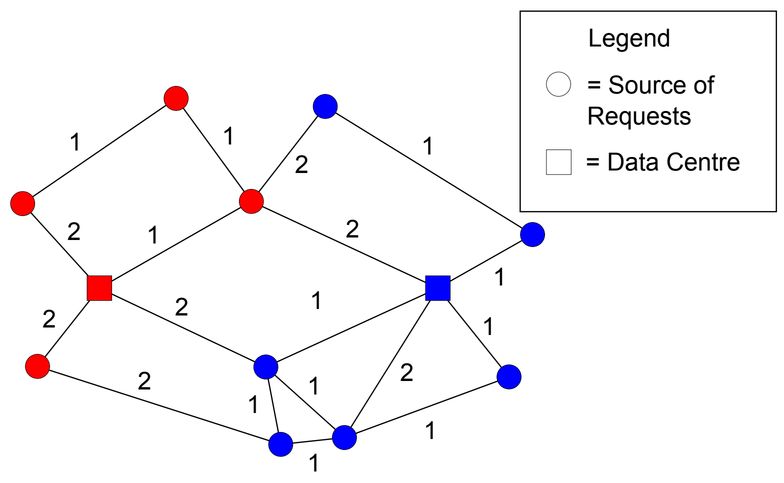


Figure 2.4: Example of how sources of requests are partitioned between two data centres. Colour indicates that the node is part of a particular partition and numbers on the edges are weights.

for cloud services and data centres which service these. Voronoi partitions are then used to determine where requests are serviced. A subset represents which sources of requests a data centre is servicing at a given time. An example of a group of sources of requests which have been partitioned between two data centres can be seen in Figure 2.4. In this figure each source of requests has a path to both data centres. The partition that the source of requests is a part of, depends on the paths to the two data centres. The partitions are made up of sources of requests, which have paths available to them with lower distances than the paths available to the other data centre. Voronoi partitions also known as Voronoi Diagrams or Dirichlet tessellations, are used in a variety of areas. While there was some early work in Voronoi diagrams by Gauss [13] and Dirichlet [13] which used quadratic forms, the generalization to higher dimensions was provided by Voronoi [13] in 1908. The earliest application of Voronoi partitions was in the fields of crystallography and more specifically geometric crystallography [13, 129]. It has also been applied to metallurgy [165, 51], precipitation estimation [153, 69], urban planning [13, 138], cartography [10] and recently robotics [39] (other applications are discussed in [13]).

Voronoi diagrams also have a number of applications in the field of computer science. Shamos [133] proposed applying Voronoi diagrams to the associative file searching problem (or post-office problem) [78]. The goal in the problem, given a subset of points, is to find the closest of these to a given query point. This is analogous to determining which partition the query point belongs to if Voronoi diagrams are used. Another application is authomatic data clustering which dividing the data into subsets which have similar in-class members and dissimilar cross-class member [65]. Voronoi partitions can also be used in scheduling record access to retrieve a batch of records in the minimum time. An exact solution has been shown to be NP-complete but a satisfactory approximation can be obtained quickly using Voronoi diagrams [83]. In Chapter 7 we use Voronoi partitions to control various aspects of the operation of the cloud.

2.5 Load Balancing

The final area in this chapter concerns the tools which cloud operators can use to control the operation of the cloud. The goal of these sections is to examine the state-of-the-art in control technology so that it can be compared with our work. Load balancing is the distribution of a workload among multiple resources to achieve various goals. It is a critical tool in the operation of the cloud and our work in Chapters 5, 6 and 7 are forms of load balancing. In this section we examine the current schemes available to a commercial load balancer and recent proposals which have been made to use load balancing to achieve new goals such as lower electricity cost or carbon emissions.

2.5.1 Current Methods

We will now examine the capabilities of a popular commercial load balancer known as the Citrix NetScaler [31] to investigate the options available to a state-of-the-art load balancing system. This service is capable of balancing load at the local level, where it assigns connections to servers in a data centre, and also at the global level, where it assigns connections to data centres in the cloud. Firstly, we examine the overall operation of the NetScaler. This is depicted in Figure 2.5. Clients request services which generate Domain Name System (DNS) requests if their local DNS does not contain a record of the service. The NetScaler can function as a DNS server and responds to the request. The response is controlled by the global load balancing service which exchanges metrics with other NetScalers via the proprietary metrics exchange protocol (MEP). A request is then generated by the client and sent to the assigned data centre. This request is sometimes recorded by the global load balancing service so that metrics are updated and is forwarded to the local load balancing where it is recorded and sent to a server. Monitor programs sometimes examine various aspects of the server and send this information to the local load balancing service so that metrics are updated.

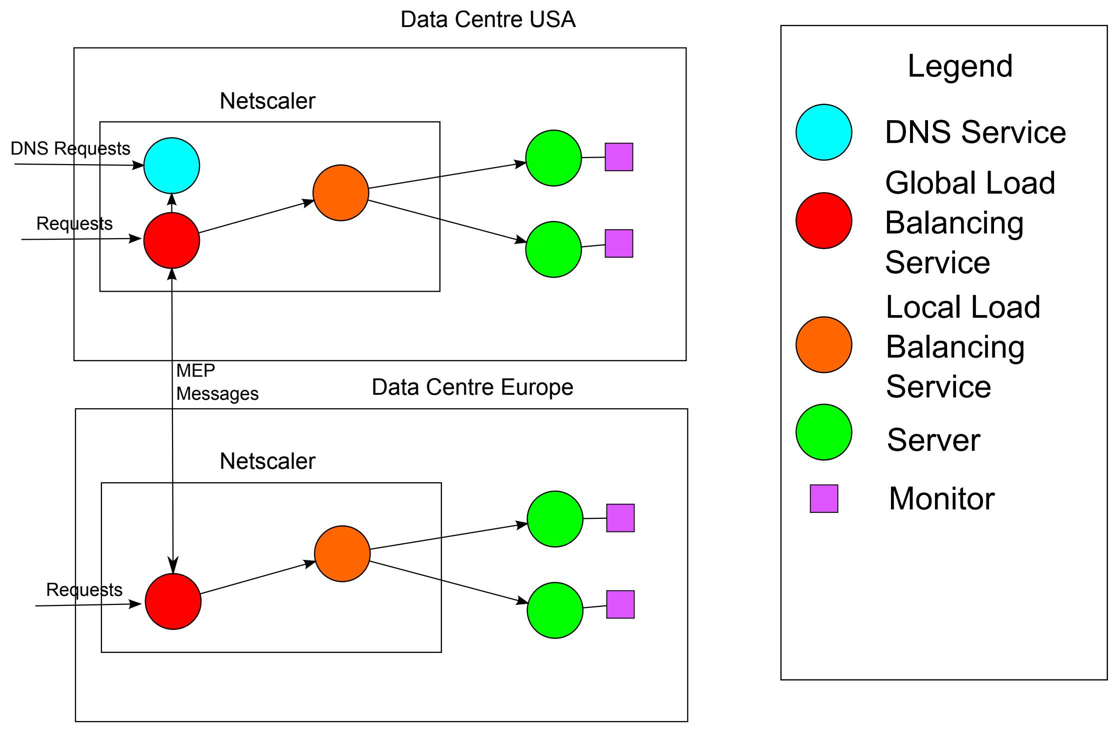


Figure 2.5: Example of how NetScaler functions in a cloud with two data centres.

Next we examine the load balancing mechanisms which are available at the local level.

Least Connection Method: In this method connections are sent to the servers with the smallest numbers of active connections. The NetScaler maintains a record of the number of active connections at each server and alters this as connections are opened or closed. This method can be modified with weights so that more powerful servers receive more connections.

Round Robin Method: This is the simplest method of load balancing and it is frequently used by other load balancing methods if the metric used is tied between two servers or when the NetScaler is starting operation. Servers are ordered in a list and a pointer which starts at the top of the list is used to determine which server an incoming connection will be sent to. After the incoming connection is sent the pointer is moved to the next server in the list and the process continues. If the pointer reaches the end of the list, it returns to the start of the list. This method can also be modified with weights.

Least Response Time Method: This method uses a combination of the response time and the number of active connections to determine where load is sent. A number of methods can be used to establish the response time. The simplest is to use the Time to First Byte (TTFB) as the response time. TTFB is the time between sending a request packet to a server and receiving a response packet from the server. It is also possible to use monitors to establish the response time. Monitors are programs which run on the servers and periodically report to the NetScaler. Monitors can examine a variety of things such as the time difference between the Internet Control Message Protocol (ICMP) ECHO request and the ICMP ECHO response in PING or the time difference between the SYN request and the SYN+ACK response in TCP to calculate the response time. Once the response time is established, a value N is calculated by multiplying the number of active connections by the response time and incoming connections are sent to the server which has the lowest N value. This method can also be modified with weights.

Hash Methods: The NetScaler can utilize a number of Hash methods [31] such as URL hash, Domain hash, source IP hash and destination IP hash. These methods are mainly used for caching. Each service is assigned a hash value which is generated using a method which depends on the particular configuration and this value is assigned to a server. Hash values are generated from incoming requests and requests are sent to the server which has the same hash value provided it is online. If the server is offline, new hash values are generated using the last log of the number of services and the IP address and port of the services and the request is sent to the server with the highest hash value.

Least Bandwidth Method: In this method an incoming request is sent to the server which is servicing the smallest amount of traffic. The NetScaler records the number of bytes transmitted and received per 14 seconds as a bandwidth value for each server. Incoming requests are send to server with the smallest bandwidth value. This method can also be modified with weights.

Least Packets Methods: In this method the number of packets that a server has sent and received is used as a load balancing metric. The NetScaler records the number of packets transmitted and received as a number of packets value N every 14 seconds. Incoming requests are sent to the server with the smallest N value. This method can be modified with weights.

Token Method: This method is similar to the Hash Methods. When a client request is received a token value is extracted from the request and the request is forwarded to a server. Further requests which generate the same token are forwarded to the same server. This method will store different values for requests to the same service which use different protocols and the token can be generated in a variety of ways.

Custom: This method uses monitor programs to examine metrics such as CPU and memory usage to determine where incoming requests are sent. Monitor programs send CPU and memory usage data to the NetScaler. The NetScaler then uses this to send incoming requests to the server with the most available memory or least utilised CPU. This method can be modified with weights.

When load balancing at the global level most of the mechanisms available are similar to local level ones. Of course, incoming requests are directed to data centres via DNS rather than servers and further load balancing takes place to determine which server they are sent to. There are, however, some mechanisms which are unique to global load balancing. The following is a list of global level mechanisms available to the NetScaler. Descriptions are omitted if they have already been described at the local level.

Round Robin

Least Connections

Least Response Time

Least Bandwidth

Least Packets

Source IP Hash

Custom

Dynamic This is similar to Least Response Time. The DNS servers are probed at regular intervals to gather response time metric data and incoming requests are sent to the data centres with the smallest response time.

Static Proximity This method uses a GeoIP database to direct incoming requests. The GeoIP database contain ranges of IP addresses and the geographical region to which these requests should be sent. The NetScaler examines the IP address of incoming requests and sends it to the data centre which is closest geographically to the region specified in the GeoIP database.

2.5.2 Current Research

In addition to the options available from commercial packages there has been considerable recent research which uses load balancing to achieve other goals. For example, recent research has focused on lowering electricity costs and carbon emissions for computing clouds. In this section we examine this and discuss its relation to our work.

Qureshi *et al.* [123] proposed a distance-constrained energy price optimiser and presented data on energy price fluctuations and simulations illustrating the potential economic gain. Stanojevic *et al.* [146] detail a distributed consensus algorithm which equalises the change in the cost of energy. This is equivalent to minimising the cost of energy while maintaining QoS levels. Rao *et al.* [125] formulate the electricity cost of a cloud as a flow network. A flow network is a directed graph were each edge has a capacity and receives a flow. It is used to model fluid in pipes and electricity in circuits. Rao *at al.* then attempt to find the minimum cost of sending a certain amount of flow through this network. This work is similar to the work in Chapter 7 but they focus solely upon electricity costs.

Mathew *et al.* [95] propose an algorithm which controls the number of servers online in the cloud to reduce energy consumption. It also maintains a sufficient number of servers at each data center to handle current requests as well spare capacity to handle spikes in traffic. This work is complimentary to the work in Chapters 6 and 7 and both can be used in conjunction with it to lower electricity costs and carbon emissions.

Liu *et al.* [90] propose distributed algorithms which minimize the sum of an energy cost and a delay cost using optimization techniques such as gradient projection to minimise the overall cost of operating the data centre. In addition, they expand their formulation to consider minimizing the sum of the social impact cost and delay cost. They define the social impact cost as a metric for environmental impact of the data centre. By examining the availability of renewable energy and directing load to the appropriate data centres they attempt to reduce the environmental impact of the data centre. Liu *et al.* [89] expand the model proposed in [90] to subtract locally generated clean energy from the energy cost calculation to allow data centres which have clean energy generation facilities to service more load. Moghaddam *et al.* [101] attempt to use a genetic algorithm-based method with virtual machine migration to lower the carbon footprint of the cloud. This work is similar to our work in Chapter 7 as both works use load balancing to lower carbon emissions but different metrics are used to lower carbon emission. This work uses weather data as a metric and this can be inaccurate if locally generated power is not used since other factors affect the carbon emissions. This is discussed in greater detail in Chaper 7.

Gao *et al.* [53] propose the FORTE system which allows the users to consider the three way trade-off between electricity cost, carbon emissions and latency. This is similar to the work in Chapter 7 but it considers a static carbon intensity for data centres while dynamic carbon intensity data is examined in Chapter 7.

Wendell *et al.* [163] propose a general framework for the mapping of traffic between clients and servers. This framework allows for a variety of mapping policies to be implemented to allow the mechanism which controls the traffic to aim for different objectives. This could be used to implement a system similar to that described in Chapter 7 but this is not investigated in this work.

2.6 Rate Limiting

Rate limiting is the control of the rate of traffic sent or received on a network interface. If the traffic exceeds the rate it is dropped or delayed by a device known as a rate limiter. Recently a form of rate limiting known as distributed rate limiting (DRL) has been proposed for use in harmonising the throughput of multiple parallel TCP links [124]. This can be applied to cloud computing as a mechanism which controls the amount of load being assigned to each data centre in the cloud for a given service. DRL can be used to implement fixed cost traffic pricing which is desirable to cloud users and can help to improve utilisation. We discuss its use in Chapter 3.

2.6.1 Distributed Rate Limiting

When DRL is used in cloud computing, an artificial network bandwidth limit is maintained for a service across the cloud. The local link bandwidth at each data centre can then change depending on the demand, provided that the sum of the local link bandwidths does not exceed this limit. In order to achieve this goal, devices known as limiters exchange information to allow aggregate network bandwidth for the service to be enforced, while trying to give equal QoS to the flows passing through the limiters to obey the following fairness postulate [124, 143].

Fairness postulate: Flows arriving at different limiters should achieve the same rates as they

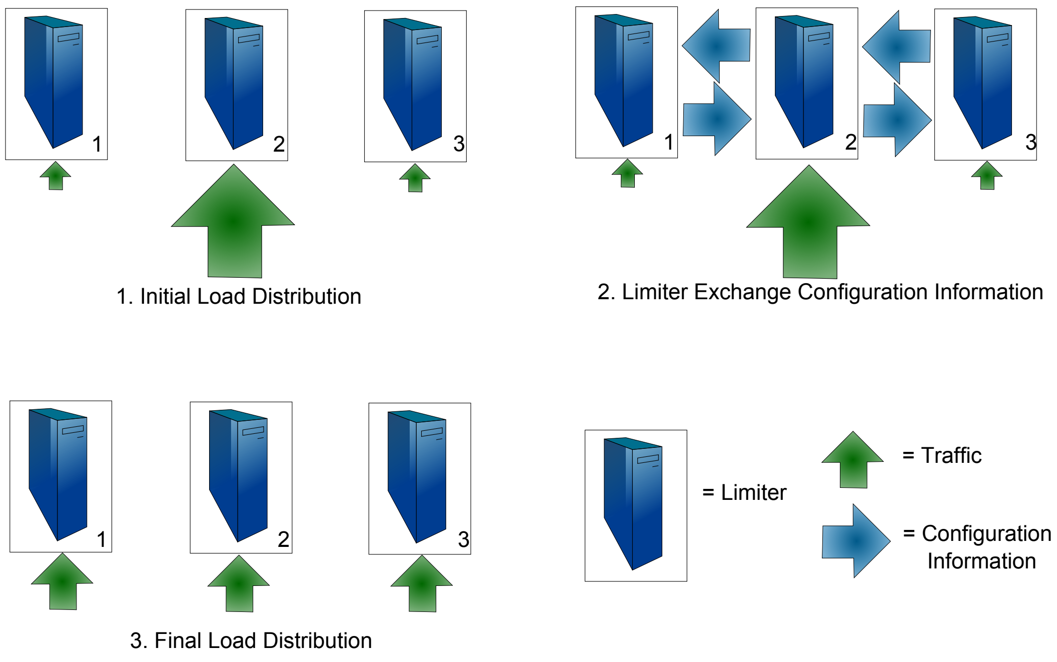

1
2
3
1. Initial Load Distribution
1
2
3
2. Limiter Exchange Configuration Information
1
2
3
3. Final Load Distribution
= Limiter
= Traffic
= Configuration Information

Figure 2.6: Simple example of DRL.

would if they were traversing a single, shared rate limiter.

An example of this is illustrated in Figure 2.6. The initial load distribution is such that there is a large amount of traffic at limiter 2. The three limiters exchange configuration information with their neighbours to balance the load distribution. This exchange of information causes the local service rate of limiter 2 to increase, thereby reducing the traffic levels to reach an equilibrium as seen in the third section of Figure 2.6. The local capacity of limiters 1 and 3 are reduced at the same time but since the level of traffic passing through these limiters is comparatively light this goes unnoticed.

A general framework for the monitoring and control of distributed systems which resulted in an early DRL-like scheme, using Planet Lab as an example is presented in [72]. The Global Random Drop (GRD) and Flow Proportional Share (FPS) algorithms proposed by Raghavan [124] were the first DRL algorithms used for the control of services in the cloud. GRD works by broadcasting the demand at the limiter using a gossip based algorithm described by Kempe [75]. The total demand is calculated at each limiter and packets are dropped randomly if the total demand is greater than the agreed upon capacity. GRD however fails to work for large numbers of limiters as it is dependent on the estimated arrival rates of all the limiters propagating swiftly.

In FPS each limiter utilizes a token bucket with a Service Rate C_i which is initialised as C/N where C is the aggregate capacity and N is the number of limiters. A weight w_i is calculated by dividing the service rate by the flow rate arriving at the limiter. This weight is broadcast to all of the limiters. The service rate is then set to be:

$$C_i = \frac{w_i}{\sum_{j=1}^{N} w_j} C. \tag{2.1}$$

The w_i value is utilized as an estimate of the number of unbottlenecked flows at the limiter. In practice, however, w_i is not a good indicator of the number of unbottlenecked flows at a limiter and this results in the poor performance of FPS as a DRL algorithm [143]. In order to ensure good performance in the key metrics of DRL, a rigorous mathematical framework and two new algorithms Cloud Control with Constant Probabilities (C3P) and Distributed Deficit Round Robin (D2R2) were proposed in [143]. This work was expanded upon in [145, 144]. C3P and D2R2 are discussed in greater details in Chapter 3 and our work experimentally evaluates these algorithms and proposes enhancements to the operation of these DRL algorithms.

2.7 Summary

The cloud computing paradigm in the delivery of internet services requires a large capital investment and current control technologies are unable to operate it in the most cost effective manner. There are several models of cloud computing and this can affect how the cloud is operated. The cloud is comprised of physical hardware which is housed in facilities known as data centres and the specifics of the hardware used and manner in which it is operated will greatly effect the operating costs. Another important aspect of the hardware of the data centre is cooling which is required to prevent equipment damage. There are a number of cooling systems which can be utilised such as aisle containment and “Free Air Cooling”. In addition, there are a number of proposals which use load balancing mechanisms to lower cooling costs. Both of these can be exploited to lower cooling costs while servicing the demand of a data centre. Hardware resources are connected with links and the management of the bandwidth of these links is crucial to the operation of the data centre. There a number of weaknesses in bandwidth management used today but this can be rectified by placing mechanisms which ensure the fair division of this bandwidth. The capital cost of a data centre is considerable and the operational cost is also significant. An examination of the cost of a data centre illustrates the importance of lowering the costs.

There are mathematical techniques that can be applied to control the data centre. Consensus algorithms involve distributed nodes exchanging information and adjusting their state according to an algorithm. This technique is useful as it is distributed which prevents bottlenecks and it can be used in a variety of scenarios. Voronoi partitions are the decomposition of a set of points into subsets. It can be used to divide clients among data centres in the cloud to achieve a variety of goals such as lower electricity costs and carbon emissions.

There are tools which cloud operators utilise to control the cloud. We examined these state-of-the-art tools as it provides a comparison to our work. Load balancing is the distribution of a workload among multiple resources to achieve various goals. Rate limiting is the control of the rate of traffic sent or received on a network interface. A recently proposed form of rate limiting known as distributed rate limiting can be used to harmonise the throughput of TCP flows at various data centres in a cloud. Our work models a cloud system and uses the mathematical tools previously discussed to create algorithms for the control of the cloud. We propose the implementation of these algorithms which function in a similar fashion to load balancing algorithms. When evaluating these we establish a model which considers the hardware of the cloud. We also examine the operation of state-of-the-art algorithms in our model so that comparisons can be made.

Chapter 3

DRL Pricing

3.1 Abstract

In order to recover the construction costs of the data centre, the owner of the cloud must endeavour to have high utilization in the cloud while maintaining performance levels for the users of the cloud. DRL is a mechanism that can be used to attack this problem. A mathematical framework and the convergence and stability properties of two DRL algorithms were proposed in previous work. In this chapter we build upon that work by evaluating the performance of these algorithms on a testbed against the results of a simulation carried out in previous work. We also evaluate the algorithms resistance to failure and propose the *good-neighbours* enhancement to improve its performance. Finally, we propose a method of augmenting the performance of the algorithms in a dynamic environment and evaluate its performance in the cloud.

3.2 Introduction

It is desirable for cloud operators to have high utilisation levels with a rental cost for resources greater than the running cost. This helps operators to recover their capital investment. It has been found that levels can be quite low e.g. 10% for a variety of reasons [57]. One of the reasons for this could be that traffic pricing is mostly usage based [6, 124]. In a usage based traffic policy the more traffic a service attracts the greater the cost will be to the cloud user. Enterprises, however, tend to prefer when an IT service is a fixed cost rather than an unpredictable usage-based cost [68, 109]. The development of a predictable, incremental, fixed cost policy for traffic could be useful in encouraging users to migrate to the cloud and hence, increase utilisation levels and revenue.

There are examples of services which could benefit from the use of a cloud platform at certain times when there is a large need for bandwidth to improve performance. These include the mass distribution of updates for operating systems, patches for video games and video streaming sites such as Youtube. In all of these examples there is a huge demand for bandwidth and the QoS experienced by users could be improved by using cloud services. In addition, the bandwidth required for all of these services varies with time. To encourage utilisation of the cloud for these services, the price for the additional bandwidth must be attractive to the service providers at the time they need it. A market-based mechanism for bandwidth in the cloud [147] could be useful for this. Amazon's EC2 already provides a facility which uses a market-based mechanism to allocate computational resources by offering spot instances. These instances allow customers to bid on unused EC2 computational resources and use these until the instance spot price exceeds the bid the customer originally made.

The cost of traffic, however, remains usage based, and service providers may only be willing to spend a certain amount of money on utilising the cloud platform. To accommodate these services a cloud operator must utilise a mechanism to limit the bandwidth to marketable quantities which have a definite fixed price. We define the quantities as marketable as the price should fluctuate depending on utilisation levels in the cloud. A simple example of this is that the price of bandwidth should fall as utilisation levels drop to encourage more use. DRL mechanisms can be used to achieve this in the cloud as they limit the bandwidth consumed by a service and divide the contracted bandwidth fairly among the data centres which make up the cloud. This helps to improve the QoS experienced by the users. In this chapter we discuss the experimental evaluation of two DRL algorithms, the *good-neighbours* enhancement which improves the resilience of the algorithms to failure and an enhancement which allows the bandwidth consumed to alter dynamically without affecting the QoS for other users of the service.

In Section 3.3 we discuss the framework mathematics behind the DRL algorithms and the specifics of the operation of the C3P and D2R2 algorithms, the *good-neighbours* enhancement and the enhancement which allows the bandwidth utilised to alter dynamically. In Section 3.4 we detail the experimental setup used to evaluate the performance of the algorithms and enhancements. In Section 3.5 we present the results of the experimental evaluation and the

```
1  UpdateCapacities()
2      Once every $\Delta$ units of time do
3        for $i = 1 : N$
4          $C_i \leftarrow C_i + \eta \sum_{(i,j)\in E}(p_i - p_j)$
5        endfor
6      enddo

7    InitializeCapacities()
8      for $i = 1 : N$
9        $C_i \leftarrow \frac{C}{N}$
10     endfor
```

Figure 3.1: Pseudo Code of C3P [143].

evaluations of the two enhancements. In Section 3.6 we summarise how DRL mechanism can be used to make operating the cloud more economically viable.

3.3 Mathematical Framework

In order to understand how the algorithms work the mathematical framework must be described briefly. A CBSP has N data centres. Each data centre $i \in 1,2,\ldots,N$ is able to locally limit the bandwidth of a particular service with a flow population of $\mathcal{F}_i$ to the local capacity C_i, by using the functionality of the limiter. The sum of the service rates at the N data centres must sum to the aggregate capacity C. This is the first constraint enforced by DRL algorithms [143].

$$\sum_{i=1}^{N} C_i = C. \tag{3.1}$$

The capacity of the local limiter can be adjusted and it can exchange information with its neighbouring limiters j. The connections between limiters form a connected undirected graph $G = (N,E)$ where N is the set of the limiters and E is the set of edges that connect the limiters. For a limiter i, j is the set of limiters which share an edge with i in the communication graph G.

3.3.1 Cloud Control With Constant Probabilities (C3P)

Here we detail the specifics of the C3P algorithm which allocates networks resources tp equalise the loss rate of local limiters. In this algorithm each limiter is capable of measuring its loss rate p_i directly. The loss rate is the amount of packets dropped as a fraction of the total packets passing through the limiter for a given time interval. p_i is dependent on the traffic passing through the limiter but in general the more flows that pass through the limiter the higher the loss rate will be. Figure 3.1 displays the pseudo code for the C3P algorithm. Each limiter is initialized to have an equal portion of the overall capacity. At each time interval Δ the capacity of the local limiter is updated using the following $C_i \leftarrow C_i + \eta \sum_{(i,j)\in E}(p_i - p_j)$. The loss rate is a good indicator of the QoS of the flows at the limiter. So when a neighbouring limiter j has a higher loss rate this means that more bandwidth should be assigned to the neighbouring limiter j, by increasing the capacity C_j at the neighbouring limiter and reducing the capacity C_i at the limiter i so that the aggregate capacity is maintained. This will reduce the loss rate at the neighbouring limiter j. The $\eta > 0$ parameter controls how responsive the algorithm is. If n_i is the number of flows passing through a local limiter, $\mathrm{RTT}_j^{(i)}$ is the round trip time of these flows and θ is a constant,[1] then B_i is defined as the following for ease of notation.

$$B_i = \sum_{j=1}^{n_i} \frac{\theta}{\mathrm{RTT}_j^{(i)}}. \tag{3.2}$$

The limiters are connected to a number of neighbours to allow information on loss rates to be communicated. If d_i is the degree[2] of node i in the communication graph G, then η must satisfy the following condition so that the described system would converge [143].

$$0 < \eta < \frac{1}{3}\left(\frac{C}{N}\right)^3 \frac{1}{\max_{1\le i\le N} B_i} \min_{1\le i\le N} \frac{B_i}{d_i}. \tag{3.3}$$

When C3P converges, the loss rates at each limiter will be equalised giving reasonable QoS to all the users of the service in most cases.

3.3.2 Distributed Deficit Round Robin (D2R2)

The setup for this algorithm is similar to that of C3P. In D2R2, however, the limiters are measuring "fair-share" which is the maximal throughput of flows at the limiter. It is measureable directly and dependant on the traffic pattern, but in most cases the more flows at a limiter for a given bandwidth the lower the "fair-share" will be. Let us define $u_s^{(i)}$ as the demand of the number of flows n_i passing through limiter i. We can assume $u_1^{(i)} \le u_2^{(i)} \le \ldots \le u_{n_i}^{(i)}$ without a loss of generality. We use the v symbol to denote "fair-share". We define the throughput

[1]For TCP flows that do not have delayed acknowledgement $\theta = 1.3098$. $\theta = 0.87$ if there is delayed acknowledgement [38].

[2]The degree is the number of neighbouring limiters that the limiter i is connected to.

```
1    UpdateCapacities()
2      Once every $\Delta$ units of time do
3        for $i = 1 : N$
4          $\tilde{v}_i \leftarrow v_0(C_i, \mathcal{F}_i)$
5          $R_i \leftarrow C_i - g_i(\tilde{v}_i)$
6          $v_i \leftarrow \tilde{v}_i + \alpha R_i$
7          $C_i \leftarrow C_i + \eta \sum_{(i,j) \in E}(v_j - v_i)$
8        endfor
9      enddo

10   InitializeCapacities()
11     for $i = 1 : N$
12       $C_i \leftarrow \frac{C}{N}$
13     endfor
```

Figure 3.2: Pseudo Code of D2R2 [143].

passing through limiter i for a given v as a piecewise linear concave function:

$$g_i(v) = \sum_{s=1}^{n_i} \min(u_s^{(i)}, v). \tag{3.4}$$

The “fair-share” $v_0(C_i, \mathcal{F}_i)$ is then either:

1. the unique solution of the equation $g_i(v) = C_i$, if the $\sum_{s=1}^{n_i} u_s^{(i)} > C_i$ or

2. $v_0(C_i, \mathcal{F}_i) = u_{n_i}^{(i)}$, if $\sum_{s=1}^{n_i} u_s^{(i)} \leq C_i$

The Residual bandwidth $R_i(C_i)$is defined as the capacity of the limiter minus the throughput of the flows passing through the limiter [143].

$$R_i(C_i) = C_i - g_i(v_0(C_i, \mathcal{F}_i)) \tag{3.5}$$

We note that the residual bandwidth is strictly positive if the demand $\sum_{s=1}^{n_i} u_s^{(i)}$ is strictly smaller than the capacity. Otherwise the residual bandwidth is zero.

The pseudo code for D2R2 is presented in Figure 3.2. It has two parameters $\alpha \geq 1$ and $\eta > 0$. The accuracy of the algorithm is determined by the α parameter while the responsiveness and stability of the algorithm is established by the η parameter. In order for the described system to converge the η parameter must satisfy [143]:

$$0 < \eta < \min_{1 \leq i \leq N} \frac{1}{2\alpha d_i}. \tag{3.6}$$

The limiters are initialized with an equal share of the overall aggregate capacity. At each time interval the "fair-share" and residual bandwidth are calculated at each limiter and the augmented "fair-share" defined as $v_i = \tilde{v_i} + \alpha R_i(C_i)$ is calculated and sent to neighbouring limiters along the edges of the communication graph G. The capacity of the limiter is then adjusted using $C_i \leftarrow C_i + \eta \sum_{(i,j)\in E}(v_j - v_i)$. Augmented "fair-share" is a good indicator of the QoS of the flows passing though a limiter. If a neighbouring limiter j has a lower augmented "fair-share" more capacity should be assigned to that limiter by decreasing the capacity at limiter i. D2R2 will converge to equalised augmented "fair-share" giving equal QoS to all the users of the service.

3.3.3 Improving Resilience to Failure

DRL is very robust in the event of failure as the loss of a limiter will not affect performance of the TCP flows at the other limiters. The other limiters will continue to function and equalise performance among the remaining users. There would be a loss in the aggregate capacity as the limiters would not be aware that one of their number has failed, but this can be prevented through the use of the *best-friend* enhancement. In the *best-friend* enhancement each local limiter i has a *best-friend* limiter b_i among the neighbouring nodes in the communication graph G and each node informs b_i of its local limit rate C_i. In the event of the failure of the i node b_i inherits its bandwidth by adjusting its bandwidth using [143]:

$$C_{b_i} = C_{b_i} + C_i \tag{3.7}$$

The algorithm then relies on the consensus algorithm to redistribute the capacity. This algorithm will work well in the event of single limiter failure but it may not be able to handle multiple limiter failure since the *best-friend* b_i of the local limiter i may have failed before the local limiter failed, resulting in a loss of aggregate bandwidth. Also, while the consensus algorithm will eventually redistribute the capacity in the event of limiter failure, this may take some time.

In order to address these issues we present the *good-neighbours* solution. Each local limiter records the number of neighbours γ_i it possesses. When communicating the loss rate or "fair-share" to neighbouring limiters j along the edges of the communication graph G it also sends the number of neighbours γ_i as well as its local capacity C_i[3]. In the event of limiter failure[4] the capacity of the neighbouring limiter is updated as follows.

$$C_j = C_j + \frac{C_i}{\gamma_i}. \tag{3.8}$$

The number of neighbours at the neighbouring limiter γ_j is updated to reflect the loss of a neighbour. This algorithm offers significant improvements by attempting to spread the lost capacity more evenly among the remaining limiters while endeavouring to ensure that no capacity is lost. This is achieved with very little cost in terms of communication overhead. Only five bytes needs to be added to the configuration messages. It can be seen that as the connectivity in the communication graph G increases and G moves towards a complete graph, the better the performance of the *good-neighbours* solution will be. A higher γ_i means that the capacity C_i is spread more evenly throughout the communication graph G, resulting in more equalised QoS. Orphaned nodes are also less likely.

3.3.4 Updating Aggregate Capacity

In order for DRL to be used to control bandwidth as a marketable quantity, the overall aggregate capacity must be able to change frequently without unintentionally negatively affecting the QoS of the users of the service. While it would be possible to simply reinitialise the algorithm with the new aggregate capacity, this would mean that all previous updates to move towards a consensus would have to be repeated[5]. In order to avoid this we present a method of updating the local capacities to reflect the change of aggregate capacity while considering previous movements towards consensus. In this method the local limiters would know the old aggregate capacity and the new aggregate capacity and use these so that the local limiter has the same fraction of the new aggregate capacity as it did the old. It does this by using the following to update the local capacity when a new aggregate capacity is set:

$$C_i^{(t+1)} = \frac{C_i^{(t)}}{C^{(t)}} C^{(t+1)}. \tag{3.9}$$

[3]A list of the identification numbers of configuration packets it receives as well as the total number it has received is maintained at each limiter.

[4]Limiter failure is detected by checking when the last configuration packet from a neighbour was received. If it was more than 5Δ, then failure has occurred.

[5]It is assumed that the new capacity, would either be sent to the local limiter by a centralised authority or propagated through the network of local limiters from a single point.

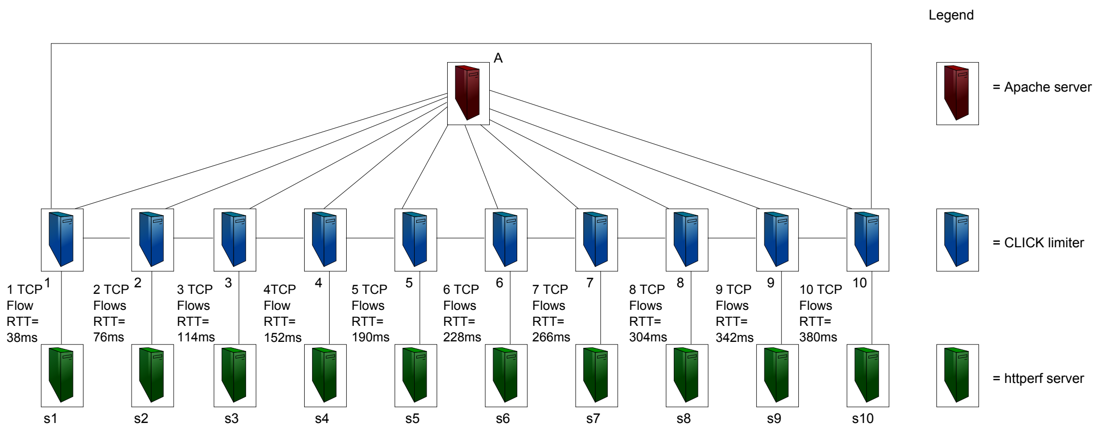


Figure 3.3: Topology of Experiment.

The use of this equation will minimise the effect to the QoS of the service users when the aggregate capacity is changed.

3.4 Experimental Setup

In this section we describe a series of experiments we carried out to evaluate the performance of the DRL algorithms in terms of the metrics of DRL, the algorithms robustness to failure and performance in a dynamic system. The first set of experiments compares the performance of the implementation of the algorithms with that of the simulation carried out in [143]. The same performance metrics and experimental setup are used, so that a clear comparison can be made. In order to properly test the algorithms a topology with ten limiters connected to ten traffic generators with greatly varying round trip times was used, as shown in Figure 3.3. The details of the experimental setup are as follows. We created $N = 10$ local limiters 1,2,...,10. Each local limiter i is connected to its neighbours $(i-1) \bmod 10$ and $(i+1) \bmod 10$. The limiters try to enforce an overall aggregate limit of 40Mbps. Each local limiter i serves i TCP flows (there are 55 TCP flows in total since 1+2+...+10=55) which are generated at 10 other nodes labelled $s1$,$s2$,...,$s10$ in Figure 3.3. These TCP flows are serviced by the node A. The packet sizes were 1500bytes and the round trip times were calculated as:

$$\mathrm{RTT}_j^{(i)} = (8\dot{i} + 30\dot{j})\mathrm{ms} \;\text{ for }\; j \in 1,2,...,i. \tag{3.10}$$

The parameter $\eta_{C3P} = 200000$bps was set to be just below the upper bound determined in (3.3) for stability in the C3P algorithm. We used $\alpha = 1$ and $\eta_{D2R2} = 0.1$ for similar reasons in D2R2. The limiters exchange information every 2 seconds ($\Delta = 2s$). The discrete RTT values for each limiter can be seen in Figure 3.3.

This topology was used so that the simulation and experimentation are directly comparable. It is also a diverse workload and tests the convergence and stability properties of the algorithms with extreme workloads. In the experiment each limiter represents a data centre. The number of flows in the experiment represents the proportion of traffic travelling to the data centre. The performance of the DRL algorithms will scale if the number of nodes servicing the flows and number of TCP flows are increased in the proportions detailed. We do not compare the algorithms with related methods of distributing load as the metrics used to evaluate the performance are different as data centres do not currently market fixed capacity to cloud users. It has been shown that the performance of the other DRL algorithms is inferior to the DRL algorithms examined in these experiments [143] so comparisons were not carried out.

This experiment was carried out on the Emulab testbed [164]. Each of the nodes ran the Ubuntu 7.04 operating system and used TCP Reno. Most of the Emulab machines used an Intel 64 bit Xeon processor and 2Gb of RAM while a few used an Intel Pentium III processor and 256MB of RAM. The limiters were implemented using the CLICK [79] platform. CLICK is a software architecture for prototyping network components. Each of these limiters ran on a separate Emulab node running CLICK 1.7.0rc1. The traffic was generated using the httperf [105] 0.9.0. The web server used at node A was Apache Web server 2.0.63.

3.5 Experimental Results

There are three overall goals to the experiments in this section. The first goal is to establish if the implementation of the DRL algorithms performs correctly. To perform correctly, the DRL algorithm should maintain the aggregate forwarding rate at the specified value and obey the fairness postulate. The second goal is to establish how well the algorithm performs in the event of limiter failure. The failure of limiters could result in the violation of SLAs. There are severe consequences if cloud operators do not adhere to these and mechanisms must be put in place to prevent this. The last goal is to establish how well the DRL algorithms function when the aggregate bandwidth limit changes. The cloud is a dynamic environment and mechanisms need to perform satisfactorily in these conditions.

The results of the first section were determined in a single experiment for each algorithm. The goal of these experiments is to determine if the performance of the DRL algorithms is as good as the performance simulated in [143]. The performance is evaluated by examining how well the algorithm emulates a centralised limiting system and how well it maintains the aggregate capacity. It is also important that the algorithm adheres to the fairness postulate. The topology described in Section 3.4 is used and the flows are run for 1000 seconds.

The results of three experiments per algorithm are shown in Section 3.5.2. The same topology is used with some minor additions which are described in Section 3.4. The length of the experiments varies but this can be seen in the figures that display the results. The goal of these experiments is to examine the performance of the algorithms in the event of limiter failure. The performance of the algorithms with the *good-neighbours* enhancement is compared with the performance of the algorithms without the enhancement. A single experiment for each algorithm is carried out in the last section. The same topology is used with some minor additions that are described in Section 3.5.3. The flows are also run for 1000 seconds in this experiment. The goal of these experiments is to determine how well the algorithms maintain equalised QoS while varying the aggregate capacity.

3.5.1 Comparison with Simulation

In order for the distributed limiters to closely resemble a centralized limiter, the loss rates of the local limiters must be equal. Since the loss rate at the local limiter is also a measure of performance, it is important that the loss rates are equalised in order to equalise the QoS. Figure 3.5 presents time on the x axis and loss rate on the y axis. Each line represents the loss rate at one of the local limiters for the C3P algorithm. The loss rates at the local limiters converge quickly with little variation once equilibrium has been reached. These simulation results presented in [143] also demonstrate that the loss rates converge to equilibrium quickly and maintain that equilibrium once it is reached. This shows that C3P algorithm emulates the behaviour of a centralised limiter well. Since D2R2 uses “fair-share” to enforce its aggregate capacity, loss rate at the local limiter is not a meaningful metric for D2R2.

An important performance metric for a DRL algorithm is its ability to maintain the aggregate forwarding rate at the specified value. Table 3.1 shows the mean and standard deviation of the aggregate forwarding rate for each of the algorithms. The accuracy of both

Table 3.1: Mean and Standard deviation of aggregate forwarding rates in C3P and D2R2.

	C3P	D2R2
Mean(Mbps)	39.92	39.65
std(Mbps)	2.5	1.83

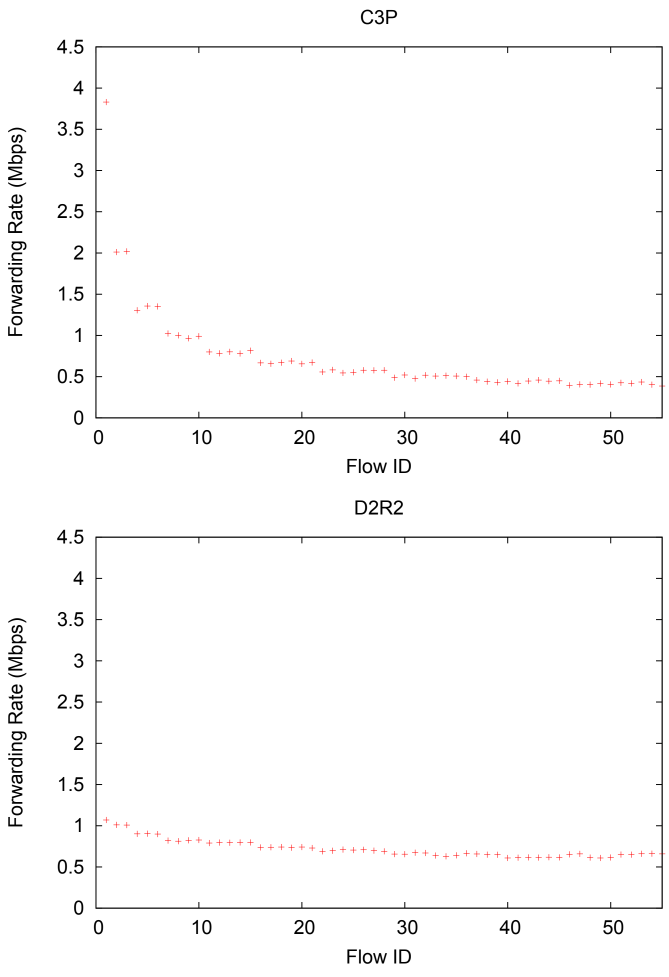


Figure 3.4: Forwarding rates achieved by 55 concurrent flows.

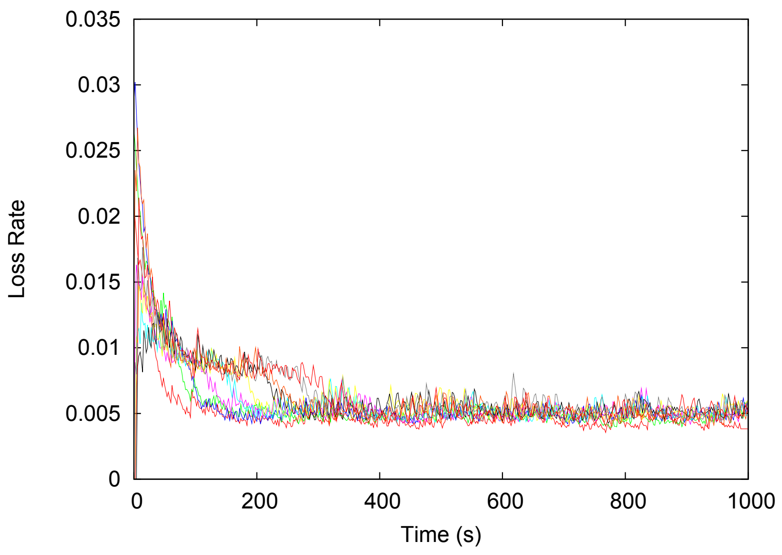


Figure 3.5: Loss Rate of C3P.

algorithms is good as the aggregate forwarding rates of both are within 99% of the specified aggregate capacity of 40Mbps. The simulation results presented in [143] also show a high degree of accuracy in maintaining the aggregate capacity. The mean aggregate capacity maintained in the simulation is slightly closer to the prescribed value of 40Mbps but the accuracy of the mean aggregate capacity obtained in the experiment is still high.

DRL algorithms must also obey the fairness postulate. The forwarding rates of the 55 individual TCP flow passing through the 10 limiters using the C3P and D2R2 algorithms are depicted in Figure 3.4. Each of the points on the graphs in Figure 3.4 represents the forwarding rate of an individual TCP flow. From this figure we can see that the difference in the forwarding rates of the flows is much smaller in the D2R2 algorithm. The figure shows that the distribution of the capacity is fairer in the D2R2 algorithm. The well established Jain's Fairness Index (JFI) [73] is a measure of the fairness of the r flows achieving sending rates $x = (x_1,x_2,\ldots,x_r)$. It is given using the following:

$$\text{JFI}(x) = \frac{(\sum_{i=1}^{r} x_i)^2}{r\sum_{i=1}^{r} x_i^2} \tag{3.11}$$

The JFI of the flows of the two algorithms is given in Table 3.2. We can see that the D2R2 has a higher JFI than C3P. In the simulation results presented in [143] D2R2 also has a higher JFI than C3P. The JFI of both algorithms is slightly higher in the simulation but the experiment confirms that D2R2 is fairer than C3P. We also note that since the JFI of D2R2 is close to 1, the QoS of service among the flows is equalised.

Table 3.2: Jain's Fairness index [73] for C3P and D2R2.

	C3P	D2R2
JFI	0.634	0.978

3.5.2 Failure Tolerance

In this section a number of different types of failure were induced in order to determine how resilient the algorithms are. The same setup is used in all the experiments with the addition of a variety of limiter failures and use of enhancements. In the first experiment no enhancement to the DRL algorithms is used and a limiter fails every 200s. Figure 3.6 shows the aggregate forwarding rate of all the local limiters on the y axis and time on the x axis for both of the algorithms. From Figure 3.6 we can see that at 200s, 400s, 600s and 800s the aggregate forwarding for both algorithms decreases. The decrease is smaller in D2R2 at 200s, 400s and 600s as the limiters which failed had small local limits. From this we can determine that each failure results in a reduction in the aggregate forwarding rate in both algorithms. The distribution of the aggregate capacity among the local limiters is more even in C3P, which is why the decrease in the aggregate forwarding rate is more linear than D2R2. The loss of capacity and the corresponding reduction in aggregate forwarding means that the first constraint of DRL (3.1) is being violated in the event of limiter failure which could result in the violation of an SLA.

Figure 3.7 shows the loss rate on the y axis and time on the x axis. Each line represents the loss rate for a local limiter utilizing the C3P algorithm. From Figure 3.7 we can determine that the loss rate is not affected by the failure of the limiters. The x axis of Figure 3.8 is time and the y axis is augmented "fair-share". Each line represents the augmented "fair-share" of a local limiter using the D2R2 algorithm. From Figure 3.8 we can see the augmented "fair-share" is also not affected by the failure. The loss rate and augmented "fair-share" of the operating limiters remains at the equilibrium. This means that although aggregate bandwidth is lost the fairness postulate is not violated and fair QoS is still received by users. We note that Figure 3.7 is similar to Figure 3.5, which demonstrates that the QoS remains the same despite the failure of the limiters.

In the next experiment the *good-neighbours* enhancement is used and there is a single limiter failure at 200s. This experiment is carried out in order to determine how quickly the algorithms will converge to equilibrium in the event of the capacity being unevenly distributed

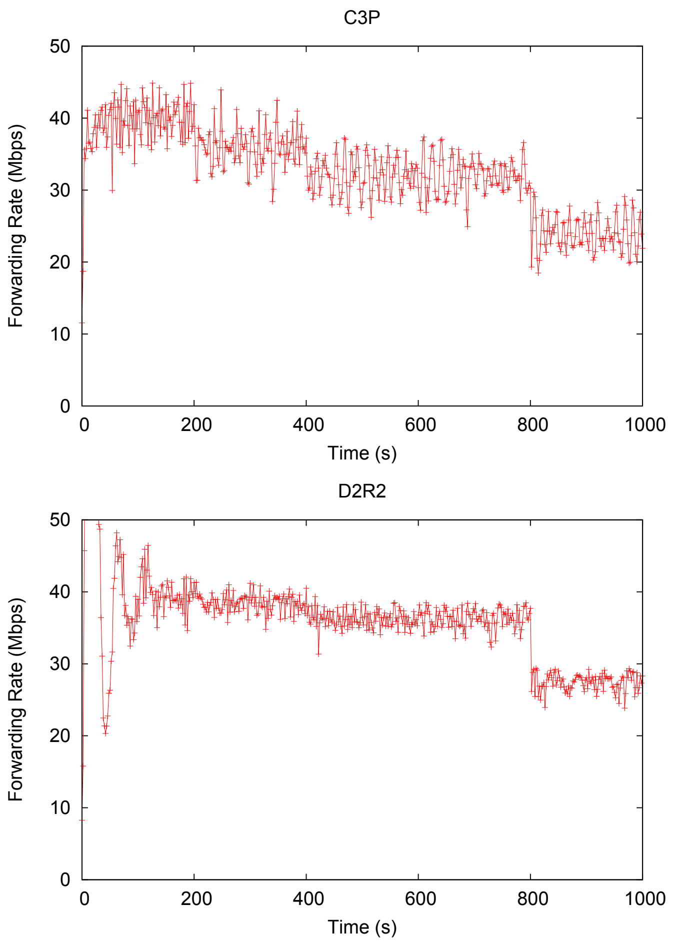


Figure 3.6: Aggregate forwarding rate during limiter failure without *good-neighbours*.

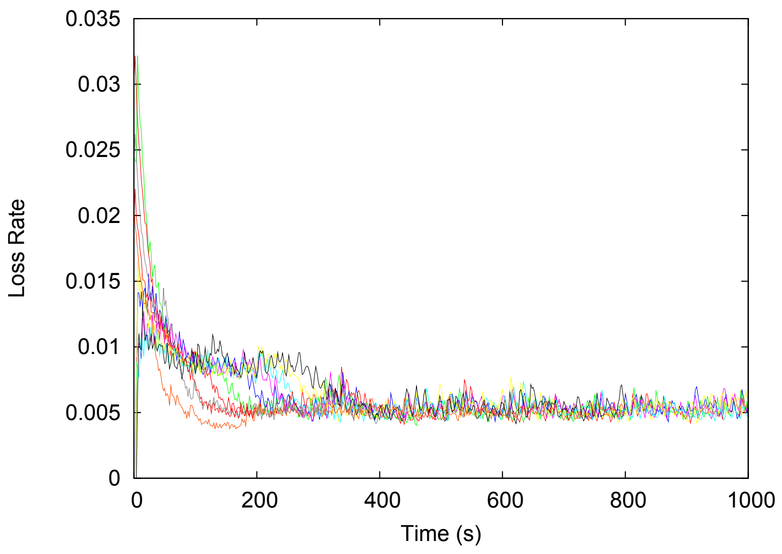


Figure 3.7: Loss rates of operating C3P limiters during limiter failure without *good-neighbours*.

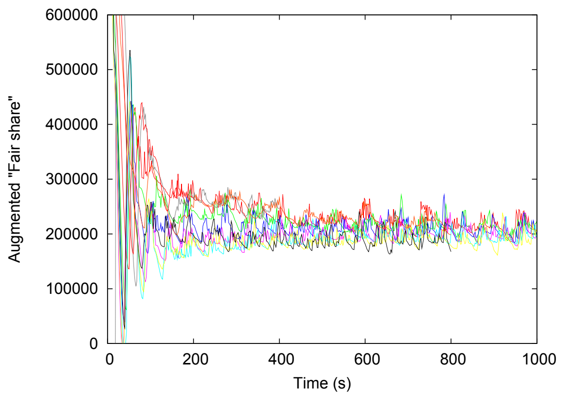


Figure 3.8: Augment “fair-share” of operating C3P limiters without *good-neighbours*.

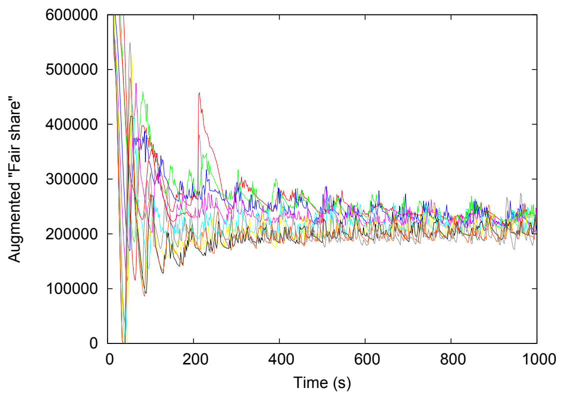


Figure 3.9: Augmented “fair-share” of operating D2R2 limiters during limiter failure with *good-neighbours*.

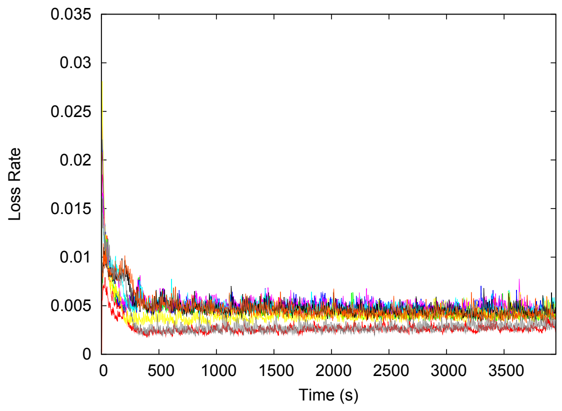


Figure 3.10: Loss Rate of C3P limiters during limiter failure with *good-neighbours*.

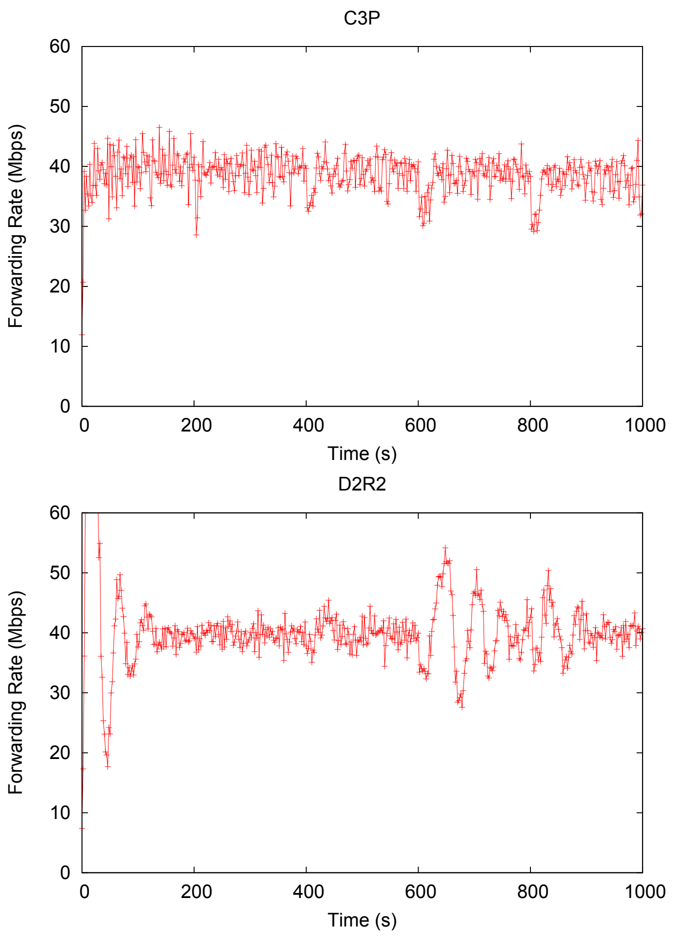


Figure 3.11: Aggregate forwarding during failure of limiters with *good-neighbours*.

due to the failure of a limiter. Figure 3.9 shows time on the x axis and augmented “fair-share” on the y axis. Each line is the augmented “fair-share” of a local limiter employing the D2R2 algorithm. In the figure we can see that the augmented “fair-share” of two local limiters spikes briefly at 200s. This occurs when the capacity of the local limiter that failed is transferred to the two local limiters. It, however, quickly returns to equilibrium and remains there. The x axis of Figure 3.10 is time and the y axis is the loss rate. Each line in the figure represents the loss rate of a local limiter running the C3P algorithm. From the figure we can see that the failure of the limiter causes the loss rate of two limiters to drop as they accept the increased capacity. We also see that loss rates converge to equilibrium as the C3P algorithm adjusts the capacities throughout the cloud to account for the failure. In comparing the performance of the two algorithms we can see that D2R2 converges to equilibrium much faster than C3P. We note that D2R2 maintains a fairer QoS than C3P in the event of limiter failure

In the last experiment the *good-neighbours* enhancement is used and there is a limiter failure every 200s. The goal of this experiment is to show that the *good-neighbours* enhancement will maintain the first constraint of DRL in the event of multiple limiter failures. Figure 3.11 shows the aggregate forwarding rate of the limiters on the y axis and time on the x axis for both of the algorithms. We can see that the aggregate forwarding rate is maintained at the prescribed level despite the failure of four of the ten limiters. In comparing the performance of the two algorithms we can see that D2R2 fluctuates at the third limiter failure at 600s. We note that C3P maintains the aggregate forwarding rate more smoothly in the event of node failure.

We can compare the performance of the algorithms with the *good-neighbours* enhancement with the performance of the algorithms without the *good-neighbours* enhancement by comparing Figure 3.6 and Figure 3.11. We can see that the use of the *good-neighbours* enhancement prevents the loss of 37.5% of the aggregate capacity in the case of C3P and 30% of the aggregate capacity in the case of D2R2 after the failure of the four limiters. This is achieved with an additional 5 bytes in each configuration message which are sent every two minutes which is a negligible overhead. This allows cloud operators to adhere to SLAs in the event of hardware failure with little additional cost.

3.5.3 Changing Aggregate Capacity

The aim of these experiments is to determine how well the algorithms reacts to changes in the specified aggregate capacity using the formula detailed in Section 3.3.4. In this experiment the same setup is used with the exception that the aggregate capacity changes to 60Mbps at 250s, 15Mbps at 500s and 30Mbps at 750s. The y axis of Figure 3.13 show the aggregate forwarding rate of all the local limiters and the x axis shows time. We can see that both algorithms reach the new capacities and stabilise around the new aggregate capacity quickly. We see that C3P's transition to the new aggregate capacity is smoother than D2R2 as the aggregate forwarding rate varies somewhat when the local limiters switch to the new aggregate capacity. Figure 3.12 shows the loss rate on the y axis and time on the x axis. Each line represents the loss rate of a local limiter running the C3P algorithm. From the figure we see that the loss rates remain equal throughout the whole experiment and move to the new equilibrium together. This means that there is no negative unintentional affect to the QoS of the Service Users. The y axis of Figure 3.14 is the augmented "fair-share" and the x axis is time. Each line represents

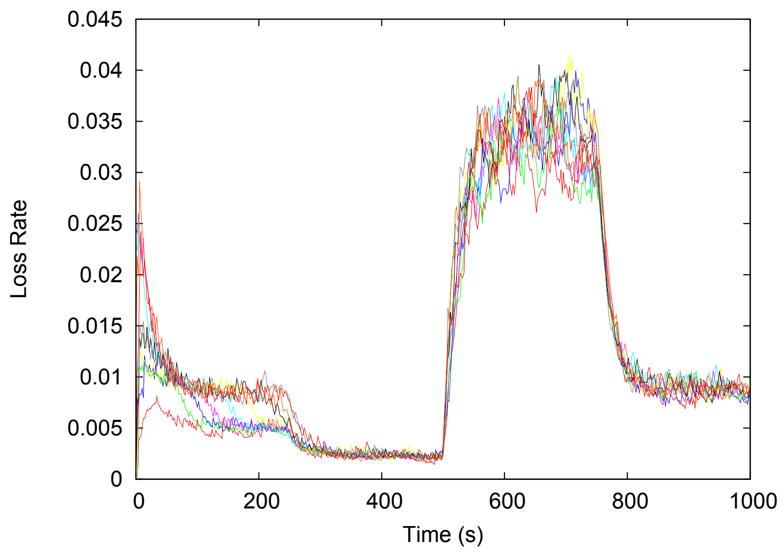
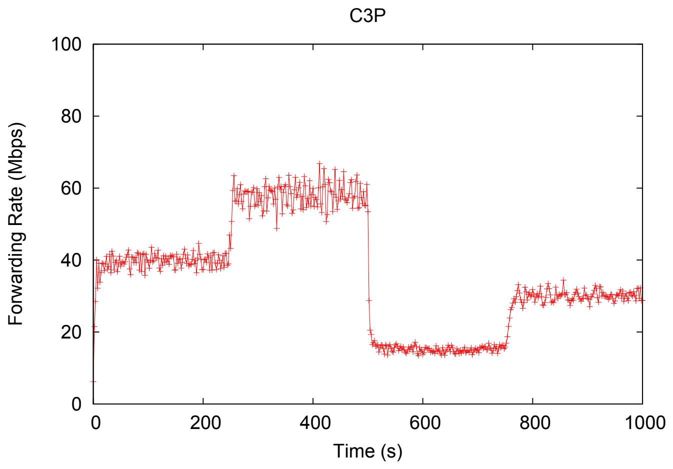


Figure 3.12: Loss Rate of C3P limiters with changing aggregate capacity.

the augmented "fair-share" of a local limiter using the D2R2 algorithm. We can see that the change in aggregate capacity does cause the augmented "fair-share" to fluctuate somewhat. It does, however, converge to equilibrium quickly. This means that there will be little or no effect to the QoS of the Service Users. In comparing the two algorithms we see that C3P performs better than D2R2 when changing its aggregate capacity as the transition in terms of QoS and aggregate forwarding rate is smoother.

3.6 Summary

In this chapter we examined DRL algorithms that can be used to implement a predictable, incremental, fixed cost policy for traffic. We postulated that such a policy could be useful in encouraging users to migrate services to the cloud and hence, increase utilisation levels and revenue. DRL algorithms operate by enforcing an overall aggregate capacity limit across different locations by adjusting local capacity limits at devices known as limiters in accordance with a fairness metric. We described the operation of the C3P and D2R2 algorithms and discussed potential flaws in these algorithms. The cloud is a dynamic environment and operators are bound by SLAs which makes the failure of limiters which utilise the algorithms unacceptable. We proposed the *good-neighbours* enhancement to handle component failure and an enhancement to handle a changing aggregate capacity to address these flaws.



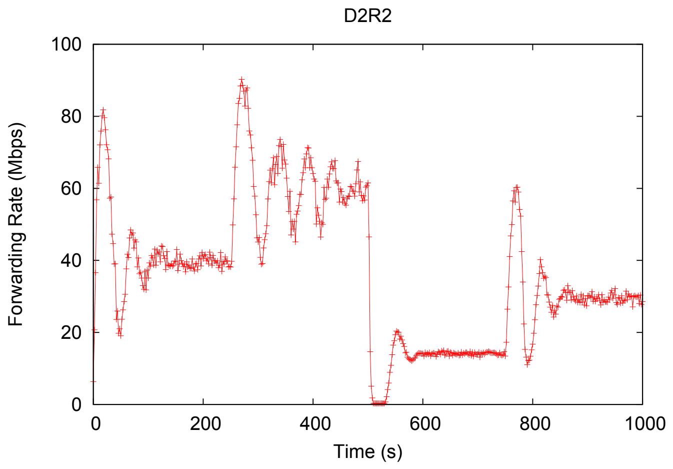


Figure 3.13: Aggregate forwarding rate with varying aggregate capacity. The specified aggregate capacity begins at 40Mbps and changes to 60Mbps at 250s, 15Mbps at 500s and 30Mbps at 750s.

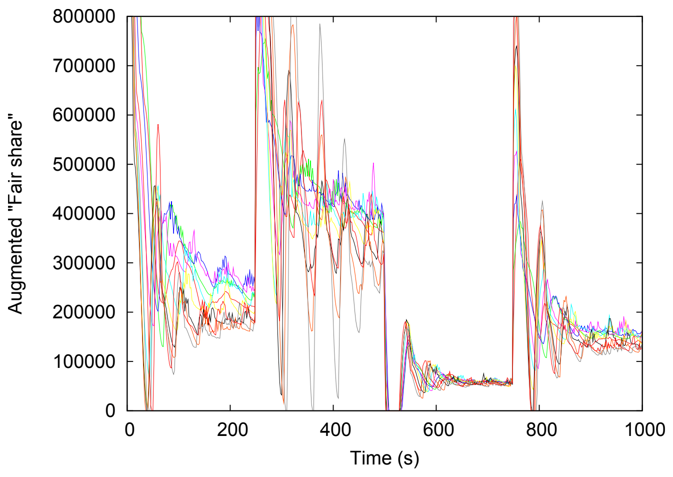


Figure 3.14: Augmented “fair-share” of D2R2 limiters with changing aggregate capacity.

We then examined a number of aspects of the performance of the algorithms. We first examined how our implementation of the algorithm performs using the two main DRL performance metrics. The first is that the difference between the sum of the forwarding rates at limiters and the specified aggregate capacity is as small as possible and does not vary significantly over time. The second is that the algorithms adhere as closely as possible to the fairness postulate. We examined how our implementation of the algorithm performs using these metrics and compared the performance to simulation results previously obtained in [143]. Our results show that both algorithms perform well in ensuring that the difference between the sum of the forwarding rates and the specified aggregate capacity is small. They also show that D2R2 distributes the bandwidth to flows in a fairer manner than C3P. These results are similar to the simulation results.

Next we examine how the algorithms perform in the event of limiter failure with the *good-neighbours* enhancement. The performance metrics are the same as the previous metrics. In the event of limiter failure the spare capacity should be absorbed by neighbouring limiters and then redistributed by the normal operation of the algorithms so that the algorithms adhere as closely as possible to the fairness postulate. We compared the performance of algorithms with the *good-neighbours* enhancement algorithm with the normal operation of the algorithms. Our results show that the *good-neighbours* enhancement maintains the aggregate capacity at the specified level in the event of limiter failure and the effect on the distribution of bandwidth to flows is minimal.

Finally, we examined the performance of algorithms with a changing aggregate capacity with an enhancement designed to accommodate this. In this case the performance metric used was how quickly the algorithms move to the new aggregate capacity, how stable the new aggregate capacity is once it has been reached, and how the well the fairness postulate is obeyed when the change occurs. Our results show that both algorithms move towards the new aggregate capacity relatively quickly. They also show that the C3P algorithm is more stable once the aggregate capacity has been reached.

Chapter 4

Bandwidth Management in Data Centres

4.1 Abstract

Current mechanisms for controlling network resources cannot prevent users from interfering with each other in a shared cloud environment. Proposals made to rectify this, use mechanisms which require interrupts with high timing precision, and usually unevenly distribute bandwidth to TCP flows. We propose the use of an alternative mechanism originally used in the DRL algorithm D2R2 to manage network resources and compare the performance of both.

4.2 Introduction

A public cloud is an environment where multiple users share various computational and network resources. Hence there is potential for both malicious and unintentional service interference where the resource usage of one user impacts the service received by another. Service interference can already been seen on clouds that are in operation today. The throughput of medium *instances* on Amazon's EC2 can vary by 66% [87, 159] and it has been conjectured, based on anecdotal evidence that the cause of this is a lack of mechanisms to prevent service interference [136]. There are numerous mechanisms which can be used for controlling and managing computational, memory and disk resources [27, 61]. However, it is generally accepted that the current mechanisms for managing network resources are not capable of preventing “service interference”. A cloud operator needs to prevent service interference as it lowers the QoS received by cloud users. This could lead to lost customers and low utilisation levels as users may decide to implement alternate methods of service delivery.

Other proposals such as Seawall by Shieh *al.* have concentrated on maintaining the fair divisions of bandwidth among servers, VMs or processes. While this is important it is also crucial that the bandwidth be divided among the flows of the VM or process in a fair manner as each flow could represent an individual service user. The principle focus of this chapter is to detail a mechanism which not only divides bandwidth fairly between servers or VMs but also among the flows connected to these. To this end we propose the use of a “fair-share” dropping mechanism in Seawall rather the token bucket mechanism originally used. Recall that Shieh *et al.* proposed Seawall [136] to manage bandwidth of the links in a data centre. Seawall used the token bucket mechanism which has two flaws. Firstly, it requires high precision interrupts[1] to operate correctly and efficiently at high rates. To generate high precision interrupts Seawall uses one core per rack of servers to generate a heartbeat packet. A heartbeat packet is defined as a packet that is used to synchronise the token bucket mechanism on a network scale. A core which is generating heartbeat packets cannot be used for anything else and represents a wasted capital investment. A second flaw results from the manner in which bandwidth is assigned. In Seawall a local entity (e.g. Virtual Machine, process) is assigned a network weight which is simply a measure of the portion of bandwidth the local entity should receive. A higher network weight results in a local entity receiving a greater proportion of the bandwidth. While the token bucket mechanism does ensure that the local entity receives a a share of the bandwidth that is proportional to its network weight along the network links, it does not ensure that the individual flows of the local entity obtain a fair allocation of the bandwidth assigned to the local entity. As a result certain flows can receive significantly more bandwidth than others. This could lead to lost customers.

Inspired by the DRL mechanism [143, 124, 144], the performance of the bandwidth management algorithm is considered, with a specific view to protecting honest users. In this chapter we compare the performance of the token bucket mechanism and the “fair-share” dropping mechanism used in the D2R2 algorithm [143] in dividing the bandwidth among flows.

In Section 4.3 we examine the operation of the “fair-share” dropping mechanism. In

[1]The ideal token bucket updates every $\frac{1}{r}$ seconds where r is desired rate.

```
UpdateFairShare()
  Once every $\Delta$ units of time do
    if $T > 0.7C$
      $v \leftarrow 0.9v + 0.1(C - T)$
    else
      $v \leftarrow 0.9v + 0.1(C)$
  enddo

PacketArrival()
  if $R < \frac{t_i - v}{v}$
      DropPacket
```

Figure 4.1: Pseudo Code of “fair-share” dropping mechanism.

Section 4.4 we detail the experimental setup used to evaluate the performance of the dropping mechanisms. In Section 4.5 we present the results of the experimental evaluation of the“fair-share” dropping mechanisms and compare it with token bucket. In Section 4.6 we summarise how the “fair-share” dropping mechanism can be used to prevent service interference while ensuring that service users receive a reasonable QoS.

4.3 “Fair-share” Dropping Mechanism

In this section we describe the operation of the “fair-share” dropping mechanism originally used in [143]. The “fair-share” v of the flow population of a local entity can be defined as the maximum forwarding rate of all the flows of the local entity. In order to ensure that the aggregate throughput T of the local entity matches the bandwidth share dictated by the network weight C, the “fair-share” v must be adjusted at a time interval Δ. The difference between the bandwidth share C and the aggregate throughput T is used to update the “fair-share” v if the aggregate throughput T is closer to, or is greater than the bandwidth share C. Otherwise the “fair-share” v is increased so that the aggregate throughput T matches the bandwidth share C. Packets from a flow are dropped if a random number R between 0 and 1 is less than the difference between the throughput t_i of flow i and the “fair-share” v divided by the “fair-share” v. This is illustrated in Figure 4.1. Fairness is achieved here as packets will not be dropped from a flow until it has reached the “fair-share” v value. At this point the flow is receiving a fair amount of the capacity and packets which are received after this will begin to drop with increasing probability as the throughput of the flow increases beyond the “fair-share” value v. Randomness is used to ensure that all flows do not scale back together and consequently to

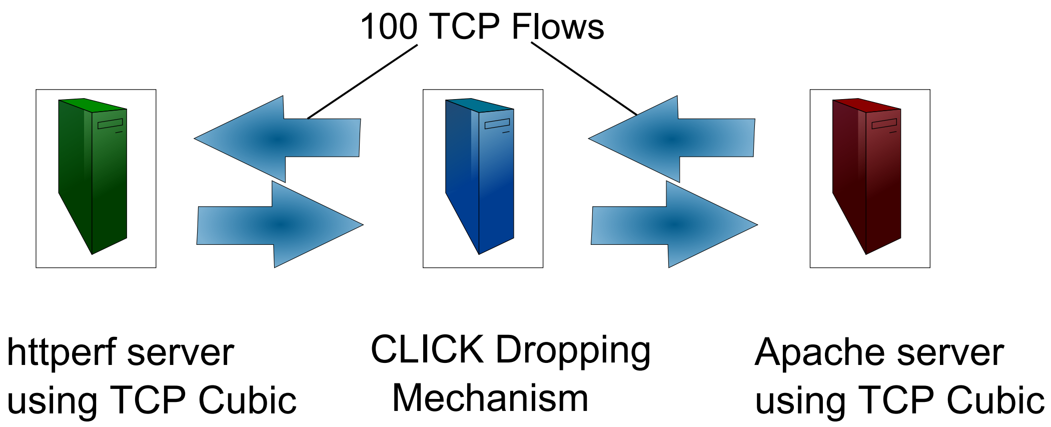


Figure 4.2: Diagram of the experimental setup.

ensure that T matches the bandwidth share C.

4.4 Experimental Setup

In order to evaluate the performance of the proposed “fair-share” dropping mechanism, we implemented both this algorithm, and for comparison, a token bucket mechanism using CLICK [79]. We then used the httperf [105] traffic generator to generate a hundred requests to fetch a large file. Large files were used so that a constant number of TCP flows would persist for the experiment. The traffic pattern generated is similar to that generated by a group of people accessing a service like YouTube. TCP Cubic was used to control the flows [64] as it is the default implementation of TCP on many modern Linux kernels. Cubic allows the examination of the throughput of each flow and the comparison of the performance of the two mechanisms. An Apache web server was used to service the requests. This is illustrated in Figure 4.2. All flows which are directed towards a destination in the data centre will pass through the same final link in the path from source to destination. Since latency in a data centre is so low in ideal conditions and the flows converge to use this single link, the setup is representative of a simulation which has multiple sources in the data centre as flow control is applied on a per connection basis and the difference in latency between the flows is negligible. Both mechanisms are attempting to enforce a bandwidth share C of 40Mbps. We selected 40Mbps as it seems likely bandwidth share given the number of VMs likely to be found on a physical server. The “fair-share” dropping mechanism updates every 2s ($\Delta = 2$s). The token bucket updates every

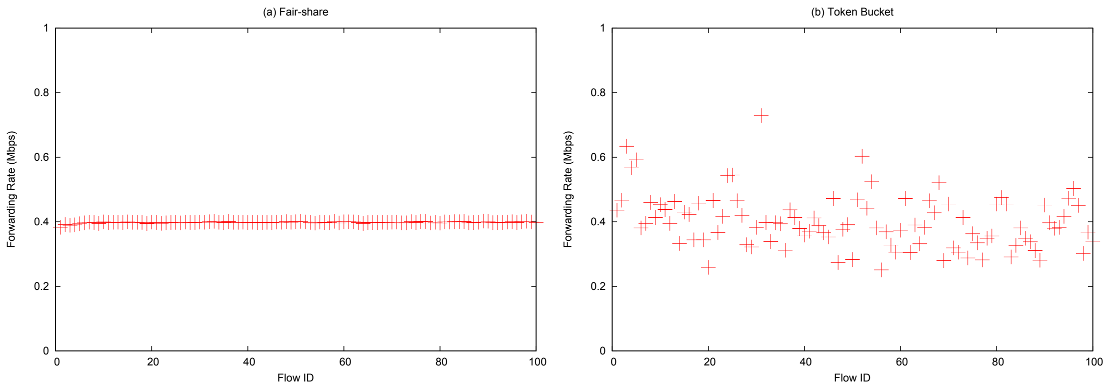


Figure 4.3: The forwarding rates of the individual TCP flows using (a) “fair-share” and (b) token bucket.

10ms. Seawall updates the token bucket every 0.1ms but this requires the use of a processor core for timing only, which as we have mentioned is inefficient. 10ms was selected as it is a small enough value that the token bucket will operate reasonably efficiently without the requirement of a heartbeat packet.

Both “dropping mechanism” implementations also record data that can be used to quantity their performance. The performance of the mechanism can be characterized using a few criteria. The first criterion is that the average aggregate throughput should match the bandwidth share. The aggregate throughput T was recorded at each update interval to investigate this criterion. In addition a criterion to establish the fairness of the division of bandwidth among the TCP flows was required. In this work we use the well known Jain’s Fairness Index (JFI) [73] as a measure for quantifying the performance of both algorithms as a criterion for fairness. Recall that JFI is a measure of the fairness of r flows achieving sending rates $x = (x_1,x_2,\ldots,x_r)$ and is given by the following:

$$JFI(x) = \frac{\left(\sum_{i=1}^{r} x_i\right)^2}{r\sum_{i=1}^{r} x_i^2}$$

A JFI value of 1 indicates that the bandwidth has been distributed fairly. The throughput of each individual flow t_i is recorded at every update interval to evaluate this criterion.

4.5 Experimental Results

The goals of these experiments are to examine the performance of the “fair-share” and token bucket dropping mechanisms in dividing bandwidth fairly among flows and as rate limiting mechanism as this is their core function. We compare the performance of the two mechanisms as the bandwidth that cloud and service users receive is dependent on these. We first examine how close the achieved average throughput of the local entity matches the bandwidth share. Both mechanisms perform well, with an average throughput within 1% of the bandwidth share of 40Mbps. Token bucket, however, achieves this performance at a much higher cost than the proposed algorithm as it must update 200 times more often in order to achieve this level of performance.

The second criterion is the fairness of the division of bandwidth among the TCP flows. The JFI of the "fair-share" dropping mechanism is 1 and the JFI of the token bucket mechanism is 0.9578. From this we can say that the "fair-share" dropping mechanism is better at dividing the bandwidth among the TCP flows fairly again at a much lower network management overhead. This can be illustrated further. Figure 4.3 depicts the flow identifier of the TCP flows on the x axis and the average forwarding rate of the flows on the y axis. Each point represents the average forwarding rate of an individual TCP flow. From Figure 4.3 we can see that the TCP flows have a equal forwarding rate when the "fair-share" dropping mechanism is used. This illustrates that a fair distribution of the bandwidth share has been achieved, and consequently for users of the local entity equal QoS has been achieved. In contrast we can see the forwarding rate of the TCP flows using token bucket ranges from 0.23Mbps to 0.73Mbps. This is clearly not a fair distribution as one user of the local entity is receiving more than three times the bandwidth of another user.

4.6 Summary

In this chapter we examine how the "fair-share" dropping mechanism can be used to improve bandwidth management in data centres. Recently there has been considerable attention given to the implementation of bandwidth management protocols in public data centres. The principal reason for this is to prevent users from exploiting weaknesses in the TCP protocol and using other methods to obtain an unfair share of bandwidth. Proposals which have been made to address this enforce bandwidth limits using the token bucket mechanism which has two flaws. Firstly to function effectively it requires a steady stream of high precision interrupts. A core per server rack is required to achieve this and this represents a significant overhead. Secondly it does not divide the bandwidth among the flows passing through the dropping mechanism fairly which could cause QoS problems. To address this we propose the use of the “fair-share” dropping mechanism in bandwidth management systems rather than the token bucket mechanism.

Firstly we detailed the operation of the “fair-share” dropping mechanism. Next we designed an experiment so that the two mechanisms can be compared. The two performance metrics used to evaluate the algorithm are investigated. Our results showed that both mechanism perform well in ensuring that achieved average throughput is close to assigned bandwidth share. They also showed that the distribution of bandwidth to flows is significantly fairer if the “fair-share” dropping mechanism is used.

Chapter 5

Thermal Management of Data Centres

5.1 Abstract

Thermal management in data centres requires a complicated trade-off between cooling costs and thermally induced equipment failure rates. Using ideas from cooperative control and distributed rate limiting, we describe a distributed architecture that can be used for thermal aware load balancing for a common type of modular data centre. The benefit of shifting load based on thermal considerations is that significant gains in cooling cost can be achieved.

5.2 Introduction

One of the most significant components of the operating costs for a cloud is the energy required for cooling. Consider a 30,000 ft^2 data centre with 1000 standard computing racks, each consuming 10 kW. If we assume an average electricity price of \$100/MWh and a PUE of 1.5-2, the annual cooling cost would be of the order of \$4–\$8 million [118]. The specific cost depends on the layout of the data centre and the particulars of the cooling system used.

Cooling, however, is of considerable importance to the long-term reliability of operations. It has been reported that every 10 °C increase above 21 °C decreases the long-term reliability of electronics by 50% [148, 120]. In addition, it has been shown that a 15 °C rise increases the failure rates of hard disk drives by a factor of two [7], although more recent studies suggest that the increase in the rate of failure may not be as severe as this [42]. It is possible, however, to lower the operational cost required for cooling through the use of the load balancing algorithms while preventing equipment damage. This allows cloud operators to lower their cost and helps them recover their capital investment.

The temperatures of server rack inlets in a data centre are not equal due to complex airflows. The cooling cost of a data centre is dependent on hottest inlet as this must be cooled to prevent equipment damage. This results in some server racks being over-cooled. There have been a number of proposals to improve the thermal management of data centres [134, 103, 35, 117] using load balancing. All of these proposals, however, use a central scheduler which is both a single point of failure and a bottleneck for control messages. They also focus on a traditional data centre architecture. In this chapter we examine thermal management in a data centre which uses modular data centre components. Modular data centres have become increasingly popular in recent years and we can examine their operation accurately as each component has little effect on its neighbouring components. We propose, a robust, distributed algorithm, which attempts to reduce cooling costs, prevent equipment damage and ensure that all service requests are satisfied without the need for detailed calibrations. The algorithm operates by adjusting the load a cloud component is servicing based upon the temperature of that component and its neighbours. Our work allows cloud operators to lower their operational costs while maintaining QoS and hence utilisation levels.

In Section 5.3 we describe the distributed algorithms we use for thermal management. In Section 5.4 we detail the simulation setup used to examine the performance of the algorithm. We present the results of the simulation in Section 5.5. In Section 5.6 we describe the experimental setup used to investigate the performance of the algorithm. In Section 5.7 we evaluate the performance of the algorithm in the experiment. Finally in Section 5.8 we summarise how our distributed algorithm can be used to lower operational costs while maintaining QoS for users of the service.

5.3 Preliminaries

We will study thermal management in the context of a large scale data centre. In the following, we use the term “machine” in a rather abstract sense in that we assume that each “machine”

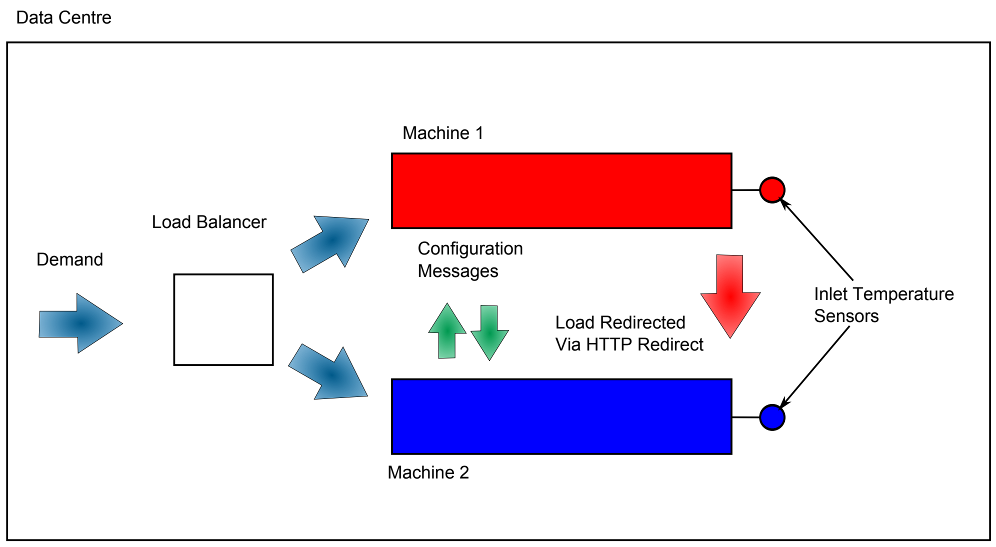


Figure 5.1: Illustration of how the thermal management algorithms operate.

actually consists of a housing that contains a large number of individual servers which, collectively, have a cooling facility associated with them. This assumption is particularly applicable in the case of the increasingly popular modular data centres, where large clusters of servers are housed in shipping containers that are all connected to a common chilled water supply which feeds the computer room air conditioner (CRAC) units, [32]. Such a container would then be considered a "machine" in the context of this work.

5.3.1 Problem setting

To use algorithms to attempt to equalise the temperature of machines in a data centre we need to formulate the operation of the data centre mathematically. We consider a data centre that is constructed using n machines. Each machine $i \in \{1,2,\ldots,n\}$ has, at time $k = 0,1,2,\ldots$, an inlet temperature (or just "temperature") of $T_i(k)$, which represents the temperature of the air sucked into the servers for cooling. Additionally, each machine is servicing a demand (also referred to as "work load") of $D_i(k)$, so that the total demand serviced by the data centre at time k is

$$D(k) := \sum_i D_i(k) \tag{5.1}$$

An important feature of our work is that the total demand $D(k)$ is regulated to some desired value D^*, which may be time-varying (for notational convenience, we omit the dependence of

D^* on k). Hence, the value of the total demand serviced and its deviation from D^* must be known either implicitly or explicitly by at least one machine in the data centre. D^* represents the demand entering the data centre through the access network. The algorithm must know this value so that all requests are serviced.

Given a constant amount of cooling energy supplied to each machine, the temperatures inside each machine will be a direct function of the demand serviced by that machine (since the heat energy dissipated by the CPUs is roughly proportional to the amount of work done by the CPUs). Borrowing from the terminology established in [77], this interdependency is described by the "utility functions" which relates the "physical state" (demand) D_i of machine i to its "utility value" (temperature):

$$T_i = f_i(D_i) \tag{5.2}$$

Note that f_i is typically non-linear and is used to model complicated fluid dynamic effects as well as natural cooling within the machines. Using this relationship our algorithms can adjust the demand to attempt to equalise the temperature of the machines. This is depicted in Figure 5.1. Demand is spread among the machines by a load balancer and the sensors are used to monitor the temperature. Machines exchange information using configuration messages. If a neighbour is colder than the machine HTTP redirects can be used to lower its demand and hence its temperature.

We assume that the workload is distributed uniformly inside the machines. We also assume that the temperature of all server inlets is the same inside a given machine; this assumption is justified in particular when suitable cold aisle containment is used inside the machine (In our simulations we found that the largest difference in temperature between the inlets of servers is 0.3 °C). In addition, one of our simulations (simulation 3) examines how much the demand serviced by neighbouring server racks affects the temperature. Furthermore, even in the presence of this assumption the algorithm works well. Note our simulations are based on Flovent [34], an industry standard computational fluid dynamics simulator, and these indicate that even in the presence of these temperature variations, the algorithm is effective. Finally, we assume that there is sufficient distance between the machines so that heat exchanges can be neglected and that these are cased and isolated from each other. Recall that "machines" refer to housings that contain a large number of individual servers.

Machines must exchange information on their temperature for the algorithms to function.

We assume that a limited information exchange between machines is possible. Specifically, at time k, machine i can provide information about itself (in particular its current temperature) to another machine j if and only if (i,j) is an arc in the directed communication graph $\mathcal{G}(k) = \big(\mathcal{N},\mathcal{E}(k)\big)$, which thus describes the topology of the possible information exchange between machines. This graph is allowed to change over time. The algorithm will not work if a node becomes isolated and it is therefore assumed that the graph is jointly strongly connected over a given, fixed time horizon $m \geq 1$[1]. A graph is defined as strongly connected if for every pair of nodes possible in the graph there is a path from the first node to the second and vice versa. A graph is jointly strongly connected over a a given fixed time horizon if every union of the m graphs that occur during the time yield a strongly connected graph. This assumption is crucial as machines need to be able to receive the updates on the overall desired demand D^* to update their demand accordingly.

We would now like to find an algorithm that will attempt to equalise the temperatures among machines, while ensuring that some total demand D^* is being serviced by the data centre.

5.3.2 Global Demand and Local Temperature Exchange (GDLTE)

In this section we describe an algorithm which can be used to equalise the temperature of machines servicing a demand which is time variant. We call this algorithm Global Demand and Local Temperature Exchange (GDLTE). Related work described in [77] gives an in-depth discussion of three iterative algorithms and variations thereof that are designed to allow a network to achieve a common goal cooperatively, while satisfying certain local constraints. The data centre load balancing problem fits into this framework and we shall now reproduce, for convenience, some of the mathematical statements from this publication (in particular the theorem which deals with stability and convergence).

To apply these results, we must assume that the utility functions are continuous, monotone functions with bounded growth rates (to guarantee feasibility of the solution) and that each machine has knowledge of the total demand being serviced by the complete data centre. It is also assumed that the update law for the algorithm uses a time scale that is larger than that of the dynamics of the settling time for the temperatures; namely that the relationship

[1]The union of a set of graphs on a common vertex set is defined as the graph consisting of that vertex set and whose edge set is the union of the edge sets of the constituent graphs.

between D_i and T_i can be adequately modelled using a static map. Given these assumptions, the following theorem (adapted from Theorem 4.1 in [77]) provides a stable update law to iteratively refine the demands being serviced by the individual machines so that the data centre converges to the desired behaviour:

Theorem 1. *For some initial condition $D_i(k=0)$ and any sequence of strongly connected communication graphs, suppose that the machines iteratively update their work load according to*

$$D_i(k+1) = D_i(k) + \sum_{(j,i)\in\mathcal{E}(k)} \eta_{ij}(k)\big(T_j(k) - T_i(k)\big) + \mu(k)\sigma(k) \tag{5.3}$$

where

$$\sigma(k) = \begin{cases} D^* - \sum_{i=1}^{n} D_i(k+1-M) & \text{if } k+1 \text{ is a multiple of } M \\ 0 & \text{otherwise} \end{cases} \tag{5.4}$$

with $M := n-1$. If the gains satisfy

$$0 < \underline{\mu} \le \mu(k) \le \bar{\mu} \quad \text{and} \quad 0 < \underline{\eta_i} \le \eta_{ij}(k) \le \bar{\eta}_i \tag{5.5}$$

then — provided $\bar{\mu} > 0$ and $\bar{\eta}_i > 0$ are sufficiently small — the demands $D_i(k)$ converge asymptotically to values D_i^* for which $f_i\big(D_i^*\big) = T^*$ for all $i = 1,\ldots,n$ and $\sum_i D_i^* = D^*$.

$\eta(k)$ and $\mu(k)$ are stability parameters and determine how quickly the algorithm moves towards convergence. $\sigma(k)$ is a parameter used to determine if the local demand should be updated due to a change in the total demand. The proof of convergence only holds if the algorithm does not alter the total demand due to a change in the total demand at each time interval and $\sigma(k)$ is used to account for this. For a proof of the Theorem and detailed description of the mathematical assumptions therein, please refer to [77].

Comment. Explicit bounds on $\bar{\eta}_i$ and $\bar{\mu}$ are also given in [77]. While for the purpose of proving convergence in the theorem small values of these constants are required, in practical situations it is found that they can be significantly larger. Mathematical details on bounds of $\bar{\eta}_i$ and $\bar{\mu}$ are given in [77]. These are quite involved and are beyond the scope of this work. In short, the stability conditions are based on connectivity arguments in graphs. As the graphs become large, then these conditions give small controller gains. This approach does not account for structural properties of the graph and consequently may be quite conservative.

```
UpdateDemand()
    Once every $\Delta$ units of time do
        for $i = 1 : n$
            $D_i \leftarrow D_i + \eta \sum_{(i,j)\in\mathcal{E}} (T_j - T_i)$
            if $\mathrm{mod}(k+1, n-1) == 0$
                $D_i \leftarrow D_i + \mu\sigma$
            endif
        endfor
        if $\mathrm{mod}(k+1, n-1) == 0$
            $\sigma \leftarrow (D^* - \sum_{i=1}^{n} D_i)$
        endif
        $k \leftarrow k + 1$
    enddo

InitialiseDemand()
    $k \leftarrow 0$
    $\sigma \leftarrow 0$
    for $i = 1 : n$
        $D_i \leftarrow D^*/n$
    endfor
```

Figure 5.2: Pseudocode for GDLTE.

The update law (5.3) from the Theorem, which we will refer to as GDLTE, thus provides a rule specifying how to iteratively update the load on the machines to balance the temperatures among the machines in the network. It consists of two parts: The first part, which sums the differences in temperatures between neighbours, is aimed at reducing the temperature differences; if, for instance, all neighbouring machines of machine i are operating cooler than machine i, then it should reduce its own demand in order to approach the temperature level of the surrounding machines. The second part in the equation is to ensure that the global demand is satisfied. It can be seen that if the total demand serviced by all the machines is below the required quantity, then each machine should increase their own work load somewhat so that the network, collectively, increases the demand serviced until it reaches the desired level.

The pseudocode shown in Figure 5.2 describes an implementation of this algorithm.

5.3.3 Local Demand and Local Temperature Exchange(LDLTE)

In situations where the communication graph is undirected, where the total required demand is not subject to change, and where the utility functions satisfy stronger assumptions, then considerable simplifications are possible and the algorithms given in [144] apply. Specifically, suppose that: (a) the communications graph is undirected; (b) the desired demand D^* is constant; (c)

```
UpdateDemand()
    Once every $\Delta$ units of time do
        for $i = 1 : n$
            $D_i \leftarrow D_i - \eta \sum_{(i,j)\in\mathcal{E}} (T_i - T_j)$
        endfor
        $k \leftarrow k + 1$
    enddo

InitialiseDemand()
    $k \leftarrow 0$
    for $i = 1 : n$
        $D_i \leftarrow D^*/n$
    endfor
```

Figure 5.3: Pseudocode for LDLTE.

the utility functions $f_i : \mathbb{R} \to \mathbb{R}$ are increasing, concave, differentiable functions, and have continuous first derivatives. These conditions hold in some data centres. The utility function is increasing because a higher demand will result in a higher temperature. It is differentiable and has continuous first derivatives as the changes are relatively smooth and do not result in either steps or a lack of change with demand. The function can be assumed as concave as the complex airflows will introduce some nonlinearity but as the function is increasing the function can be assumed to be concave rather than nonlinear. As two way communication is standard in the data centre the assumption that the communication graph is undirected is justified. The desired demand cannot be considered constant for some data centres as user traffic varies with time. In other data centres the demand can be controlled and a static constant demand is used as these data centres are focused on high performance computing or similar tasks.. As a result of these assumptions the algorithm which we call Local Demand and Local Temperature Exchange (LDLTE) described by the pseudocode in Figure 5.3 may be used to equalise the temperature of machines in a data centre.

The basic idea encapsulated above is as follows. Initially, the demand is divided evenly among the server racks. The demand D_i of each machine i is then iteratively updated with a term that is proportional to the sum over the differences between its own temperature and that of neighbouring racks, that is

$$D_1(0) = D_2(0) = \ldots = D_n(0) \tag{5.6a}$$

$$D_i(k+1) = D_i(k) - \eta \sum_{(i,j)\in\mathcal{E}} \big(T_i(k) - T_j(k)\big) \tag{5.6b}$$

The gain parameter η determines the responsiveness and stability of the algorithm and the choice is discussed in [144]. The stability and convergence properties are captured by the following theorem, cf. [144]. Before stating this theorem, we need to establish some further terminology. Denote by $g_i = f_i^{-1}$ the inverse functions of the f_i, which must exist given the convexity and differentiability assumption above. Note also, due to symmetry, the system given by (5.6) satisfies the demand constraint (5.1) for all $k = 0,1,2,\ldots$.

Theorem 2. *Let d_i be the degree of node i in the communication graph. If η satisfies:*

$$0 < \eta < \frac{1}{2} \min_{1\le i\le n} \Big[-g_i'\big(T_i(0)\big)\Big] \min_{1\le i\le n} \frac{1}{d_i}, \tag{5.7}$$

then the system given by (5.6) *will converge to*

$$\lim_{k\to\infty} D_i(k) = D_i^*$$

with $\sum_i D_i^* = D$, and

$$\lim_{k\to\infty} T_i(k) = T^*$$

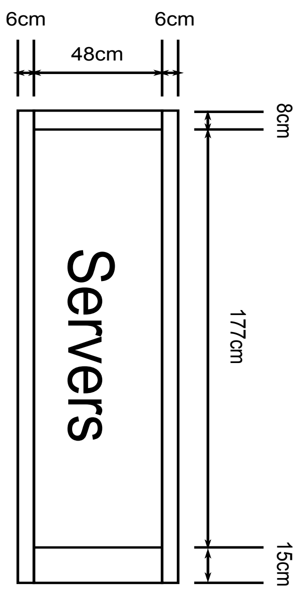
for all $i = 1,\ldots,n$.

Proof. See [144]. □

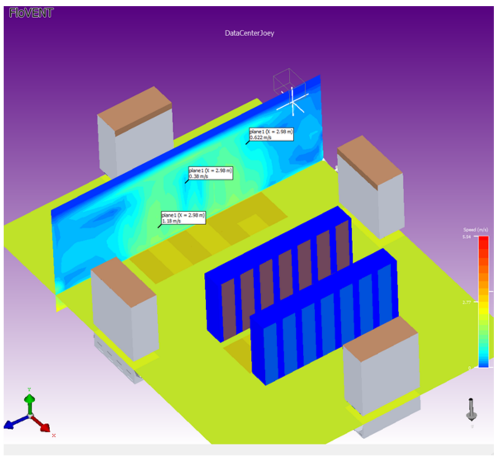
Δ is the time interval between updates and is related to the settling time associated with the $f_i()$ functions. In the original work on this topic [77], these functions are static maps whereas in this application this is not entirely true. Our assumption is, that the dynamics associated with the $f_i()$ functions are fast when compared with the update law. Of course this assumption has to be validated experimentally and the step size determined empirically. For our simulation Δ was chosen to be much larger than the dynamics associated with the $f_i()$ functions. The objective of this work was to illustrate the use of a new and innovative distributed control algorithm developed in [77] for thermal management purposes. In our CFD simulations each iteration of the Flovent software was run every Δ seconds.

5.4 Simulation Setup

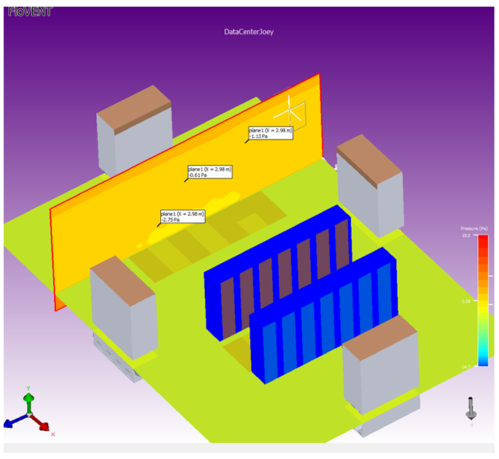
To evaluate the performance of our algorithms we first needed to generate utility functions relating temperature to load for different types of machines. To do this we created computational



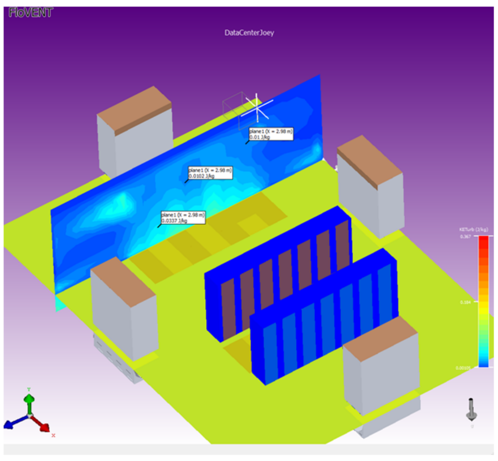
(a) Geometry of the server rack. All components in the server rack are 60cm thick.



(b) Air velocities in front of the server rack.



(c) Pressure conditions in front of the server racks.



(d) Turbulence in front of the server racks.

Figure 5.4: Geometry of the server rack as well as the air velocity, pressure and turbulence conditions in front of the server racks

fluid dynamics models and simulated the operation of these data centres at different demand levels. Once the simulation was completed the temperature was recorded for use in the utility functions. Our simulation setup is similar to that used by Sharma *et al.* and Moore *et al.* in 2005 [134, 103]. Each machine (data centre) used in this study has dimensions 11.7 m $\times$ 8.5 m $\times$ 3.1 m with a 0.6 m raised floor plenum that supplies cool air through perforated floor tiles. There are four rows of servers with seven 40 U racks in each case, resulting in a total of 1120 servers. The dimensions of the server racks are depicted in Figure 5.4(a). Note the front and rear doors where removed to allow the air to flow freely. The servers simulated were based on Hewlett-Packard's *Proliant DL360 G3s* model, which consumes 150 W of power when idle and 285 W at 100% utilization. From this we could determine that the total power consumption of the data centre is 168 kW ($40 \times 28 \times$ 150 W) when idle and 319.2 kW ($40 \times 28 \times$ 285 W) at full utilisation. The flow rate of the server rack was 1,500 $\mathrm{ft}^3/\mathrm{min}$ representing 40 servers with a flow rate of 37.5 $\mathrm{ft}^3/\mathrm{min}$. We also used an ideal *energy proportional* version of said server, [91]. These represent advanced servers where the idle power is virtually zero.

For cooling, the data centre is equipped with four CRAC units "A", "B", "C" and "D" whose locations are also indicated in Figure 5.5. Each CRAC unit pushes air chilled to 15 °C into the plenum at a rate of 10,000 $\mathrm{ft}^3/\mathrm{min}$. The cooling capacity of each CRAC unit is limited to 90 kW. Flovent uses the popular k-ϵ turbulence model; (see documentation describing mathematical modelling given as part of [34]). The air velocity, pressure and turbulence at the front boundary of the server racks are depicted in Figure 5.4(b), 5.4(c) and 5.4(d). CFD simulations are used routinely in data centre design and have been found be to very useful for thermal management purposes by practising engineers. There is, however, a need for detailed validation against experimental data. These concerns, however, are not addressed in this work and CFD simulations are carried out using the popular commercial package Flovent. Given this simple setting, we now describe three basic variations that may arise. In the first case, the machine is exactly as described above with standard HP servers. This type does not use containment and utilises normal servers. It is hereafter referred to as "NC-NP". For the second case, the machine was modelled using a "cold aisle containment" assumption, [99] with CRAC unit D offline. Recall that cold aisle containment refers to the segregation of the cold aisle from the rest of the data centre using physical barriers such as PVC curtains or Plexiglas [17]. This type of data centre is likely to become a more popular option in the future, [121]. This type uses containment and normal servers. It is hereafter referred to as "C-NP". It is also possible

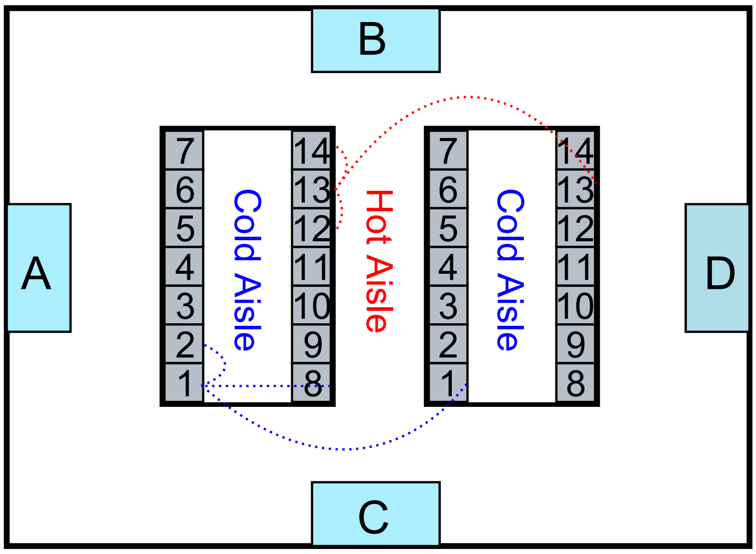


Figure 5.5: Layout of simulation setup. To illustrate the communication graphs used in simulation 3, the dotted lines indicate which other machines the racks 1 and 13 from the left cold aisle can exchange temperature information with. Server racks in adjacent aisles are connected to lower temperature differences between aisles.

to use "hot aisle containment" which is the segregation of the hot aisle from the rest of the data centre. While there is some evidence that "hot aisle containment" may be a more efficient design, it is recognised [121] that there are difficulties in retrofitting this solution which may make "cold aisle containment" the more popular design, at least in the short-term. The third case is identical to the second case, but servers of the *energy proportional* type were used. This type use containment and energy proportional servers. It is hereafter referred to as "C-P".

Each case has a utility function associated with it that describes the relationship between load and maximum inlet temperature found inside the machine. These utility functions, which we will use later in our simulations, were generated by implementing a CFD model in Flovent and running simulations. The power consumed by the server racks were set to reflect the utilisation level. The simulation was then executed and the maximum inlet temperature indicated by the Flovent software was recorded. This was repeated in 10% intervals from idle to full power. The utility functions are depicted in Figure 5.6. From the figure we can see that the NC-NP and C-NP variations operate at a higher temperature when idle. This is expected as the C-P variation does not consume power when idle and hence will be cooler under similar cooling conditions. The temperature of the NC-NP variation rises faster than the C-NP initially as it does not use containment. The temperature of the C-NP variation spikes at around 80%

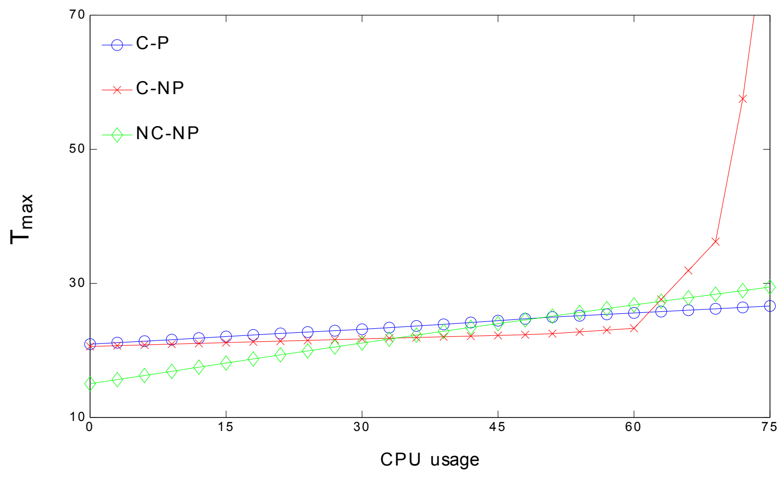


Figure 5.6: Utility functions.

utilisation as the operating CRAC units are unable to remove all of the heat being generated by the servers. The temperature of the C-P variation rises quicker than C-NP as the increases in power with each utilisation level are higher.

In order to evaluate the performance of the algorithms we need to calculate the cooling costs. Let Q be the amount of power the servers consume, T_{sup} the temperature of the air that the CRAC units supply, $T_{\text{safe}} = 25$ °C the maximum permissible temperature at the server inlets in order to prevent equipment damage, T_{max} the maximum temperature at the server inlets of the machine and P_{fan} the power required by the fans of the CRAC units. The "coefficient of performance" (COP) is the ratio of heat removed to the work necessary to remove that heat. It is a function of the temperature T_{sup} of the air supplied to the CRAC. There is considerable debate over the maximum temperature that should be permitted at the inlet in order to prevent equipment damage. While there are guidelines in this area they have changed recently [11] as the recommended maximum operating temperature has been set at a higher value. We selected the value $T_{\text{safe}} = 25$ °C as it is a well established value which has been used in other systems [134, 103]. The cooling cost C can then be calculated as:

$$C = \frac{Q}{\text{COP}(T_{\text{sup}})} + P_{\text{fan}} \tag{5.8}$$

If the highest temperature found at any inlet in the data centre is significantly less than the "red-line" temperature, then the CRAC is cooling the data centre excessively. In such a situation, the supply temperature can be raised by $T_{\text{safe}} - T_{\text{max}}$ (by reducing the amount of cooling) to reduce costs while still observing T_{safe}.

Comment. While other factors such as complex nonlinear flow effects and the cost of pumping air to difficult-to-reach parts of the data centre affect the cooling cost, the highest machine temperature is the major driver of cost. This observation is what motivates us to equalise temperatures. By equalising the temperatures we lower the highest temperature of the machines which allow us to increase the temperature of the air supplied to the CRAC units and lower cooling costs.

If, in turn, $T_{\text{safe}} - T_{\text{max}}$ is negative the equipment is in danger of being damaged and the supply temperature must be lowered in order to cool down the machine responsible for T_{max}. In our simulations, to determine this maximum inlet temperature, we used the commercial CFD simulator Flovent. Each of the four CRAC fan units consumed a constant 10 kW so that for each machine $P_{\text{fan}} = 40$ kW and $T_{\text{sup}} = 15\ ^\circ$C. The COP curve used to calculate the cooling costs is a standard curve for a water chilled CRAC and is given in [103].

Once we had established the utility function for the different modular data centres we constructed a number of simulations to examine the performance of different aspects of the algorithms. The first two sets of experiments were carried out using MATLAB. The particulars of the simulation relating to the number of machines, total demand and the communication graph were programmed into the simulation. The update rule was executed at each machine for a number of iterations and the temperature and local demand of the machines were recorded at each interval. In the final simulation Flovent was used for greater accuracy. In this case the update rule of LDLTE was run at each machine. The demand and temperatures were recorded. The demand was then altered and the CFD simulation was executed again. This was repeated for a number of iterations.

5.5 Simulation Results

In this section we examine three aspects of the algorithms. Firstly we look at the operation of GDLTE with varying demand. Some data centres are dynamic environments and any algorithm designed to lower operational costs must function correctly in changing circumstances. Not all data centres have dynamic demands as they may be used for high performance computing or similar work and hence the operator can specify the demand. For this reason we look at the performance of LDLTE with a static demand. We also investigate how the algorithm reacts in the event of a network link breaking. Hundreds of links exist in data centres and failure is common so algorithms should be able to handle these failures accordingly. Finally we inspect the performance of the algorithm when we remove the assumption regarding lack of heat exchange between machines. This was achieved by examining how LDLTE performs in a scenario where the machine abstraction represents server racks rather than modular data centres. Heat exchange can occur in some data centres and an examination of how the algorithm performs can be used to determine if it can be used in this scenario.

5.5.1 Simulation 1

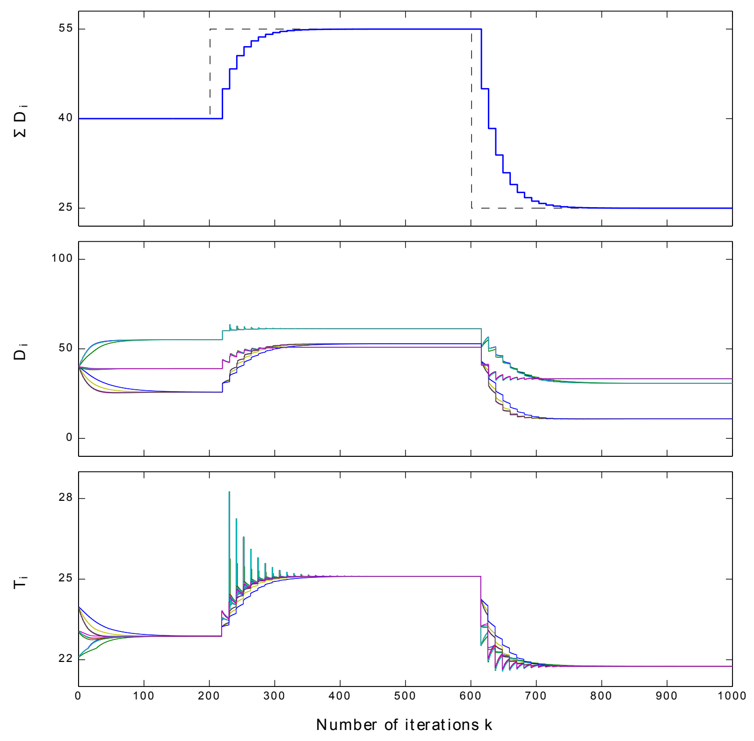


Figure 5.7: Performance of GDLTE at three demand levels $D^* = \{40\%, 55\%, 25\%\}$, which are indicated by the dashed line in the top plot, using $\eta = 0.1$, $\mu = 1/3$.

The setup for this simulation consists of a modular data centre site housing twelve machines (containers), four of each type described in the simulation setup. Our objective in the following is to regulate the aggregate CPU load to three levels: 40%, 55% and finally 25% in a situation where the resulting communication network is strongly connected, but chosen randomly. A random network was chosen to show that the algorithm can function in a variety of networks. The resulting evolution over time of the aggregate demand $\sum D_i$, individual demands D_i and individual temperatures T_i are given in Figure 5.7. Each line in the middle and bottom graph represents the demand and temperature of one machine. The simulations are measured in terms of iterations for reasons which are discussed in Section 5.3.3. If we examine the last graph in Figure 5.7 we can see that the temperature of the hottest machines cools as the simulation progresses during the first one hundred iterations. This drop in temperature occurs when applying the algorithm with 40% utilisation. The cost saving achieved are significant when compared to the data centre when demand is distributed equally. The maximum rack inlet temperature drops by approximately 1 °C and consequently the cooling cost drops from 1.514 MW to 1.416 MW, yielding a 6.5% reduction in the cooling costs. This represents a saving of tens of thousands of dollars annually.

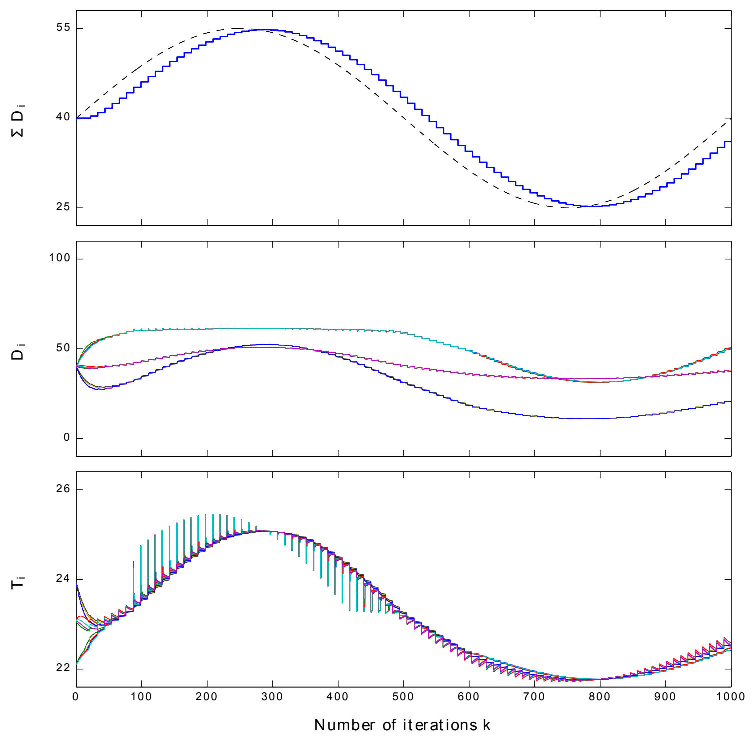


Figure 5.8: Performance of GDLTE with demand varying in a periodic fashion, using $\eta = 0.1$, $\mu = 1/3$.

Comment. As the equilibrium state is achieved for any randomly connected graph, these cost savings are robust with respect to changes (such as link failures) in the communication topology.

Finally, Figure 5.8 depicts the aggregate demand $\sum D_i$, individual demands D_i and

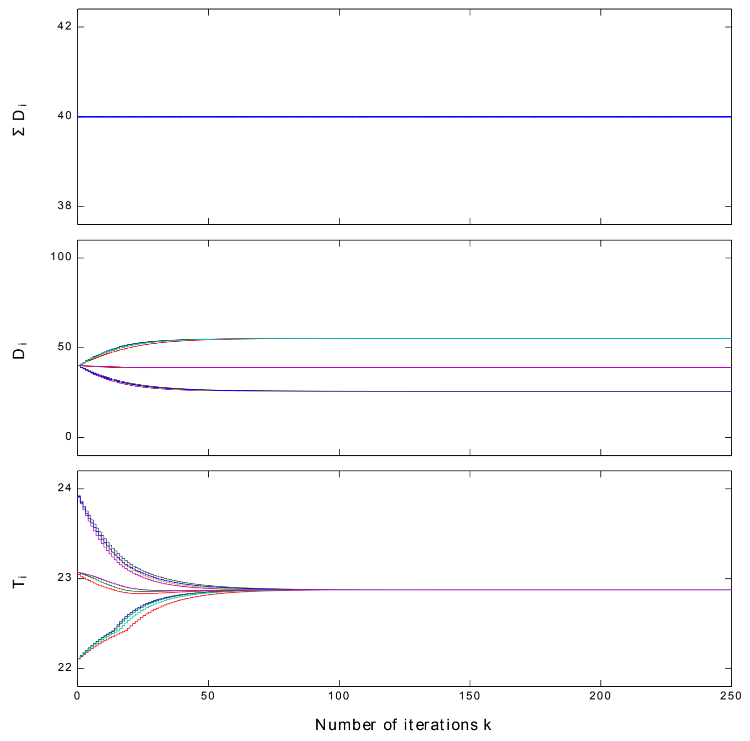


Figure 5.9: Performance of LDLTE with constant demand of 40%, using $\eta = 0.1$.

individual temperatures T_i in a scenario where the demand is varying in a periodic fashion; for example daily demand patterns are often assumed to be periodic. Each line in the middle and bottom graph represents the demand and temperature of each machine respectively. The dotted line in the top graph of Figure 5.8 represents the demand entering the data centre and solid line is the demand being serviced by the machines. The reason that it lags is that changes in the total demand must be communicated to the servers so that they can accept more demand. If we inspect the first and last graphs shown in Figure 5.8 we can see the temperature of the machines remain equal for the most part and satisfactorily tracks the aggregate demand even though GDLTE is designed for fixed point regulation only.

5.5.2 Simulation 2

The setup is exactly as before. However, we now enforce the additional assumptions made in Sec. 5.3.3; recall particular the assumption that the communication graph is undirected and hence there is symmetry in the information exchange. This means that both nodes can send and receive information from each other. Figure 5.9 depicts the aggregate demand $\sum D_i$,

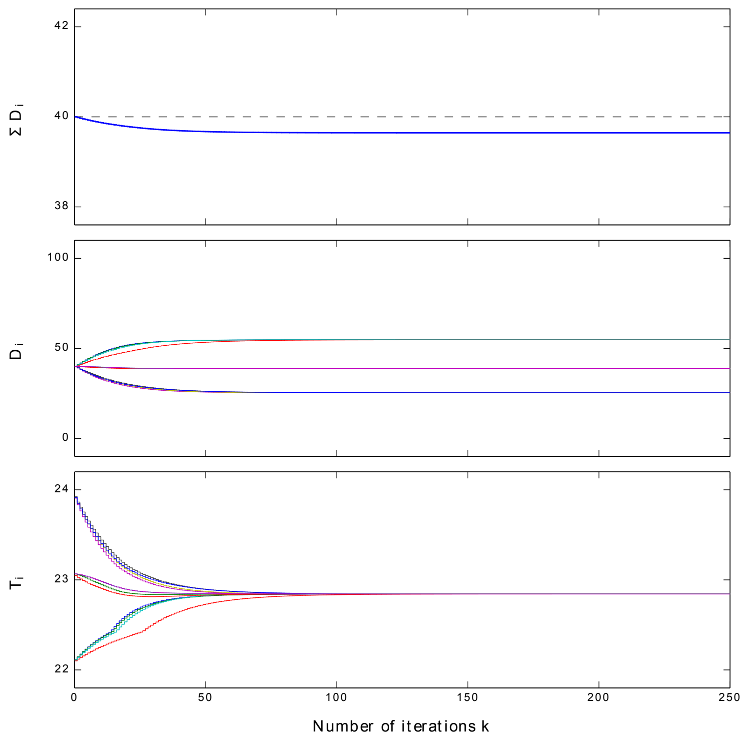


Figure 5.10: Performance of LDLTE with a single broken link and constant demand of 40%, using $\eta = 0.1$.

individual demands D_i and individual temperatures T_i while LDLTE is equalising temperatures for a constant total load of 40%. Again we can see from the last graph in Figure 5.9 that temperature of the hottest machines cools as the simulation progresses during the first one hundred iteration lowering the cooling cost. Note, however, that breaking a single link so that two nodes can no longer exchange information result in a drop in the total demand. Figure 5.10 depicts the aggregate demand $\sum D_i$, individual demands D_i and individual temperatures T_i while LDLTE is operating with a broken link in the communication graph. As can be seen in top graph in Figure 5.10, the aggregate demand falls which means that the total demand constraint can no longer be satisfied and the site cannot satisfy the most basic quality of service requirement, namely that all requests are answered. This result was expected as the algorithm was never designed to operate in such a scenario and enhancements similar to the *best-friend* and *good-neighbours* discussed in Chapter 3 could be used to prevent this.

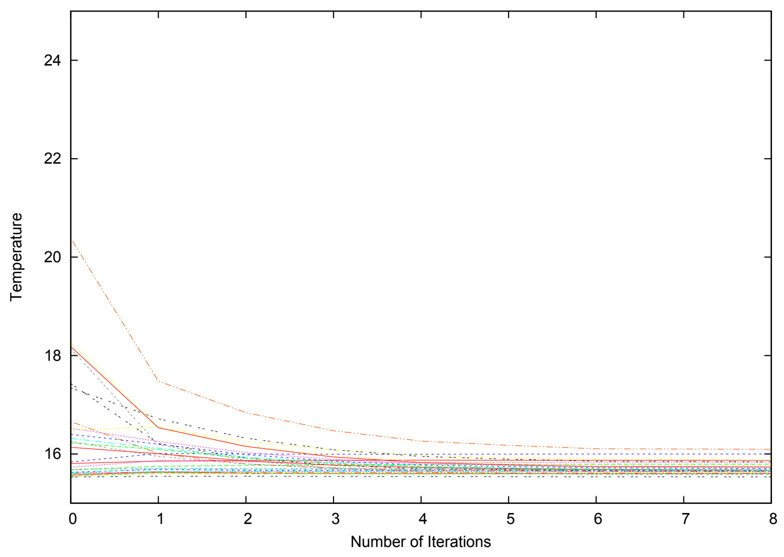


Figure 5.11: $\eta = 13$. The temperature of inlets of the server racks at each iteration of LDLTE inside a modular data centre.

5.5.3 Simulation 3

A basic criticism of the discussion thus far may concern the type of data centre site considered. The simulations thus far have worked under the assumption of containment. This is not always valid. Namely, that T_i of machine i depends not only on D_i but also potentially on D_j $(j \neq i)$. In Simulation 3 we remove the assumption regarding the lack of heat exchange between machines. The algorithm works well in this case also. To conclude this section, we briefly apply LDLTE to one such situation. Similar results can be expected for GDLTE also.

Let us now consider a single machine as described above, in particular of type C-F. Then assume that the server racks inside are labelled in pairs "1" to "14" as shown in Figure 5.5. While any connected communication graph could be used in our algorithm, we chose the following topology for $\mathcal{G}$: Let each server rack be connected to its immediately adjacent server racks and to the server rack with same label in the other cold aisle. Server racks labelled "1" and "7" are connected to server racks directly opposite them across the cold aisle (that is, "8" and "14" respectively). This results in $\mathcal{G}$ being a connected, undirected (3-regular) graph which is a resilient graph as all nodes are of equal importance in terms of connectivity. This topology was selected as it allows servers to equalise the temperatures inside the enclosure and accept demand from the other enclosure if there is a temperature difference between them. As can be seen from Figure 5.11, even in this non-ideal scenario where heat exchange is possible, LDLTE still manages to, more or less, equalise the temperatures across server racks. It is important to

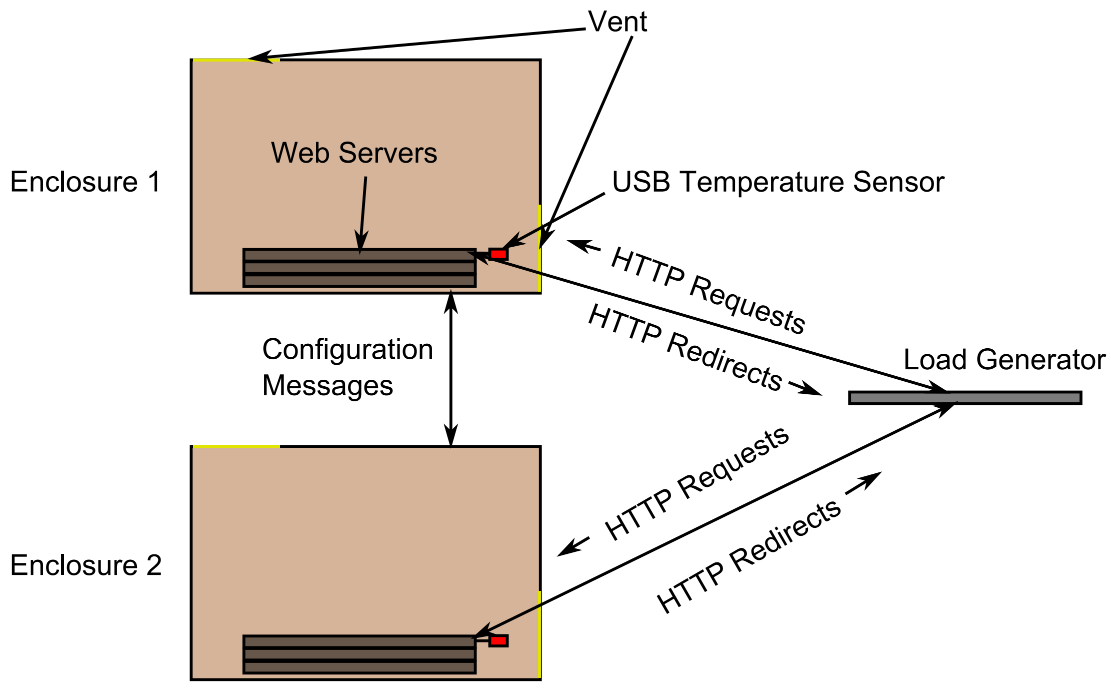


Figure 5.12: Diagram for the experimental setup used to evaluate LDLTE.

note that this simulation is not MATLAB based but is a full scale Flovent CFD simulation. In this case a CFD model with the initial conditions was set up. The simulation was then executed and the temperature of the server inlets was recorded. These were used to adjust the power consumed by the server racks and the simulation was run again. This was repeated until the temperatures converged.

5.6 Experimental Setup using Real Hardware

In this section we examine a small scale experiment devised to evaluate the performance of LDLTE on real hardware. We only experimentally evaluated LDLTE as GDLTE is designed for use with a large number of nodes and the resources were not available to examine this. There were two main goals to this experiment. The first was to examine how to implement the algorithms in data centres. The CFD simulation provides data on the performance of the algorithm, further work needed to be done to design protocols which could control the demand using HTTP redirects. The second goal was to examine how the experimental results compare with the simulation results. CFD simulations are usually very accurate but airflows can be extremely complex and can deviate from the models used by the CFD simulations.

To achieve these goals we put together the testbed depicted in Figure 5.12. We built two wooden enclosures each housing three Dell R300 servers with Intel 64 bit Xeon processors and 2GB of RAM. Each enclosure represents a "machine" abstraction discussed in Section 5.3. The dimensions of the enclosures were 122cm$\times$82cm$\times$82cm. The servers were placed in the wooden enclosures to contain their heat output. Three servers do not generate enough heat to significantly change the temperature of a room. The temperature difference between idle and full power inside the enclosure, however, is significant. Each enclosure was built with vents to allow the wooden enclosures to be quickly cooled between experiments. They also contained a USB temperature sensor which could be used to measure the temperature at the inlets of the servers.

A packet processing component (implemented as a CLICK [79] element) was created to control the demand according to the specifics of the algorithm. At each Δ interval the packet processor obtains the temperature at the server inlets and sends this data in a configuration message to the other server enclosure. Upon receipt of this configuration message the packet processor determines how much demand it should service using the update rule of LDLTE which is depicted in Figure 5.2. It then uses HTTP redirects to send demand it should not service to the other machine. If it should service all of its demand then the element does nothing as the redirects issued by the other machine will increase its demand as LDLTE dictates. In Figure 5.12 we can see that a load generator is used to generate demand for the machines. The traffic generator used was httperf [105] 0.9.0. This traffic generator does not respond to http redirects so another CLICK element was used at the load generator to detect HTTP redirects and issue commands to httperf to re-issue the request.

Before we carried out the experiment to examine the performance of LDLTE we examined the relationship between temperature and demand for a single wooden enclosure. This is needed to generate a η value. To do this we recorded a temperature and then ran the three servers at various utilisation levels for 30min and recorded the temperature again. We allowed the enclosure to return to a stable temperature before running it at another utilisation level. The utility function of the enclosures is depicted in Figure 5.13. It should be noted that the y axis of the figure shows the temperature difference between the start temperature and end temperature and rather than the absolute. The reason for this is that the experiments were carried out in an environment where the ambient temperature outside the enclosures could vary slightly. As such the temperature difference is a more accurate measure of the relationship

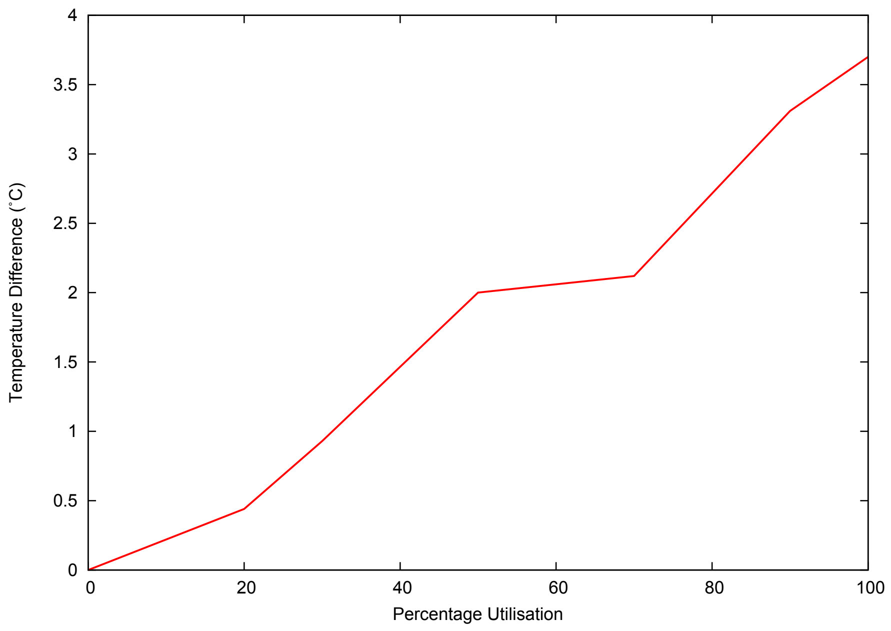


Figure 5.13: Utility function for machines in experiment.

between temperature and demand. The relation between absolute temperature and demand could be inferred by adding the temperature difference to a constant temperature value. From Figure 5.13 we can see that the temperature rises as the percentage demand increases. An η value could be calculated from the figure and the bounds given in [144] which would guarantee stability. We decided, however, to set it to an artificially high value for the experiment to increase the speed of convergence.

5.7 Experimental Results

In this experiment we directed load to one enclosure for an hour to create a temperature difference between the enclosures and then sent enough requests to achieve 50% utilisation at each enclosure and activated the algorithm. The performance of the algorithm is depicted in Figure 5.14. From this figure we can see that the temperature of the hottest machines drops from 35.62°C to 33.5°C and that the temperature of the colder machine increases from 27.88°C to 31.06°C. This compares favourably with the simulation results as both cause a drop in the temperature of the highest machine which could be used to lower cooling costs. After an hour the system saturates and the lack of demand at the hot machine does not cause its temperature to drop any further. It is possible that the convergence seen in the simulation was not present

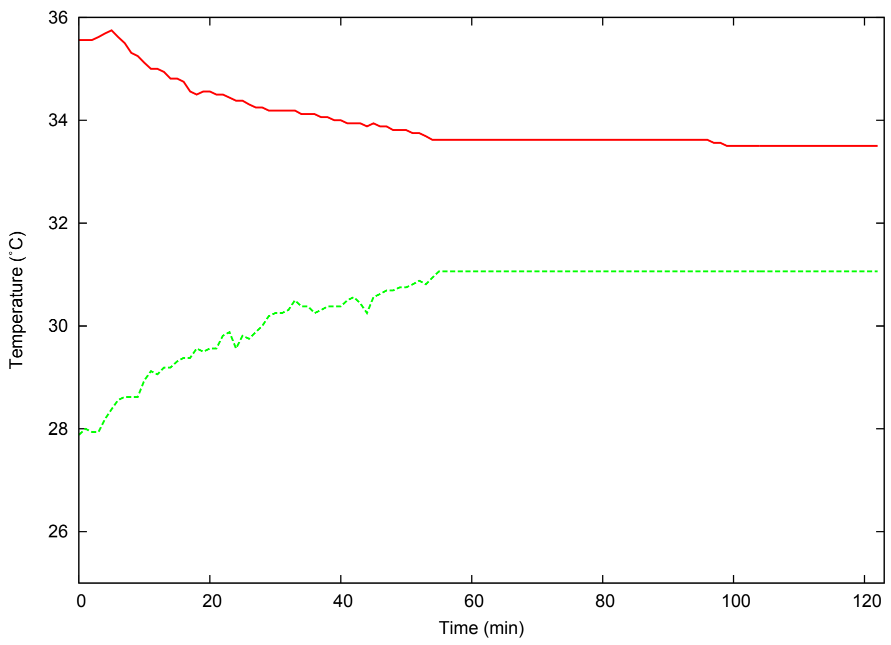


Figure 5.14: Temperature of machines while LDLTE is operating.

here because heat exchange was occurring between the enclosures as they were both in the same room.

5.8 Summary

In this chapter we evaluated several algorithms to lower the cooling costs of data centres without causing equipment damage. By equalising the temperature of the machines and lowering the highest of these temperatures the supply temperature can be lowered to lower cooling costs while still preventing equipment damage. To this end we evaluated two algorithms to equalise temperature while servicing the demand.

Chapter 6

Carbon Emissions Control

6.1 Abstract

The carbon emissions of electricity suppliers can vary greatly between different geographical regions. The traffic for a given service can come from anywhere on the planet and the further the request has to travel the greater the negative effect on quality of service (QoS). It is desirable to route traffic to the resources which cause the lowest carbon emissions but this can affect the QoS. We propose an algorithm that minimises the total cost of the trade-off described. Our results imply that carbon emissions can be reduced with little effect on the QoS.

6.2 Introduction

A public cloud can consist of several data centres located in different geographical regions. If this is the case, data centres will use different electricity suppliers which in turn utilise different types of power plants to generate electricity. The carbon emissions associated with powering data centres has become an important factor in the operation of the cloud. Greenpeace report [60] on the carbon emissions of selected data centres and the percentage of electricity used which is generated by power plants that utilize fuels which emit a relatively large amount of carbon. This report has been used to exert political and popular pressure on the companies to improve their environmental policies.

The carbon intensity of a power generation plant is defined as the carbon emitted for a given amount of energy generated. The carbon intensity of power plants which utilise a particular fuel is presented in [52, 84]. While there is currently little financial motivation to utilise green or clean energy, there are two reasons to consider using load balancing to lower the carbon emissions emitted by the cloud. Firstly, there has been an increase in the regulation of carbon emissions and schemes like the European Union Emissions Trading Scheme (EU ETS) [2] are in operation. It is probable that the right to emit carbon into the atmosphere will be traded as a commodity in the future. If this is the case it will directly contribute to lowering costs for the cloud operator. Secondly cloud users may prefer to utilise the cloud service which has the smallest environmental impact and the use of load balancing which considers carbon emissions will encourage utilisation of the cloud and help the cloud operator to recover their capital investment.

In this chapter we examine carbon emissions control in the cloud. Lowering carbon emissions is a desirable goal but a reasonable QoS must be simultaneously maintained as some services will have time constaints. Anecdotal evidence suggests that increasing the latency of a web service by half a second has been found to lead to a 20% drop in traffic and revenue [88]. We propose that there can be a trade-off between average service request time and carbon emissions. In our work we model the trade-off as a cost function with a relative price function to represent the relative importance of the two factors. A similar approach is used in active queue management (AQM) [142]. We then use the subgradient method to minimise the cost function. We use the subgradient method as it is a well established method for minimising functions.

In Section 6.3 we detail our algorithm to minimise the cost function which represents the trade-off between carbon emissions and average service request time. We described the simulation setup we utilised to investigate the performance of the algorithm in Section 6.4 . We will then evaluate the results of the simulation in Section 6.5. In Section 6.6 we summarise how our algorithm can be used to lower operational costs while maintaining QoS for users of the service.

6.3 Mathematical Preliminaries

6.3.1 Problem Formulation

We begin by examining the cost function for a scenario where a cloud user opts to use servers in a single data centre. The cloud user can vary the number of servers n that are providing a service. A cost function can then be established to represent the average job time $T(n)$, the carbon emissions $G(n)$, and the relationship between them. The average job time will decrease as the number of servers increases, but the amount of carbon emitted will increase as the number of servers increase. The cloud user would like to minimize the average job time and carbon emissions or some combination thereof. The cost function can be defined as:

$$C(n) = T(n) + P(G(n)),$$

where $P(G(n))$ is the relative price function reflecting the importance of short job time versus carbon emissions. A relative price function is used to represent the fact that carbon emissions G(n) have the same importance as an average job time P(G(n)) to the cloud user. We assume that C(n) is a convex function of n so that convex optimisation techniques can be used to minimize the cost function. This is a reasonable assumption as the average job time will decrease as the number of servers increases until a point is reached where the carbon emissions will cause the cost function $C(n)$ to increase.

We can incorporate multiple data centres (DCs) in this framework. If $n = \{n_i\}$ is the vector of the number of servers at each DC and N is the total number of DCs, the cost function can be redefined as:

$$C(n) = \sum_{i=1}^{N} \Big(T_i(n_i) + P(G_i(n_i)) \Big).$$

The goal of the problem is to minimize $C(n)$ and it can be stated as an optimisation problem as follows:

$$\begin{aligned} \text{minimize} \quad & C(n) = \sum_{i=1}^{N} \Big(T_i(n_i) + P(G_i(n_i)) \Big) \\ \text{subject to} \quad & n_i \geq 0 \quad \forall i \end{aligned}$$

6.3.2 Algorithm

The subgradient method can be used to solve this problem and the pseudo-code for the algorithm is presented in Figure 6.1. The subgradient method is an iterative method for solving convex minimisation problems. The number of servers operating at each DC is initially set to be $\frac{S}{N}$. Where S is the total number of servers operating initially. The cloud user will set the S value

```
1  UpdateServerNumber()
2      Once every $\Delta$ units of time do
3        for $i = 1 : N$
4          $g(k)$ = Subgradient of $C(n_i)$ in $n_i(k)$
5          $n_i \leftarrow n_i - \alpha_k g(k)$
6        endfor
7        $k \leftarrow k + 1$
8      enddo

9   InitializeServerNumbers()
10      for $i = 1 : N$
11          $n_i \leftarrow \frac{S}{N}$
12      endfor
```

Figure 6.1: Pseudo-code for algorithm

based upon an estimate of the number of servers required to provide the service. Initially an equal fraction of the estimated number of servers is activated at each DC. The subgradient of $C(n_i)$ at each DC will then be calculated and used to update the number of servers operating at the DC. The goal of this update is to gradually move toward the optimal point of the cost function. α_k is the step size. It determines the stability and responsiveness properties of the algorithm. k is the iteration number and is used in the calculation of the step size. The use of k is required to ensure convergence, as a diminishing step size is usually required to reach the optimal point.

6.4 Simulation Setup

In this section we describe the experiments used to establish realistic average job time $T_i(n_i)$ and carbon emission $G_i(n_i)$ functions and the setup for the simulation of the algorithm described in Section 4. In order to simulate the operation of the algorithm we needed to establish realistic average job time $T_i(n_i)$ and carbon emission $G_i(n_i)$ functions. To do this we used the *httperf* [105] traffic generator to generate load for an Apache web server. Our test load involved the fetching of a dynamic webpage which required 50ms of computation, and the transfer of 300kB of data. This load was selected as it is equivalent to the average size of a webpage in experiments carried out by Google [55] and it contains a computation component which consumes more power than data transfer alone. The server used was a Dell R300 with an Intel 64 bit Xeon processor and 2Gb of RAM. A number of webpage requests were directed to the server and the average job time was recorded. The number of webpage requests was gradually increased while

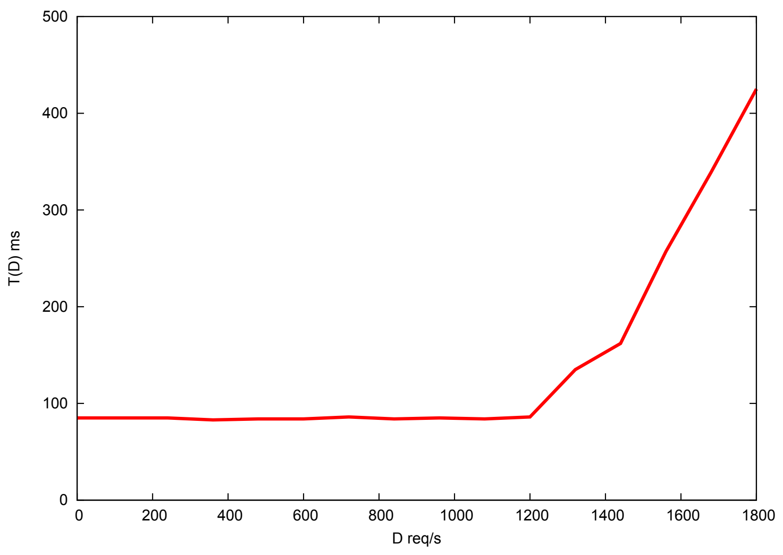


Figure 6.2: Average Job Time with various Demand.

recording the average job time. This is depicted in Figure 6.2. It is seen in the figure that the average job times remain effectively constant until a point where the server can no longer handle the number of requests is reached and the requests are then queued. This causes the increase in average job time. We then used these results to establish a $T_i(n_i)$ function for a given number of requests for a service.

We assume that the sources of load are globally distributed. This mean that the clients of the service are distributed evenly across the globe. If a large portion of the load is directed to a single DC there will be a negative effect on the average job time. There are two reasons for this:

- The internal network topology of some DCs means that links become more oversubscribed in terms of bandwidth as overall traffic increases [58].

- The average job time is affected by the computation time, the length of the links between the client and the server [141] and the time spent in router buffers before being forwarded to the next link. If the load is globally distributed, the average distance between the client and server along the links will increase as more load is sent to a single DC.

This negative effect was incorporated into $T_i(n_i)$ function by increasing the average job time when a large portion of the load is assigned to a single DC.

We connected the server to an electricity usage monitor and its power consumption was recorded under the same load conditions used to establish the average job time function. This

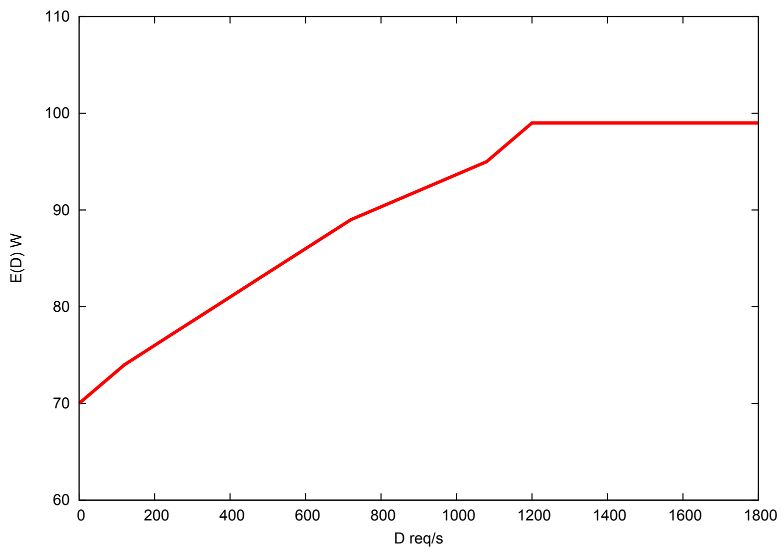


Figure 6.3: Power with various Demand.

is depicted in Figure 6.3. From the figure we can see that the power consumption increases steadily until it is operating at full power. At this point, requests are queued as they cannot be processed directly, and the processor cannot consume any more power. This data was used to establish a carbon emission function for a specific number of requests for a service. Our results found that idle power is ∼70% of the server's peak power. This is consistent with the results from the literature which indicate that the idle power of servers is between 60% and 75% of the peak power [25, 45]. In order to establish a realistic carbon emission function we need to know the carbon emissions per kilowatt hour. This is dependent on the electricity supplier used by each DC.

In our simulations we constructed a scenario where the traffic is being divided among two DCs. The first data centre uses relatively clean energy with associated emissions of 100 grams of carbon per kilowatt hour $100g/kWh$. The second DC uses relatively dirty energy with associated emissions of 500 grams of carbon per kilowatt hour $500g/kWh$.

In selecting these values we attempted to pick contrasting but realistic carbon emission profiles. A DC using the cleanest energy possible would cause approximately 20 grams of carbon per kilowatt hour $20g/kWh$ [52, 84]. Such a figure is difficult to achieve currently as it would require energy from purely renewable sources. A DC using the dirtiest energy possible would produce approximately 900 grams of carbon per kilowatt hour $900g/kWh$ [52, 84]. Electricity suppliers, however, will rarely use coal and oil only, as hydro and gas power plants are the best to react to sudden changes in demand.

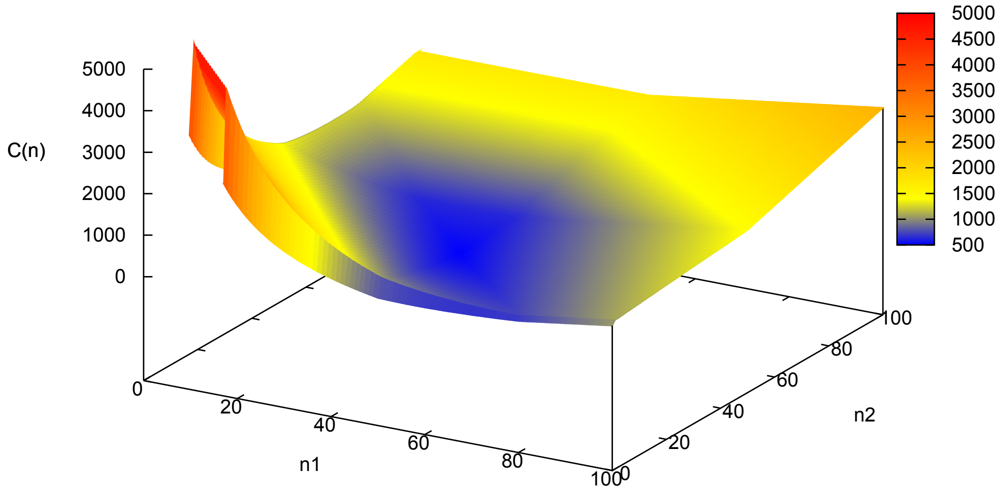


Figure 6.4: Overall cost function for different numbers of servers at the two DCs.

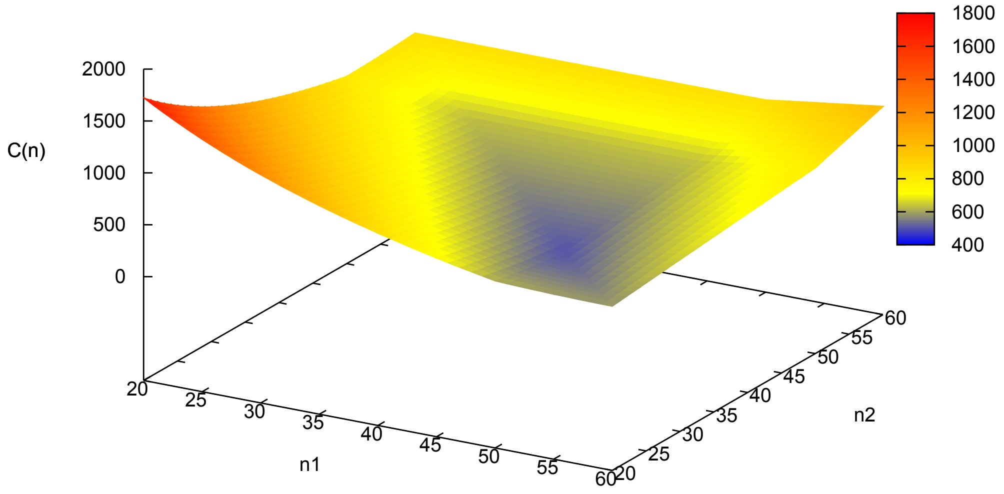


Figure 6.5: Close-up of optimal point of overall cost function for different numbers of servers at the two DCs.

To simulate the algorithm we have assumed static traffic conditions but the algorithm easily extends to dynamic conditions. The service receives 96,000 requests every minute. We chose this level of requests as it is a reasonable demand for a service offered by a relatively large content provider such as a news website. We also assumed that the requests and servers in the DCs are homogeneous. By homogeneous requests we mean that all requests require the same amount of computation and data to be transferred. By homogeneous servers we mean that all servers have identical processors, RAM and network cards. This means that the servers will complete a computation in the same amount of time and use the same power to complete it. As a result $T_1(n_1) = T_2(n_2)$.

In order to establish a value for the relative price function we must examine the cost function at one DC. Under initial conditions the average job time at the first DC is 85ms $T_1(n_1) = 85ms$, and the carbon emissions are 6.6g/min; namely $G_1(n_1) = 6.6g/min$. This yields a total cost of $C_1(n_1) = T_1(n_1) + G_1(n_1) = 85 + 6.6 = 91.6$ if average service request time and carbon emissions are equally important ($P(G(n)) = G(n)$). In this case a drop of 10ms in the average job time would justify an additional 9g/min of carbon emissions. This is not a reasonable trade-off between carbon emissions and QoS as users will not notice a 10ms delay while the increase in carbon emissions is significant. The relative price function was then scaled to $P(G(n)) = 10G(n)$ to provide a more reasonable trade-off between carbon emissions and QoS.

The requests are distributed evenly among all the servers and a server runs at peak power when it is receiving 1200 requests a minute or more. This means that 80 servers running at peak power are able to service the static traffic conditions of 96,000 requests every minute. We set the initial number of servers to 80 ($S = 80$). The algorithm updates every minute ($\Delta = 1min$). This value was selected to allow enough time for the servers to switch on or off before another update occurs. Using the established functions we can plot the overall cost functions for different numbers of servers at the two DCs using the relative price function for the static traffic conditions. Figure 6.4 depicts the number of servers operating at the first DC, n_1, on the x axis, the number of servers operating at the second DC, n_2, on the y axis and the corresponding cost $C(n)$ on the z axis . Figure 6.5 is a close-up of Figure 6.4 and from this we can see that the optimal point is when $n_1 = 50$ and $n_2 = 30$ or $n =$ (50,30).

6.5 Simulation Results

The goal of the simulations is to show that the algorithm converges to the optimal point of the cost function. Figure 6.6 depicts time on the x axis and the number of servers operating at the DC on the y axis as the algorithm operates. Each line represents the number of servers operating at a DC. The number of servers in each DC moves steadily from the initial configuration of $n =$ (40,40) to the optimal point of $n =$ (50,30). This results in the carbon emissions dropping from 39.6g/min to 33g/min. This represents a drop of 16% in carbon emissions with little effect to the QoS.

The rate of convergence to the optimal point is quite slow. The optimal point is reached

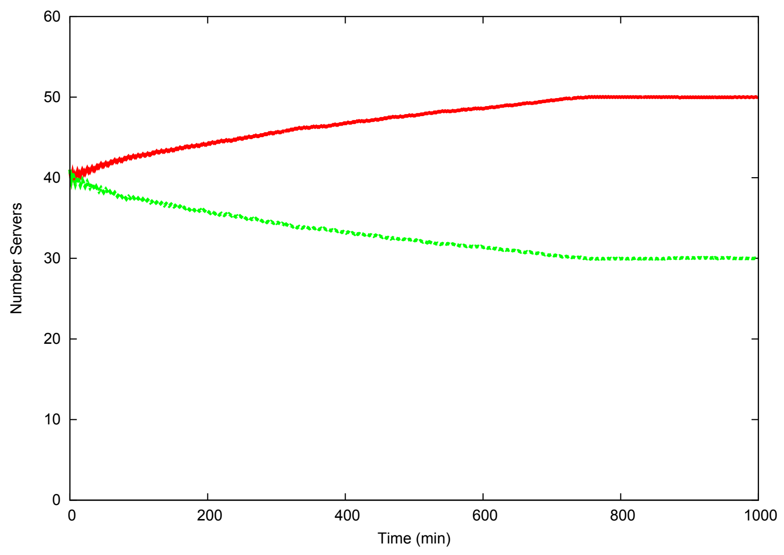


Figure 6.6: Number of servers at the DCs with $a_k = \frac{1}{\sqrt{k}||g(k)||_2}$. The number of servers at each data centre moves steadily from the initial conditions to the optimal point.

after 750 minutes. One reason for this is that the step size was chosen to ensure convergence to the optimal point and prevent large movements in the number of servers operating. In this case the step size was:

$$a_k = \frac{1}{\sqrt{k}||g(k)||_2}.$$

The absolute value of the subgradient $||g(k)||_2$[1] is used to provide a constant step length. We can use a step size that contains an estimate of the optimal cost $C(n^*)$ to improve the rate of convergence. The step size we used to improve the rate of convergence was:

$$a_k = -\frac{0.1(C(n^*) - C(n))}{||g(k)||_2 + 1}.$$

Figure 6.7 depicts time on the x axis and the number of servers operating at the DC on the y axis as the algorithm operates. Each line represents the number of servers operating at a DC. Using the later step size the number of servers operating at each DC converges from the initial configuration to the optimal point more quickly. There are large jumps in the number of servers operating but the optimal point is reached after 10 minutes. Using this step size the rate of convergence is more 70 times greater then that the other step size. This allows the algorithm to perform well in the dynamic environment of the cloud.

[1]The value of $||g(k)||_2$ is calculated as $||g(k)||_2 = \sqrt{g(k)^2}$.

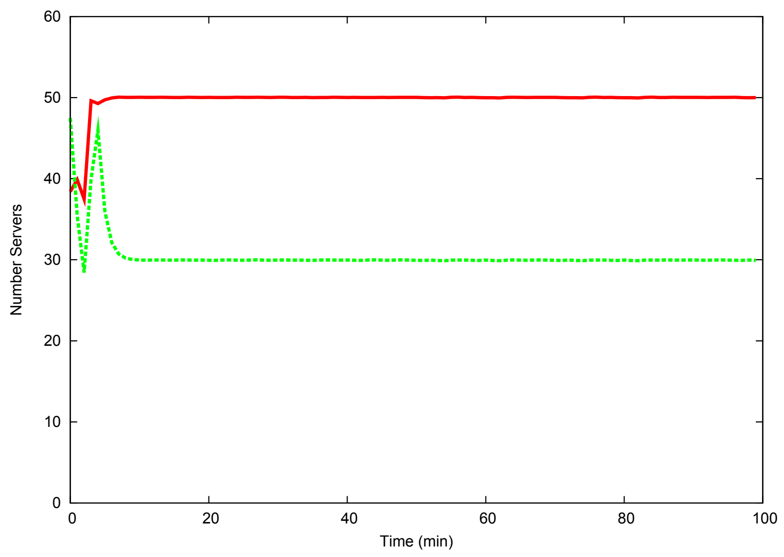


Figure 6.7: Number of servers at the DCs with $a_k = -\frac{0.1(C(n^*)-C(n))}{||g(k)||_2+1}$.

6.6 Summary

A cloud can consist of several data centres in different geographical regions. Each data centre can be powered by an electricity supplier with a different carbon intensity. This can be exploited using load balancing to lower carbon emissions of the cloud. This can, however, have a negative effect on QoS as it can cause the average distance a service request has to travel to increase. In this chapter we formulated this trade-off as a cost function with a relative price function to indicate the importance of the factors to the operator. We then used the subgradient method to minimise this trade-off by directing service requests to the appropriate data centre. Our simulations showed that in particular scenarios carbon emissions can be significantly reduced with little disruption to QoS.

Chapter 7

Unified Global Load Balancing

7.1 Abstract

It is desirable, for latency purposes, to route the traffic to the data centre that is closest in terms of geographical distance, costs the least to power and emits the smallest amount of carbon for a given request. It is not always possible to achieve all of these goals so we model both the networking and computational components of the infrastructure as a graph and propose the Stratus system which utilises Voronoi partitions to determine which data centre requests should be routed to based on the relative priorities of the cloud operator.

7.2 Introduction

Recently the carbon emissions associated with powering DCs have become important due to an increased concern in the carbon footprint of all industries. There have been some proposals to use locally generated clean energy [89] or employ load balancing based upon the carbon intensity of the electricity supplier [90]. These proposals, however, use weather data a metric for load balancing. While this is a useful metric when using locally generated electricity it can be inaccurate when electricity is obtained from an external supplier as other factors also affect their carbon intensity. This is discussed in greater detail in Section 7.4.2. In addition, the proposals do not consider the carbon emitted as a result of packets travelling across the network from the client to the server. While the energy consumed by the networking equipment as part of the cloud computing has been analysed [15], additional analysis is required to examine the total carbon emission caused by a cloud computing system. This load balancing can be achieved with protocol-level mechanisms which are in use today such as dynamically generated DNS responses, HTTP redirection and the forwarding of HTTP requests. All of these have been evaluated thoroughly [33, 94, 119].

Carbon emissions are seldom the sole concern of cloud operators and other factors must be considered. The electricity cost can vary considerably between different geographical regions and this fact can be exploited by cloud operators to lower the overall operational cost.

The manner in which a data centre is cooled can affect both the electricity cost and carbon emissions as certain schemes such as "Free Air Cooling" require less energy and hence emit less carbon. Finally, cloud operators are usually bound by a service level agreement (SLA) and therefore must maintain a minimum QoS for service users.

It is not always possible to achieve the best case scenario for all of these factors as they sometimes conflict, so we formulate a graph-based approach which we call Stratus that can be used to examine and control the operation of the cloud. Stratus uses Voronoi partitions which are a graph-based approach which have been used to solve similar problems in other areas such as robotics [39]. In this chapter we examine how Stratus can be used to achieve a variety of goals.

7.3 Mathematical Preliminaries

7.3.1 Problem Formulation

In this section we formulate the problem. Let $|J|$ be a set of J geographically concentrated sources of requests and $|N|$ be a set of N data centres. Let $|Q|$ be a finite set of points that represent either sources of requests or data centres. These points are connected by E edges in an undirected weighted graph $\mathcal{G} = (|Q|,|E|,|w|)$. The weights are calculated as functions of the time required to service a fraction of the request T_i, the carbon emissions associated with servicing the fraction G_i and the electricity cost E_i if any associated with servicing the request along the edge.

$$w_i = f(T_i,G_i) = T_i + R_1(G_i) + R_2(E_i) \ \ \forall i \in |w|$$

where R_1, R_2 are the relative price functions which are used to specify the relative importance of the factors. It should be noted that the weights of the graph represent the networking and

```
1  $U := P_i(t) \cup P_j(t)$
2  for $x \in U$
3      $W_i := \{x \in U : d(x,i) \leq d(x,j)\}$
       $W_j := \{x \in U : d(x,i) > d(x,j)\}$
4  endfor
5  $P_i(t+1) := W_i$
   $P_j(t+1) := W_j$
```

Figure 7.1: Pseudocode for pairwise partitioning rule

computational aspects of servicing a request.

The set $|Q|$ is partitioned into N subsets representing the regions serviced by each data centre. This results in a collection $P = \{P_i\}_{i=1}^{N}$ of N subsets of $|Q|$ such that:

1. $\bigcup_{i=1}^{N} P_i = Q$

2. $P_i \cap P_j = \emptyset$ if $i \neq j$

3. $P_i \neq \emptyset \ \forall i \in \{1, \ldots, N\}$

4. P_i is connected for all $i \in \{1, \ldots, N\}$

Two subgraphs P_i and P_j are connected if there are two vertices q_i, q_j belonging, respectively, to P_i and P_j such that $(q_i,q_j) \in |E|$.

We can use Voronoi partitions to establish a collection of subsets which minimises the combination of carbon emissions, electricity cost and average request time. In this case the Voronoi partition P_i associated with data centre $i \in |N|$ can be defined as the set of points whose distance to data centre i is less than or equal to the distance to another data centre $j \in |N|$. In order to compute this we need to define how the distance between two points is calculated. A standard notion of distance between two points $d(i,j)$ in a weighted graph is the lowest weight of a path between the two points (i,j). The weight of a path is the sum of the weights of the edges in the path. The goal of using the Voronoi partitions in this scenario is to minimise the distance between the sources of requests and the data centres. This can be defined as:

$$\min \sum_{i=1}^{N} \sum_{j \in P_i} d(i,j)$$

Note if a source is equidistant to more than one data centre the point is assigned to the Voronoi partition that has the least members to attempt to balance the load on the data centres.

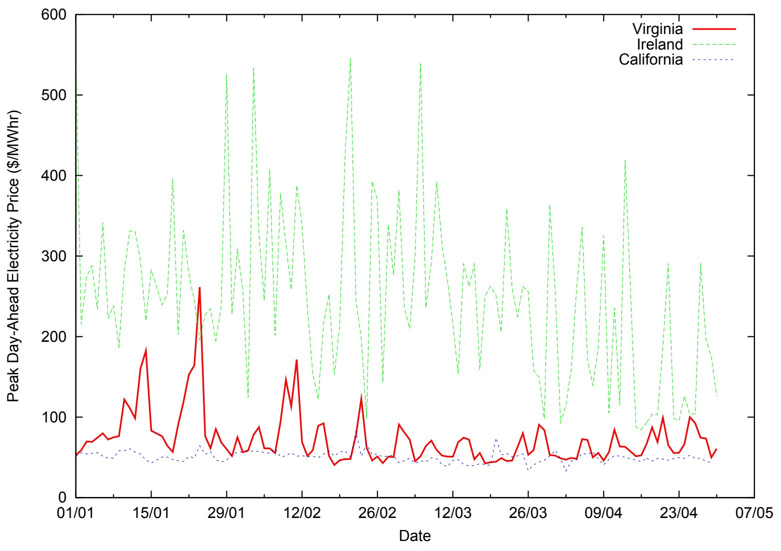


Figure 7.2: Peak Daily price of electricity for suppliers in the regions of the three data centres studied.

7.3.2 Pairwise Partitioning Rule

At time t data centre i and data centre j communicate by exchanging the partitions P_i and P_j so that each data centre can examine all the regions associated with the two data centres to determine if there is a better route available between a data centre and a region. We assume without a loss of generality that $i < j$. Each data centre then performs the actions depicted in the pseudocode in Figure 7.1. The paths between each region and the two data centres are examined. If the path between the data centre i and a region is shorter than the path between the region and the data centre j then the region is added to a temporary partition associated with data centre i. Otherwise it is added to a temporary partition associated with data centre j. The partitions of the two regions are then updated with the appropriate temporary partition. In order to generate the initial partitions the distance between each node in the graph and all the data centres is calculated. The nodes are then added to the partition which yields the minimal distance between the nodes and the data centre.

7.4 Analysis

In this section we examine the variation in the costs that exist between data centres as a consequence of the source of electricity and the cooling architecture used in the cloud.

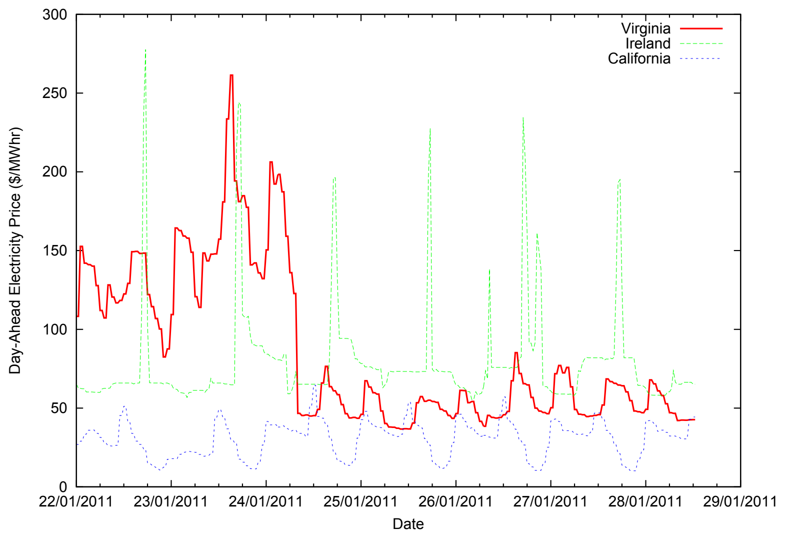


Figure 7.3: Price of electricity for suppliers in the regions of the three data centres studied.

7.4.1 Electricity Cost

The price of electricity on the wholesale market depends on a number of factors. The wholesale electricity market is administered by an authority known as a Regional Transmission Organisation (RTO) in the United States and the Single Market Operator (SEMO) in Ireland. In this market, power producers present supply offers, consumers present bids and an authority determines how the electricity should flow and sets prices. The price is determined based on the bids and offers as well as other factors such as reliability and grid connectivity. The variation of local electricity prices in different geographical regions can be exploited by cloud operators to lower operational costs [123]. To illustrate this we examine the potential savings that can be made by a cloud provider operating a subset of the data centres in Amazons's EC2 [6] cloud. We examine the local prices of electricity suppliers located at the California, Virginia and Ireland data centres. Pacific Gas and Electric (PG&E) is one supplier in the California region and Dominion (DOM) is a supplier in the Virginia region. Ireland uses a single market for electricity known as SEMO and this sets a single price for wholesale electricity. The peak, daily, day-ahead electricity price from these suppliers from January 2011 through April 2011 is depicted in Figure 7.2.

It is interesting to note that the maximum price can approach $550/MWh and that the peak price for the electricity is nearly always greatest in the Ireland region. This would suggest that little traffic would be routed to the Ireland data centre if a load balancing scheme designed to minimise electricity prices was utilised. If, however, we examine the hourly variation of

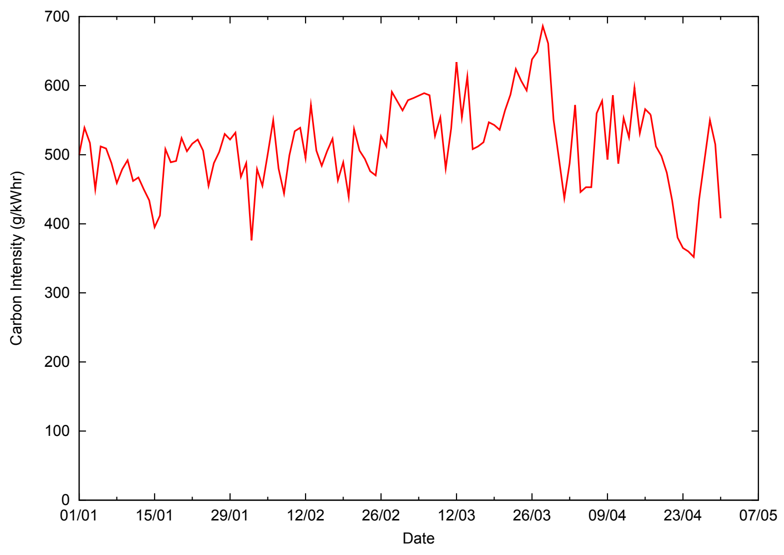


Figure 7.4: Daily peak carbon intensity of electricity supplier in the region of the Ireland data centre studied.

electricity prices we can see that this is not the case. The day-ahead electricity price for the electricity suppliers from the 22nd January 2011 through the 29th January 2011 (an arbitrarily chosen time period) is depicted in Figure 7.3. From this we can see that peaks in electricity price in the Ireland region tend to be very sharp and that at non-peak times the variation in price between geographical regions is much smaller.

7.4.2 Carbon Emissions

An analysis of the carbon intensity of electricity suppliers in various geographical regions is useful when attempting to minimise the environmental impact of a cloud. To illustrate this we examine the carbon emitted by a service which has users in a number of different geographical regions accessing the EC2 infrastructure. The carbon intensity data for the data centres and sources of requests were obtained from the CARMA website [1] and can be seen in Table 7.1. The data for states in the United States were in agreement with data from the US EPA [156]. The carbon intensity of an electricity supplier is calculated using the weighted average (where the power generated by the power plant is the weight used) of the carbon intensity of the power plants operated by the electricity supplier.

The demand for electricity changes over the course of a day and electricity suppliers turn power plants on and off to react to the changes in the demand. A consequence of this is that the carbon intensity of an electricity supplier varies over time. To the author's knowledge, the

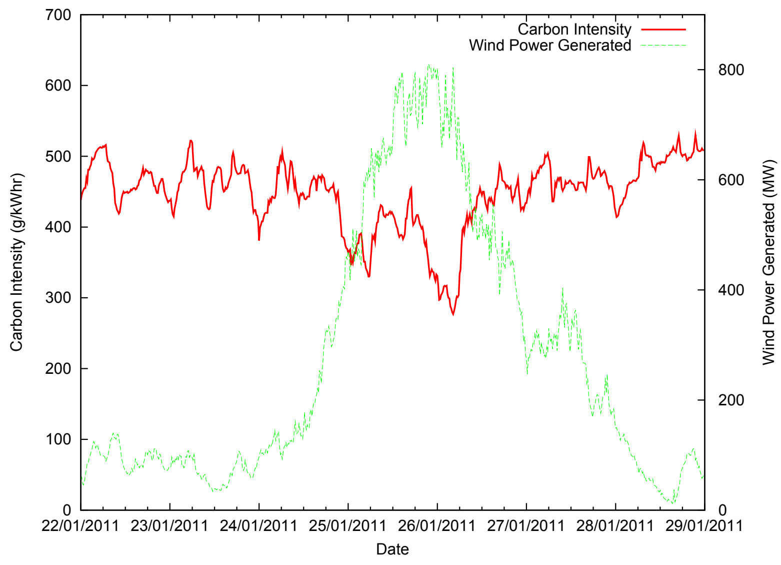


Figure 7.5: Carbon intensity and generated wind power of electricity supplier in the region of the Ireland data centre studied.

realtime carbon intensity of all the geographical regions in Table 7.1 is not publicly available. It is, however, available for the Ireland region. Figure 7.4 depicts the daily peak carbon intensity of the electricity supplier in Ireland from January through April 2011. This data was obtained from the Ireland Transmission System Operator Eirgrid [40]. We can see that there is a large variation with time. This suggests that the data can be exploited to minimise the environmental impact of the cloud. Figure 7.5 depicts the carbon intensity of the SEMO suppliers from the 22nd January 2011 through the 29th January 2011. The interval between data points is fifteen minutes. From Figure 7.5 we can see that the carbon intensity is not as volatile as the electricity market price but varies enough to allow the cloud operator to utilise the realtime data to minimise the environmental impact.

A novel aspect of our approach to minimising carbon emissions when compared with other approaches [90, 101] is that we use carbon intensity data rather than weather data when determining where to route load. This does not affect schemes where power is generated locally by the cloud operator but it can have a significant effect when cloud operators draw power from an external electricity supplier for two reasons. Firstly it is not always possible to utilise solar and wind power. An electricity network must carefully balance supply and demand and ideally the market authority would use the cleanest power plants available to meet the demand. This, however, cannot be achieved in reality as it would required power plants to be able to turn on or off in very short spaces of time and some power plants (e.g. coal) take a long time to turn

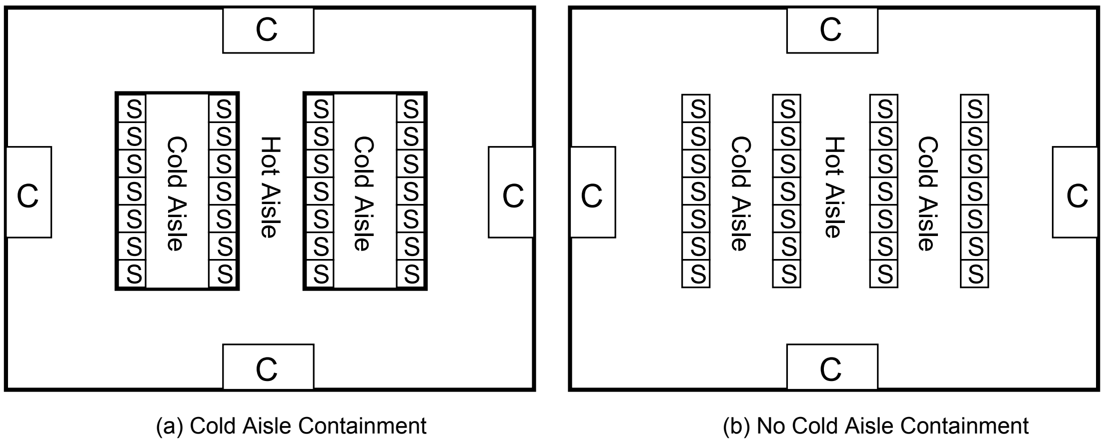


Figure 7.6: Layout of cooling cost simulations with (a) cold aisle containment and (b) no cold aisle containment.

on or off. The result of this is that they are very rarely turned off and if there is insufficient demand power is wasted.

The second reason that weather data can be an inaccurate metric is that even if there is sufficient demand and solar and wind power is utilised the changes in the operation of other power plants can affect the carbon intensity. As a result there is not a direct correlation between availability of wind and solar power and carbon emissions. For example if a pumped storage plant is turned on and the wind speed drops carbon intensity may still go down as the reduction in carbon emissions caused by the use of the pumped storage plant may be greater than the increase in carbon emissions which is caused by other power plants supplying the electricity which is no longer delivered by the wind turbines. If we examine Figure 7.5 we can see an example of this. There is some correlation between the wind power generated and carbon intensity but it is not direct. Sometimes when the wind power generated increases the carbon intensity also increases.

7.4.3 Cooling Cost

Cooling costs for a data centre are dependent on its design and the local climate in addition to the load placed upon it. If a data centre uses aisle containment [99] it can significantly reduce the cost of cooling the data centre. Aisle containment is the separation of the inlets and outlets of servers with a barrier such as PVC curtains or Plexiglas [17] in order to prevent air migration which adversely affects cooling costs.

In addition "Free Air Cooling" can be used. This is the use of air economizers to draw

Table 7.1: Average round trip time between data centres and sources of requests, carbon intensity of data centres and sources of requests and daily number of requests at source

Region	California (ms)	Ireland (ms)	Virginia (ms)	Carbon Intensity (g/kWhr)	Number of Requests (Millions)
Austria (AUS)	177.98	47.67	159.07	870	7.038
Belgium (BEL)	171.98	28.45	158.09	317	11.736
California (CAL)				384	
Colorado (COL)	42.77	155.36	101.27	903	6.76
Connecticut (CON)	88.05	117.65	75.73	392	4.293
Finland (FIN)	188.47	55.77	176.72	99	5.418
Florida (FLO)	54.26	171.87	98.21	762	26.365
France (FRA)	192.44	21.24	184.75	96	61.355
Georgia (GEO)	58.91	115.12	77.43	694	1.968
Germany (GER)	177.74	40.89	157.68	612	58.76
Illinois (ILL)	63.81	142.78	102.08	544	18.049
Indiana (IND)	69.65	151.05	83	986	7.803
Ireland (IRE)				655	
Italy (ITA)	188.71	44.71	167.3	473	55.372
Kansas (KAN)	50.48	148.4	85.2	817	4.545
Kentucky (KEN)	71.98	146.85	87.29	968	5.169
Maryland (MAR)	99.71	140.46	88.05	641	6.69
Massachusetts (MAS)	89.33	98.02	72.1	603	9.602
Minnesota (MIN)	62.09	147.74	85.79	744	6.724
Netherlands (NET)	163.93	19.71	138.79	548	15.527
New York (NEW)	96.45	78.71	134.11	386	27.604
North Carolina (NCA)	72.32	72.45	31.12	604	11.817
Norway (NOR)	194.82	48.84	183.08	6	6.69
Ohio (OHI)	83.81	132.08	69.17	873	14.828
Oklahoma (OKL)	46.42	159.96	98.22	819	4.378
Ontario (ONT)	90.84	142.81	97.12	224	16.64
Oregon (ORE)	27.66	213.48	153.28	246	4.807
Pennsylvania (PEN)	71.99	118.53	52.78	597	16.097
Portugal (POR)	222.69	64.11	190.02	550	10.925
Spain (SPA)	194.55	35.83	172.47	487	40.633
Sweden (SWE)	186.01	48.79	170.28	19	11.887
Tennessee (TEN)	235.61	276.43	311.61	661	7.891
Texas (TEX)	37.61	151.93	98.96	763	31.015
UK (UK)	175.59	17.62	163.25	614	78.647
Virginia (VIR)				559	
Washington (WAS)	29.57	192.11	126.53	938	10.119
Wisconsin (WIS)	67	146.49	94.03	834	7.025

in cold air from the environment into the data centre when the climate conditions are suitable, displacing the use of computer room air conditioner (CRAC) chiller units and lowering the cooling costs [12]. Water cooling [130, 5] is an option but it is rarely used in data centres at present.

In order to examine how data centre cooling cost varies with demand we constructed two models of data centres in the computational fluid dynamics (CFD) simulation software Flovent [34]. These represent typical data centres which have been examined in previous research [134, 103]. One data centre used cold aisle containment and the other does not. Apart from this the data centres were of similar construction. Each data centre has dimensions 11.7m $\times$ 8.5m $\times$ 3.1m with a 0.6m raised floor plenum that supplies cool air through perforated floor tiles. There are four rows of servers with seven 40U racks in each case, resulting in a total of 1120 servers. The servers simulated were based on Hewlett-Packard's *Proliant DL360 G3s* model, which consumes 150kW of power when idle and 285kW at 100% utilization. From this we can determine that the total power consumption of the data centre is 168kW when idle and 319.2kW at full utilisation. For cooling, the data centre is equipped with four CRAC units. Each CRAC unit pushes air chilled to 15°C into the plenum at a rate of 16,990$\frac{\mathrm{m}^3}{\mathrm{h}}$. The cooling capacity of the each CRAC unit is limited to 90kW, and the fans of each CRAC unit consume 10kW.

The layout of the two data centres modelled is shown in Figure 7.6. Racks of servers are represented as boxes with the letter "S" and CRAC units can be identified as boxes with the letter "C". The simulations are used to establish the maximum inlet temperature of a server rack T_{max} under different load conditions. We can use this to establish the cooling costs C which can be calculated as follows:

$$C = \frac{Q}{COP(T_{sup} + (T_{safe} - T_{max}))} + P_{fan} \tag{7.1}$$

Where Q is the amount of power the servers consume, T_{sup} the temperature of the air that the CRAC units supply, T_{safe} the maximum permissible temperature at the server inlets in order to prevent equipment damage, T_{max} the maximum temperature of the server inlets in the data centre, P_{fan} the power required by the fans of the CRAC units and COP is the "coefficient of performance" (COP). Recall that the COP is the ratio of heat removed to work necessary to remove the heat and it is a function of the temperature of the air being supplied by the CRAC

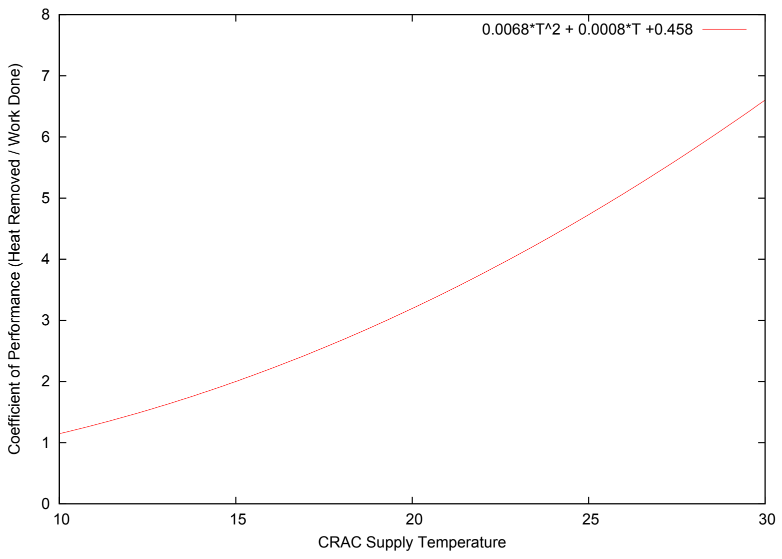


Figure 7.7: Typical model of the Coefficient of Performance (COP) curve for a chilled water CRAC unit.

unit. The COP of a typical chilled-water CRAC unit used in the calculations of cooling costs is depicted in Figure 7.7. We assume a T_{safe} value of 25°C.

Figure 7.8 depicts the results of the cooling cost simulations. The percentage utilisation of the data centre is shown on the x axis and the cooling cost in kilowatts is shown on the y axis. Each line represents a different system. The CAC-CRAC line is a system which uses cold aisle containment and CRAC cooling. The FAC line is a system uses “Free Air Cooling”. The NCAC-CRAC line is a system which does not use cold aisle containment and CRAC cooling. “Free Air Cooling ” only consumes fan power and is therefore constant.

7.4.4 Average Job Time

The previous sections establish that electricity cost and carbon emissions can be lowered. It is likely, however, that there will be an increase in the average service request time which the cloud operator will have to take into account when determining its load balancing policy. The average service request time is determined by a combination of resource usage and latency which is the time required for packets to travel along the links from the client to the server and back again. Anecdotal evidence suggests that traffic levels can be seriously affected by latency [88] so cloud operators should monitor this carefully to ensure user satisfaction. To establish the round trip time data we conducted an experiment on PlanetLab [30] which established a server at each node location by requesting a PlanetLab slice on servers at locations from Austria to

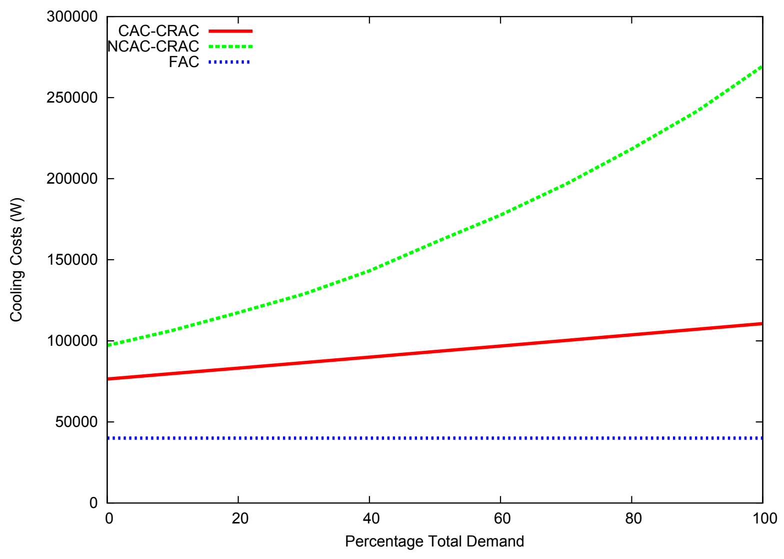


Figure 7.8: Cooling cost of various data centre cooling systems at various levels of demand

Wisconsin. We wrote scripts causing nodes in the same region as our three data centre locations to ping the other geographical regions at fifteen minute intervals for approximately two days. The average latency established from this experiment can been seen in Table 7.1.

Average service request times could be reduced by routing load from a geographical region to the data centre region using lowest latency as a criterion to route load. From Table 7.1 we can see that if such a load balancing scheme was used, each data centre region will have some of the load routed to it as each data centre has the lowest latency for some of the regions. From this we can conclude that any reduction in carbon emissions or electricity cost will likely cause an increase in the average latency as load will not be routed with latency as the sole metric.

Figures 7.9, 7.10 and 7.11 depict the measured latency between the 34 regions and the California, Ireland and Virginia data centres respectively. It is interesting to note that the latencies remain mostly constant over time at the California and Ireland data centres but vary frequently at the Virginia data centre. We postulate that this is a result of congestion at the Virginia region. In addition, we can see that there is some variation in the latency between the other regions and the California and Ireland data centres. Thus we can conclude that latency varies with time particularly in regions where congestion takes place and it should be monitored so that the increase in average service request time caused by reducing carbon emissions or electricity cost can be measured correctly. Indeed there may be times, where there is no increase in average service request time associated with a reduction in carbon emissions

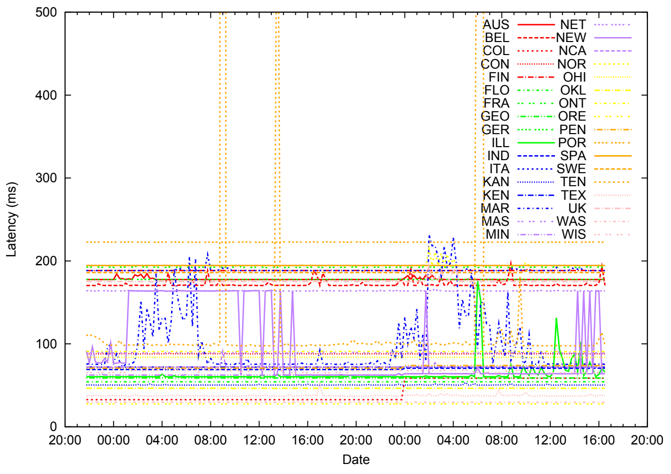


Figure 7.9: Latency between California and different geographical regions at fifteen minute intervals.

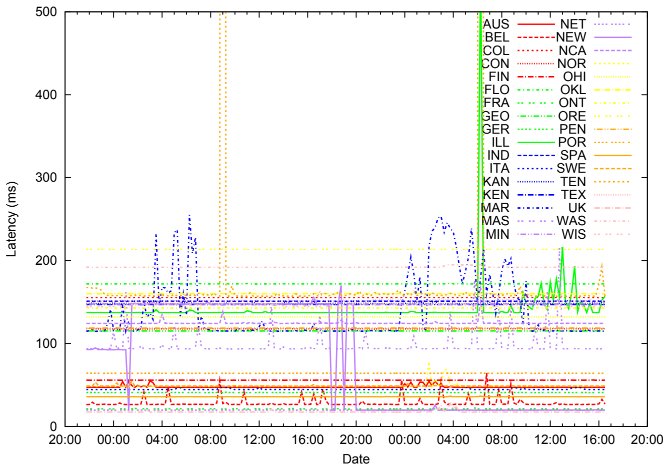


Figure 7.10: Latency between Ireland and different geographical regions at fifteen minute intervals.

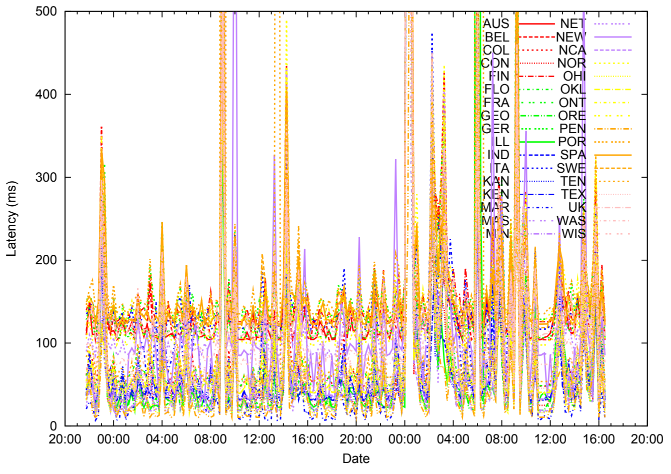


Figure 7.11: Latency between Virginia and different geographical regions at fifteen minute intervals.

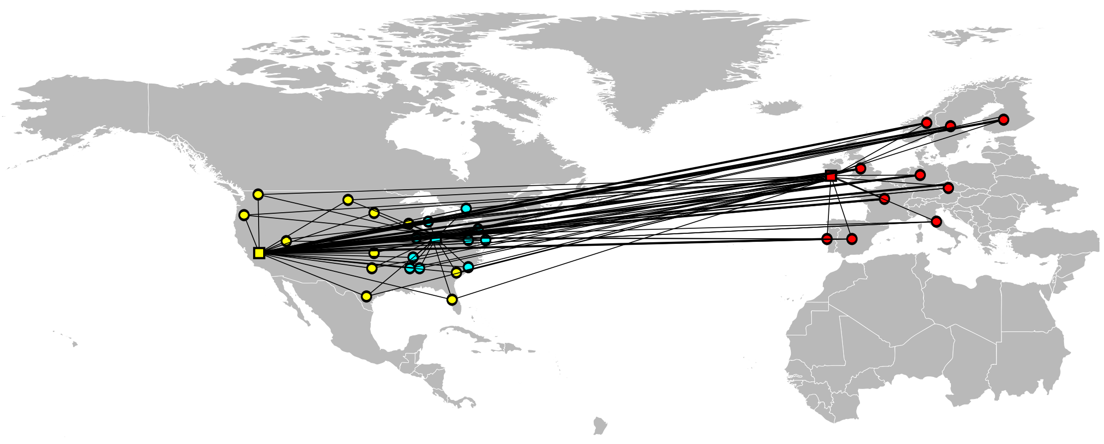


Figure 7.12: Diagram of the simulation setup. The colour of the node indicates that the node is part of a particular partition.

or electricity cost.

7.5 Simulation Setup

In this section we describe the setup for the simulation of the algorithm described in Section 7.3.2 and our methodology for establishing the weights of the graph described in Section 7.3.1. We simulate three data centres. One of these is in Ireland and the other two are in the United States in Virginia and California. We chose these locations to mimic Amazon's EC2 platform [6] which currently has major data centres at these locations. We model 34 sources of requests in the simulation which represent certain countries in Europe, states in the United States and provinces in Canada. Each source is connected to each data centre by a single edge. This is illustrated in Figure 7.12.

To calculate the weights of the edges of the graph we needed to determine the time, carbon emissions and electricity cost associated with servicing the networking and computational portions of a request. For the networking portion of a request, we firstly assume that each service request requires the transfer of relatively small amount of data and therefore the duration of the connection can be approximated by the round trip time. To calculate the time associated with the computational portion of the request we assume that each request requires 50ms of computation.

In order to calculate the carbon emissions and electricity cost of serving a request we assume that the data centre uses Hewlett-Packard's *Proliant DL360 G3s*. This type of server consumes 150W at 0% utilisation and 285W at 100% utilization. This yields dynamic power of 135W for each server. This is then multiplied by the time required to service the computational portion of the request (50ms) to yield the energy required to service a request.

We must then consider the energy required for the additional cooling required by servicing the requests. We assume that the Ireland data centre uses “free air cooling”, the Virginia data centre uses cold aisle containment and the California data centre uses a standard cooling system with no cold aisle containment. The cooling energy required by the data centre when it is not processing the request is subtracted from the energy required when it is processing the request to give the cooling energy caused by the request. This is added to the energy already calculated to yield the total computational energy. The total computational energy is then multiplied by the electricity price to yield the electricity cost E_i. The energy is also multiplied by the carbon intensity of the data centre to yield the computational carbon emitted.

The networking aspect of the weights must also be considered. The power consumed by a switch can be altered by powering off (disabling) ports when they are not in use and powering on (enabling) ports when they need to be used. In order to calculate the carbon emissions associated with servicing the networking portion of the request we assumed that only two ports would open during the duration of the request. To calculate the carbon emitted we first obtain the energy consumed by multiplying the duration of the round trip by a power value required to open a port (0.7W). This value was an intermediate value of those presented in [93]. The energy consumed is then multiplied by the average carbon intensity at the source of the request and at the data centre to obtain the network carbon emitted. This is added to the computational carbon to give the total carbon emitted and the carbon weight G_i. While regulation is likely to hold cloud operators at least partially responsible for the carbon emissions of the networking equipment, current modes of operation suggest that they are not held responsible for the vast majority of the electricity cost of this component and as such it is ignored.

In all simulations the algorithm runs for a two day period using the latency, electricity price and carbon intensity data described in Section 7.4. The latency between the sources of requests and the data centre is as seen in Figure 7.9, 7.10 & 7.11. The electricity price is as seen in the first two days of Figure 7.3 for the three data centres. The carbon intensity data is as seen in Table 7.1 for all the regions except Ireland which uses the data seen in the first two days of the January period shown in Figure 7.5. The algorithm updates every fifteen minutes. It should be noted that we assume that the bandwidth which connects the data centres to the internet is such that redirecting additional requests to each data centre does not cause congestion or affect the average service request time.

In order to examine the overall costs to the cloud we must consider the number of requests coming from each source. We used figures from the websites [139, 70] which estimated the number of Facebook users in each source location and assumed that the daily average number of service requests from a single user was 2.6. We used Facebook as it is representative of a broad range of cloud applications. The daily number of requests for each source can be seen in Table 7.1. We also needed to establish the number of requests at each source during each fifteen minute interval over the two day period. It has previously been found that realistic workloads have a diurnal cycle with a trough at approximately 6:00am and a peak of roughly four times the trough value at approximately midnight [123]. We divided the daily number of requests at each source into this diurnal pattern and adjust the peaks to match the time difference of the region. The total demand can be seen in Figure 7.13.

In the first set of simulation we examine the extremes of the algorithm by looking at four scenarios. In the first scenario we set the relative price functions to zero. This represents a situation where time is crucial and the operator is attempting to minimise the time taken to service a request with no regard for electricity cost and associated carbon emissions $R_1(G_i) = 0$,$R_2(E_i) = 0$ and the weights of the edges of the graph become $w_i = T_i$. We shall hereafter refer to this scenario as "Time Focused".

In the second scenario we set the first relative price function to ten thousand times the carbon emissions. This essentially functions by making the carbon emission the solely important factor. This represents a scenario where time and electricity cost are unimportant and all efforts can be made to reduce the associated carbon emissions $R_1(G_i) = 10000G_i$,$R_2(E_i) = 0$. The weights of the edges of the graph become $w_i = T_i + 10000G_i$. We shall hereafter refer to this scenario as "Carbon Focused".

In the third scenario we set the second relative price function to ten thousand time the electricity cost. This represents a scenario where time and carbon emissions are unimportant and all efforts must be made to lower the electricity costs $R_1(G_i) = 0$,$R_2(E_i) = 10000E_i$. The weights of the edges of the graph become $w_i = T_i + 10000E_i$. We shall hereafter refer to this scenario as "Electricity Focused". In the final scenario we examine a baseline for current load balancing operations by examining a round robin scheme. This is the default option in many commercial load balancing solutions. We shall hereafter to refer to this scenario as

Table 7.2: Average Service Request time, Daily Carbon Emission and Number of Requests Serviced at Each DC for Various Scenarios

Scenario	Average Service Request Time (ms)	Carbon Emissions (kg)	Electricity Cost ($)	Number of Requests Serviced by California (million)	Number of Requests Serviced by Ireland (million)	Number of Requests Serviced by Virginia (million)
Time Focused	90	1378	240	241	822	302
Carbon Focused	132	1200	195	0	1365	0
Electricity Focused	177	1567	100	1276	0	89
RoundRobin	170	1522	257	455	455	455

"RoundRobin".

In the second set of simulations we explore scenarios where the cloud operator needs to strike a balance between the three factors. In this set of simulations we examine the performance of the algorithm under scenarios which attempt to balance the various factors by adjusting (α,β) $R_1(G_i) = \alpha G_i$,$R_2(E_i) = \beta E_i$ in intervals of 100 from 0 to 10000 and examining the total electricity cost, carbon emissions and average service request time of each scenario to examine what savings in electricity cost and carbon emissions can be made when there are constraints on the average service request time.

7.6 Simulation Results

It this section we examine the results of the two sets of simulation carried out. Subsections 7.6.1 to 7.6.3 examine the first set of simulations while the results of the second set of simulations are dealt with in subsection 7.6.4. Subsection 7.6.1 investigates the overall performance of the algorithm under the various scenarios. In the subsection 7.6.2 we examine the request distribution under the various scenarios to inspect how performance improvements are achieved. Subsection 7.6.3 examines how the performance of the algorithm changes over time. Subsection 7.6.4 examines the performance of the algorithm when the cloud operator is attempting to strike a balance between the three factors.

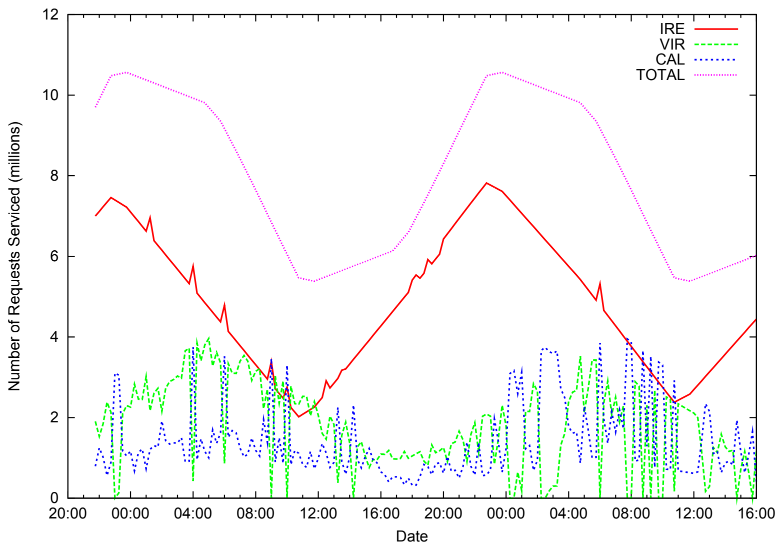


Figure 7.13: Number of requests serviced at each data centre when the “Time Focused” scenario is used. It shows the UTC+0 time zone.

7.6.1 Overall Performance

The key performance metrics for the simulations are the average service request time, the electricity cost and the carbon emissions associated with servicing requests for the two day period studied. We examine these key performance metrics for the first set of simulations. They are shown for each scenario in Table 7.2. When comparing the carbon focused scenario with the roundrobin baseline we can see that carbon emissions for a service can be reduced by 21%. If we examine the electricity focused scenario and the roundrobin baseline we can see that the electricity cost can be reduced by 61%. There is, however, a corresponding increase in the average service request time of 7ms. If we investigate the time focused scenario and the roundrobin baseline we can see that the average service request time can be reduced by 47%. It is also interesting to compare the three focused scenarios. If we compare the time focused scenario and the carbon focused scenario we see that the latter emits 13% less carbon but has an average service request time that is 42ms higher. If we examine the time focused scenario and the electricity focused scenario we can see that the latter costs 58% less but has an average service request time that is 87ms higher. These comparisons are useful for the cloud operator as it allows them to see if the scenarios are feasible under SLAs and whether it is more desirable to concentrate on lower electricity costs or carbon emissions.

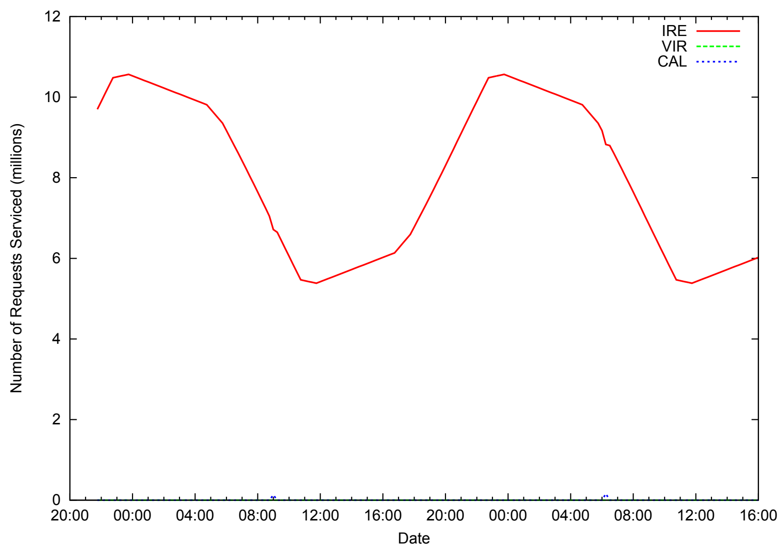


Figure 7.14: Number of requests serviced at each data centre when the “Carbon Focused” scenario is used. It shows the UTC+0 time zone.

7.6.2 Request Distribution

From Table 7.2 we can also see the number of requests sent to each data centre under the various scenarios. We can see that under the carbon focused scenario all of the requests go to the Ireland data centre. This is interesting as the carbon intensity for the California region is relatively low. The additional carbon caused by the cooling setup used in the data centre is sufficiently high that all the requests go to the Ireland data centre. We can also see that under the electricity focused scenario most of the load goes the California data centre. The additional electricity cost of the cooling setup at California is insufficient to overcome the local electricity price differential and the requests are driven to the California data centre. We also examine the number of requests serviced at the data centres at each time interval in Figure 7.13. We can see that the number of requests serviced in Ireland follows the change in demand relatively steadily while the number of service requests for Virginia and California fluctuates. The reason for this that Ireland will attract most requests from the European regions because the latency remains relatively steady. California is likely to attract requests which would otherwise go to Virginia if there is congestion as the latency between these regions and the California data centre is lower than that between the regions and Ireland. Figure 7.14 depicts the number of requests serviced at each data centre during the time period studied for the carbon focused scenario. From Figure 7.14 we can see that the all requests go to Ireland for all the time intervals. Figure 7.15 depicts the number of requests serviced at each data centre during time period studied for

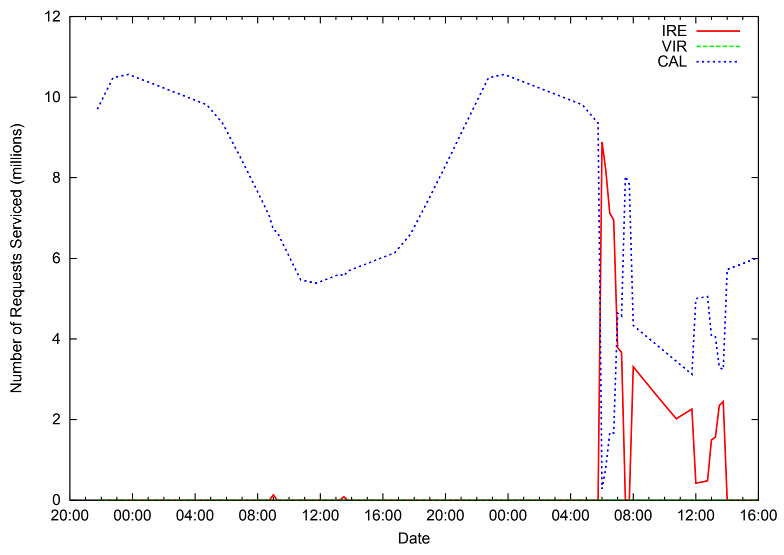


Figure 7.15: Number of requests serviced at each data centre when the "Electricity Focused" scenario is used. It shows the UTC+0 time zone.

the electricity focused scenario. From Figure 7.15 we can see that California services all the requests for the first part of the time period studied and in second part the electricity costs of the Ireland and California become almost equal so that some of the requests go to the California data centre and some go to the Ireland data centre. Finally, all the requests get serviced by California as the prices diverge.

7.6.3 Performance Variance

Figure 7.16 depicts the carbon emitted under each scenario over the same time period. From Figure 7.16 we can see that the carbon emitted is greatly affected by overall number of service requests at a given time under all scenarios. This is as we expected as more service requests require more energy which increases the carbon emissions. We can also see there are no spikes in the emissions which is as we expected since Figure 7.5 shows that the change in carbon intensity over time is gradual. Finally, we can see that the difference between the schemes is quite small. The carbon intensities of the three data centres are relatively similar. Ireland's ranges from 369g/kWhr to 522g/kWhr, Virginia has a carbon intensity of 559g/kWhr and California comes in at 384g/kWhr but this is offset by the cooling setup we chose for that location. The difference in carbon intensities between regions can be dramatic. For example Norway with its high level of hydropower has a carbon intensity of 6g/kWhr while Austria has a carbon intensity of 870g/kWhr.

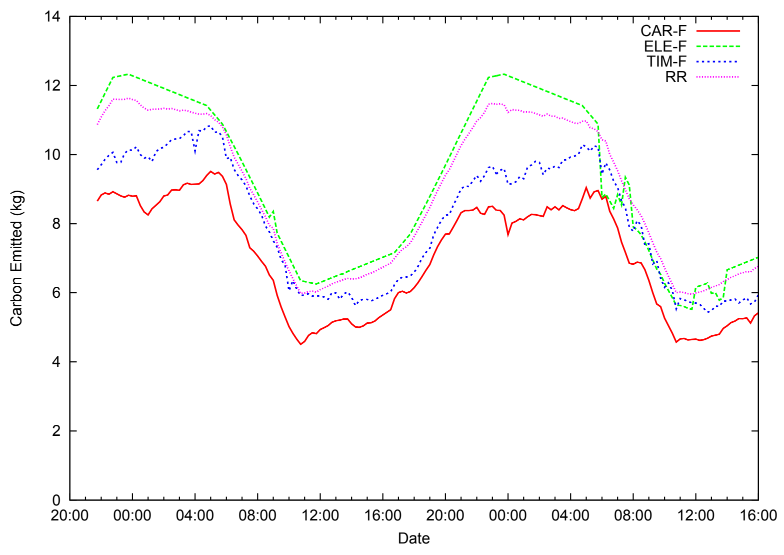


Figure 7.16: Carbon Emitted at each time interval under a variety of scenarios. It shows the UTC+0 time zone.

Figure 7.17 depicts the electricity cost under each scenario over the time period. We can see that the difference between scenarios can be quite large particularly when there are peaks in the electricity cost at the data centre where requests are being serviced. The difference can also be quite small as we can see that the electricity cost of the carbon focused scenario and the electricity focused scenario are effectively the same towards the end of the time period. It should be noted that the while the overall electricity cost is quite low it is the relative differences in the electricity price that are the most important. In our model we assumed that a single server performs all the computation required for a single request. While it is possible for cloud services to use this approach, latency considerations frequently result in a Partition/Aggregate design pattern being used [4]. In this approach a request is broken into pieces which are then farmed out to worker servers. The responses of the workers are combined together by aggregator servers to yield the result responses. In this design tens or even hundreds of servers can be used to process a single request. Energy consumption for a request is significantly higher as tens of servers are operating simultaneously and the overall electricity cost would be significantly higher. We chose not to model the requests in this fashion as without trace data it is difficult to simulate this design accurately. Each worker server frequently operates for different lengths of time and therefore the individual energy consumed will be different.

Figure 7.18 depicts the average service request time under the various scenarios. From Figure 7.18 we can see that the carbon focused and electricity focused scenarios have average

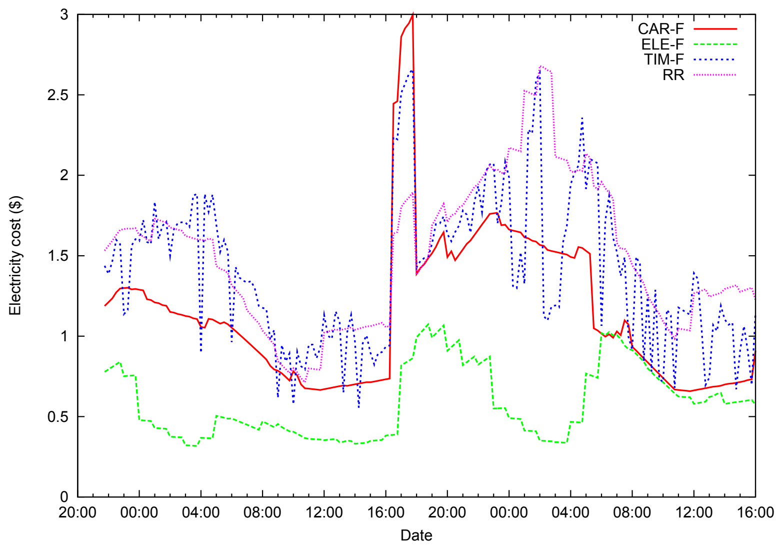


Figure 7.17: Electricity Cost at each time interval under a variety of scenarios. It shows the UTC+0 time zone.

service request times that fluctuate smoothly. This is a result of not using the Virginia data centre to service the load as its congestion causes the average service request time to change frequently. The reason for the change in average service request time for the carbon focused and electricity focused scenarios is the changes in demand in the regions. If there are more requests in from a region with high latency at a given time interval this will increase the average service request time. This does not occur in the roundrobin and time focused scenarios as the requests are spread among the three data centres and the effect is diluted. It is important that the cloud operator considers this so that SLAs are not violated.

7.6.4 Balanced Performance

We now examine the second set of simulations. Figure 7.19 depicts the total carbon output as we vary α and β. From Figure 7.19 we can see that as we increase α the carbon emissions decrease and that as we increase β the total carbon emissions increase. The total electricity cost as we vary α and β is depicted in Figure 7.20. We can see that as we increase α the electricity cost increases and as we increase β the electricity cost decreases in Figure 7.20. Figure 7.21 depicts the average service request time as we vary α and β. From Figure 7.21 we can see that as we increase α and β we increase the average service request time. The increase is more severe in β's case but this is a result of the particulars of the simulations as the average latency between all the regions and California is higher than the average latency between all the regions

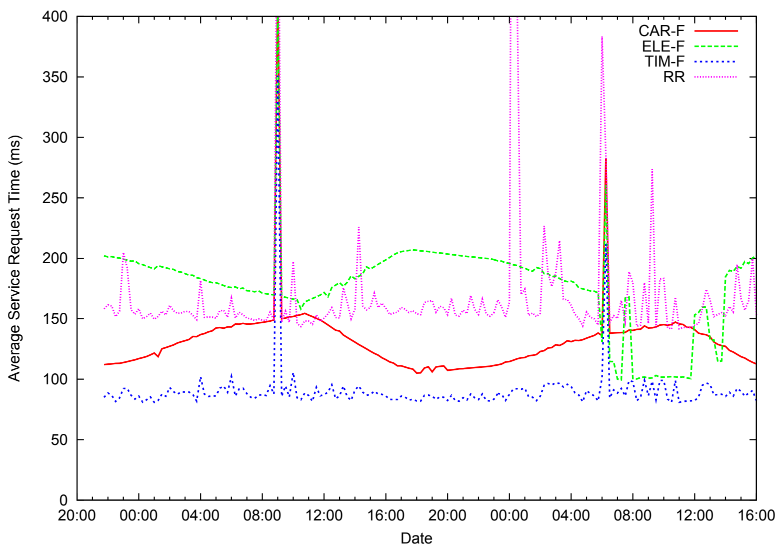


Figure 7.18: Average service request time at each time interval under a variety of scenarios. It shows the UTC+0 time zone.

and Ireland.

The selection of α and β are of crucial importance to the operation of this algorithm. Ultimately the selection of these values will depend on the SLAs the cloud operator has agreed to. Savings in electricity cost and reductions in carbon emissions can only be achieved if SLAs can still be maintained. The selection of whether to lower carbon emissions or electricity cost will depend on whether the cloud operator is under any regulation to limit its carbon emissions, public relations pressures or the price of carbon on carbon trading schemes. Our first set of simulations has shown the average service request time will vary with time. From this we can conclude that α and β will also vary with time. It would, however, be relatively simple to alter the algorithm so that α and β are adjusted at each time interval and this is left for future work.

7.7 Summary

A cloud can consist of several data centres in different geographical regions. There are several factors which will affect the operational costs. The electricity price and associated carbon emission can vary significantly. These factors are also affect by the cooling system used. Finally, the latency between the data centres and clients will differ at each location. In our Stratus system we modelled these factors using Voronoi partitions and use a pairwise partitioning rule as a distributed algorithm to attempt to lower electricity prices, carbon emissions and average

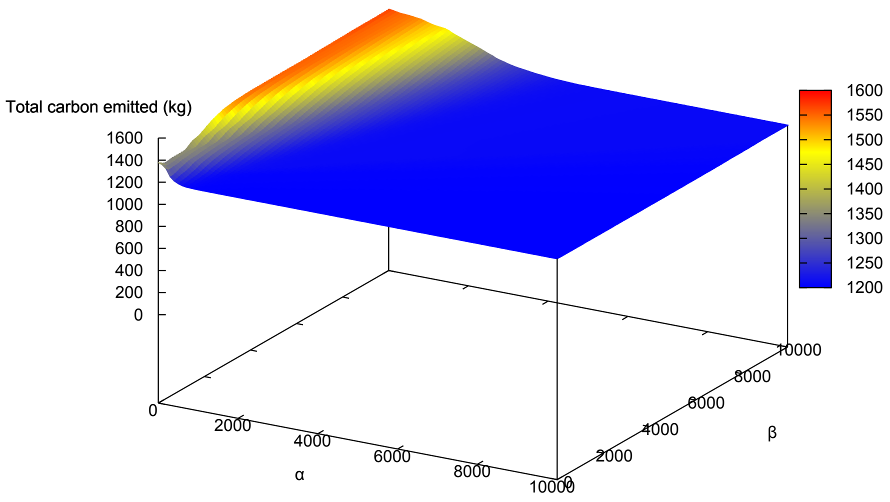


Figure 7.19: Total carbon output with varying relative price functions.

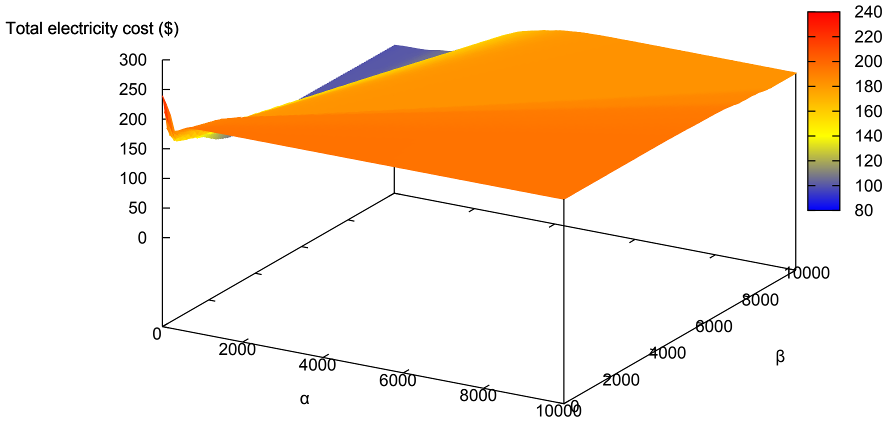


Figure 7.20: Total electricity cost with varying relative price functions.

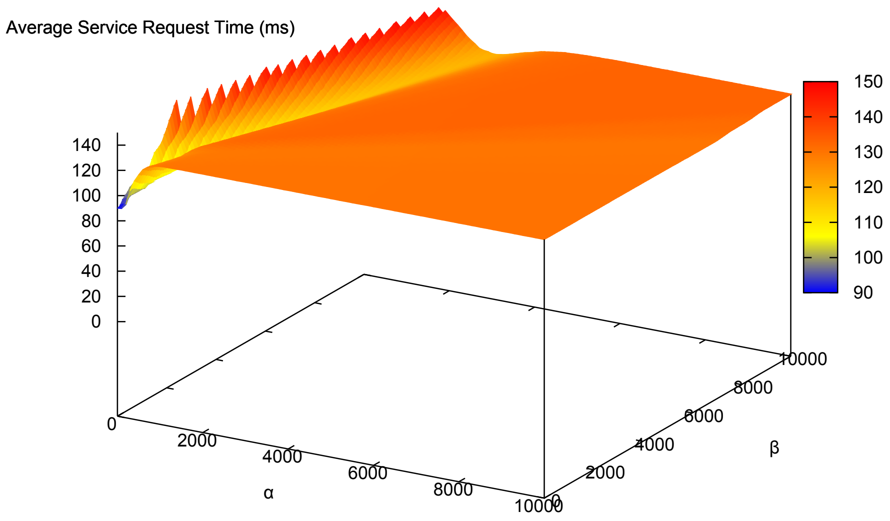


Figure 7.21: Average service request time with varying relative price functions.

service request time. It is not always possible to achieve the best case scenario for all of these factors as they sometimes conflict but operators can specify their relative importance to allow the operator to improve the performance of the salient factor.

We investigated electricity prices, carbon emission, cooling system and latency of a cloud similar to part of Amazon's EC2 cloud. From this we saw that there is a significant difference between electricity prices, carbon intensity and latencies which can exploited to improve overall performance of the cloud. In our first set of simulations we inspected three scenarios where the operators were attempting to achieve the lowest carbon emissions, electricity cost and average service request time respectively. We compared this with a standard load balancing technique. In all cases significant improvements were made. Finally, we examine the performance of the algorithm when the operator is attempting to strike a balance between the three factors. A balance between the three factors can be achieved by setting the relative price function appropriately.

Chapter 8

Conclusions and Future Work

Large public clouds with data centres in different geographical locations can provide reduced latency, increased data transmission rates and improved redundancy. The data centre must be operated correctly to achieve these goals as mismanagement can lead to the opposite effect.

The construction of a cloud is a substantial cost and the operating cost is also large. The goal of this work is to show how cloud operators can lower their operational cost and use predictable, incremental fixed cost pricing policies to recover their capital investment. Load balancing and rate limiting can be used by the cloud operator to achieve these goals. Rate limiting can be used to offer a predictable incremental fixed cost policy by placing an artificial limit on the rate of traffic. Load balancing can be used to lower operating costs by directing load to the data centre which lowers the overall cost.

In this dissertation we firstly examined a proposal for enforcing predictable, incremental fixed cost pricing schemes for traffic using DRL. The aim of this investigation was to show that the algorithms enforced the bandwidth limit and divide bandwidth fairly among the flows of the cloud user. The algorithms also needed to be resilient to failure and able to function in the dynamic cloud environment so that cloud operators could adhere to SLAs. Our results showed that our implementation of the algorithms and the enhancements performed well at all of these goals. The algorithms could be used to implement a incremental, predictable fixed cost pricing policy and hence increase utilisation.

Secondly, we proposed the use of the "fair-share" dropping mechanism in a bandwidth management system to distribute bandwidth fairly to both the users of the clouds and the users of the services provided from the cloud. Our experimentation clearly demonstrated that the "fair-share" dropping mechanism distributes the bandwidth fairly and uses less resources than other dropping mechanisms. This makes it a superior dropping mechanism for use in bandwidth management schemes in data centres.

In this dissertation we examined three ways in which load balancing can be used to lower operating costs. Firstly, we considered thermal management in data centres. Cooling costs are dependent on the highest server inlet temperature inside the data centre. This results in some servers being excessively cooled. In this work we proposed a consensus algorithms which equalises temperatures by altering the demand serviced by a machine. Our simulations and experimentation showed that power required for cooling and hence the cost can be reduced using these algorithms.

Secondly, we inspected the carbon emissions of clouds. The carbon intensity of electricity suppliers varies in different geographical regions. We proposed an algorithm to minimise a cost function which represents trade-off between average service request time and carbon emissions. Our results showed that the carbon emission can be reduced by 16% with little effect on the average service request time. This algorithm could be used to lower operational costs if carbon trading schemes have been implemented

Lastly, we investigated a unified model which utilises a number of factors to determine where to direct service requests. The price of electricity and carbon intensity of different electricity supplier varies in different geographical regions. The total electricity cost and carbon emissions will also be affected by the cooling architecture used in the data centre and average service request time must also be considered. In this dissertation we presented an algorithm to control the load distribution in the cloud to achieve a variety of goals. Our work shows that if a cloud operator considers one factor the most important the algorithm can be used to improve the performance of that factor. Our work also showed that it is possible to balance the performance of the algorithm so that all factors are considered.

In this dissertation we have demonstrated how load balancing and rate limiting can be used to improve the performance of a cloud. In recent years there has been considerable focus on improving the efficiency of the data centres which comprise the cloud. As a result the PUE of new data centres has fallen considerably to levels where little improvement is possible in some cases. For example Facebook report that they have achieved a PUE of 1.08 [44]. Much of this work, however, focuses on directing load to the best data centre to achieve a certain goal. Differences in PUE between new and old data centres will help the algorithms to lower electricity costs and carbon emissions and differences in electricity prices and carbon intensities are likely to remain as they are dependant on weather conditions. As such the possibility of exploiting difference in carbon intensities and electricity prices will remain a prospect in the near future.

In addition, new applications are migrating to the cloud. Both HPC and cloud gaming present a number of new challenges to cloud operators. For example, unlike conventional video streaming cloud gaming requires both high constant downlink bandwidth and low latency. Cloud operators will still have to consider the electricity and carbon emissions of these applications. Thus, the algorithms discussed in this work can still be applied to the changing cloud, albeit, with different constraints. New areas of research in cloud computing are beginning, but, there will still be a need to manage both electricity costs and carbon emissions for the best performance.

Large public clouds with data centres in different geographical locations can provide a number of benefits to operators and the design of such centres gives rise to a number of questions some of which have been addressed in this thesis. While the questions that we have we considered are important, perhaps even most important some other issues have not be addressed and could provide the basis for future work. For example, we have only consider one type of data centre in the context of thermal management. Future work should and must consider more general data centres. On a technical level, much work also remains to be done in this direction. For example, we have ignored the dynamics of heat equalisation, the effects of communication delays, and no formal stability proof were given. All of this must be considered in future work. A further issue arises due to the nature of our simulations. Throughout we have considered homogeneous workloads. A more complete work would have evaluated the techniques with heterogeneous and rapidly time varying workload distributions. An interesting future direction also concerns the use of compression algorithms for new applications in the cloud such as cloud gaming. Here the idea is to investigate the trade-off between bandwidth consistency and latency for gaming applications. Similar issues are likely to arise when the cloud is used in a connected mobility context (Intelligent Transportation Systems). Finally, the installation of renewable energy power sources on the same sites as data centres gives rise to new areas of research. Energy storage must be installed at these sites to allow their operation in various weather conditions and the load distribution must be managed to prevent data centres from running out of energy. Despite these omissions we hope that we have demonstrated that smart management of data centres can lead to significant cost reduction in their operation, and conversely, not so smart management can lead to significant waste.

Bibliography

[1] Carbon Monitoring for Action. http://carma.org/.

[2] European Union Emissions Trading System. http://ec.europa.eu/clima/policies/ets/.

[3] M. Al-Fares, A. Loukissas, and A. Vahdat. A scalable, commodity data center network architecture. In *Proceedings of the ACM SIGCOMM*, pages 63–74, Seattle, 17-22 August 2008.

[4] M. Alizadeh, A. Greenberg, D. A. Maltz, J. Padhye, P. Patel, B. Prabhakar, S. Sengupta, and M. Sridharan. Data center TCP (DCTCP). In *Proceeding of SIGCOMM*, pages 63–74, New Delhi, 30 August - 3 September 2010.

[5] A. Almoli, A. Thompson, N. Kapur, J. Summers, H. Thompson, and G. Hannah. Computational fluid dynamic investigation of liquid rack cooling in data centres. *Applied Energy*, 89:150–155, 2012.

[6] Amazon. Elastic Compute Cloud. http://aws.amazon.com/ec2.

[7] D. Anderson, J. Dykes, and E. Riedel. More Than an Interface—SCSI vs. ATA. In *Proceedings of the 2nd USENIX Conference on File and Storage Technologies*, pages 245–257, San Francisco, 31 March - 2 April 2003.

[8] M. R. Anderson and A. C. Robbins. Formation flight as a cooperative game. In *Proceeding of the AIAA guidance, Navigation and Control Conferences*, volume 1, pages 244–251, Boston, August 1998.

[9] M. Armbrust, A. Fox, R. Griffith, A. D. Joseph, R. H. Katz, A. Konwinski, G. Lee, D. A. Patterson, A. Rabkin, I. Stoica, and M. Zaharia. Above the clouds: A Berkeley view of

cloud computing. Technical Report UCB/EECS-2009-28, EECS Department, University of California, Berkeley, Feb 2009.

[10] D. B. Arnold and W. J. Milne. The use of Voronoi tessellations in processing soil survey results. *IEEE Computer Graphics Applications*, 4:22–30, 1984.

[11] ASHRAE Technical Committee 9.9. 2011 Thermal Guidelines for Data Processing Environments - Expanded Data Center Classes and Usage Guidance, 2011. http://tc99.ashraetcs.org/documents/ASHRAE Whitepaper - 2011 Thermal Guidelines for Data Processing Environments.pdf.

[12] D. Atwood and J. G. Miner. Reducing data center cost with an air economizer, August 2008. http://www.intel.com/content/www/us/en/data-center-efficiency/data-center-efficiency-xeon-reducing-data-center-cost-with-air-economizer-brief.html.

[13] F. Aurenhammer. Voronoi Diagrams-a survey of a fundamental geometric data structure. *ACM Computing Surveys*, 23(3):345–405, September 1991.

[14] T. Balch and R. C. Arkin. Behavior-based formation control for multirobot teams. *IEEE Transactions on Robotics and Automation*, 14(6):926–939, 1998.

[15] J. Baliga, R. W. A. Ayre, K. Hinton, and R. S. Tucker. Green cloud computing: Balancing energy in processing, storage, and transport. *Proceeding of the IEEE*, 99(1):149–167, 2011.

[16] H. Ballani, P. Costa, T. Karagiannis, and A. Rowstron. Towards predictable datacenter networks. In *Proceedings of SIGCOMM*, pages 242–253, Toronto, 16 - 18 August 2011.

[17] L. A. Barroso and U. Hölzle. The datacenter as a computer: An introduction to the design of warehouse-scale machines. *Synthesis Lectures on Computer Architecture*, 2009.

[18] R. W. Beard, J. R. T. Lawton, and F. Y. Hadaegh. A feedback architecture for formation control. In *Proceedings of ACC*, volume 6, pages 4087–4091, Chicago, 28 - 30 June 2000.

[19] R. W. Beard and V. Stepanyan. Information consensus in distributed multiple vehicle coordinated control. In *Proceedings of CDC*, pages 2029–2034, Maui, 9 - 12 December 2003.

[20] J. Benediktsson and P. Swain. Consensus theoretic classification methods. *Systems, Man and Cybernetics, IEEE Transactions on*, 22(4):688–704, July/August 1992.

[21] V. D. Blondel, J. M. Hendrickx, A. Olshevsky, and J. N. Tsitsiklis. Convergence in multiagent coordination, consensus and flocking. In *Proceedings of CDC-ECC*, pages 2996–3000, Seville, 12 - 15 December 2005.

[22] V. Borkar and P. P. Varaiya. Asymptotic agreement in distributed estimation. *IEEE Transactions on Automatic Control*, 27(3):650–655, 1982.

[23] M. Cao, A. S. Morse, and B. D. O. Anderson. Agreeing asynchronously: Announcement of results. In *Proceedings of CDC*, pages 4301–4306, San Diego, 13 - 15 December 2006.

[24] R. Carli, A. Chiuso, L. Schenato, and S. Zampieri. A PI consensus controller for networked clock synchroznization. In *Proceedings of IFAC-WC*, volume 17, July 2008.

[25] G. Chen, W. He, J. Liu, S. Nath, L. Rigas, L. Xiao, and F. Zhao. Energy-aware server provisioning and load dispatching for connection-intensive internet services. In *Proceedings of NSDI*, pages 337–350, San Francisco, 16-18 April 2008.

[26] Q. Chen and J. Luh. Coordination and control of a group of small mobile robots. In *Proceedings of the IEEE International Conference on Robotics and Automation*, volume 3, pages 2315–2320, San Diego, 8 - 13 May 1994.

[27] L. Cherkasova, D. Guta, and A. Vahdat. Comparison of the three CPU schedulers in Xen. *SIGMETRICS Performance Evaluation Review*, 35:42–51, 2007.

[28] J. Choi, Y. Kim, A. Sivasubramaniam, J. Srebric, Q. Wang, and J. Lee. Modeling and managing thermal profiles of rack-mounted servers with thermostat. *Proceedings of HPCA*, pages 205–215, 10-14 February 2007.

[29] M. Chowdhury, M. Zaharia, J. MA, M. I. Jordan, and I. Stoica. Managing data transfers in computer clusters with orchestra. In *Proceedings of SIGCOMM*, pages 98–109, Toronto, 16 - 18 August 2011.

[30] B. Chun, D. Culler, T. Roscoe, A. Bavier, L. Peterson, M. Wawrzoniak, and M. Bowman. PlanetLab: an overlay testbed for broad-coverage services. *SIGCOMM Computer Communication Review*, 33(3):3–12, 2003.

[31] Citrix Systems, Inc. Citrix netscaler traffic management guide. http://support.citrix.com/servlet/KbServlet/download/20696-102-342843/NS-TrafficMgmt-Guide.pdf.

[32] H. C. Coles and M. Bramfitt. Modular/container data centers procurement guide: Optimizing for energy efficiency and quick deployment. Lawrence Berkeley National Laboratory, Feb. 2011. Available at `http://goo.gl/ho9cC`.

[33] M. Conti, E. Gregori, and F. Panzieri. Load distribution among replicated Web servers: a QoS-based approach. *SIGMETRICS Performance Evaluation Review*, 27(4):12–19, 2000.

[34] M. G. Corporation. Flovent version 9.1, 2010.

[35] R. Das, J. O. Kephart, J. Lenchner, and H. Hamann. Utility-function-driven energy-efficient cooling in data centers. In *Proceedings of international conference on Autonomic computing*, pages 61–70, Washington, 7 - 11 June 2010.

[36] J. Dean and S. Ghemawat. MapReduce: Simplified Data Processing on Large Clusters. *USENIX Symposium on Operating Systems Design and Implementation*, 2004.

[37] M. H. DeGroot. Reaching a consensus. *Journal of the American Statistical Association*, 69(345):118–121, March 1974.

[38] V. Dumas, F. Guillemin, and P. Robert. A Markovian Analysis of Additive-Increase Multiplicative-Decrease Algorithms. *Adv. in Appl. Probab.*, 34(2002) no.1, 2002.

[39] J. W. Durham, R. Carli, P. Frasca, and F. Bullo. Discrete Partitioning and Coverage Control for Gossiping Robots. *IEEE Transactions on Robots*, 28(2):364–378, 2012.

[40] Eirgrid. http://www.eirgrid.com.

[41] E. Eisenberg and D. Gale. Consensus of Subjective Probabilities: The Pari-Mutuel Method. *The Annals of Mathematical Statistics*, 30(1):165–168, March 1959.

[42] N. El-Sayed, I. Stefanovice, G. Amvrosiadis, A. A. Hwang, and B. Schroeder. Temperature management in data centers: Why some (might) like it hot. In *Proceedings of SIGMETRICS*, pages 163–174, London, 11 - 15 June 2012.

[43] D. Estrin, L. Girod, G. Pottie, and M. Srivastava. Instrumenting the world with wireless sensor networks. In *Proceeding of IEEE International Conference on Acoustics, Speech, and Signal Processing (ICASSP '01)*, volume 4, pages 2033–2036, Salt Lake City, 7 - 11 May 2001.

[44] Facebook. Facebook's carbon & energy impact, Retrieved September 2012. https://s3.amazonaws.com/wildfireapp/facebook/ FacebookGreen/pdf/FB_carbon_energy_impact.pdf.

[45] X. Fan, W.-D. Weber, and L. A. Barroso. Power provisioning for a warehouse-sized computer. In *ACM symposium on Computer Architecture*, pages 13–23, San Diego, 09-11 June 2007.

[46] L. Fang, P. J. Antsaklis, and A. Tzimas. Asynchronous consensus protocols: Preliminary results, simulations and open questions. In *Proceedings of CDC-ECC*, pages 2194–2199, Seville, 12 - 15 December 2005.

[47] A. Faraz and T. Vijaykumar. Joint optimization of idle and cooling power in data centers while maintaining response time. In *Proceedings of ASPLOS*, pages 243–256, Pittsburgh, 13 -17 March 2010.

[48] N. Farrington, E. Rubow, and A. Vahdat. Data center switch architecture in the age of merchant silicon. In *Proceedings of the IEEE Symposium on Hot Interconnects*, pages 93–102, New York, 25-27 August 2009.

[49] J. A. Fax and R. M. Murray. Information flow and cooperative control of vehicle formations. *IEEE Transactions on Automatic Control*, 49(9):1465–1476, 2004.

[50] R. Fielding, J. Gettys, J. Mogul, H. Krystyk, L. Masinter, and P. L. T. Berners-Lee. Hypertext Transer Protocol - HTTP/1.1 RFC 2616. June 1999.

[51] F. C. Frank and J. S. Casper. Complex alloy structures regarded as sphere packings. *Acta Crstallogr.*, 11:184–190, 1958.

[52] I. B. Fridleifsson, R. Bertani, E. Huenges, J. W. Lund, A. Ragnarsson, and L. Rybach. The possible role and contribution of geothermal energy to the mitigation of climate change. *O. Hohmeyer and T. Trittin (Eds.) IPCC Scoping Meeting on Renewable Energy Sources. Proceedings*, pages 59–80, 2008.

[53] P. X. Gao, A. R. Curtis, B. Wong, and S. Keshav. It's not easy being green. In *Proceedings of SIGCOMM*, pages 221–222, Helsinki, 13 - 17 August 2012.

[54] Google. App Engine. http://code.google.com/appengine/.

[55] Google. Web metrics: Size and number of resources, Retreived February 2011. http://code.google.com/speed/articles/web-metrics.html.

[56] Green Grid. http://www.thegreengrid.org.

[57] A. Greenberg, J. Hamilton, D. A. Maltz, and P. Patel. The cost of a cloud: Research problems in data center networks. *SIGCOMM CCR*, 39(1), 2009.

[58] A. Greenberg, J. R. Hamilton, N. Jain, S. Kandula, C. Kim, P. Lahiri, D. A. Maltz, P. Patel, and S. Sengupta. VL2: a scalable and flexible data center network. In *Proceedings of ACM SIGCOMM*, pages 51–62, Barcelona, 17-21 August 2009.

[59] A. Greenberg, P. Lahiri, D. A. Maltz, P. Patel, and S. Sengupta. Towards a next generation data center architecture: Scalability and commoditization. In *Proceedings of the ACM Workshop PRESTO'08*, pages 57–62, Seattle, 17-22 August 2008.

[60] Greenpeace. Make IT green cloud computing and its contribution to climate change, Retrieved February 2011. http://www.greenpeace.org/international/Global /international/planet-2/report/2010/3/make-it-green-cloud-computing.pdf.

[61] A. Gulati, I. Ahmad, and C. A. Waldspurger. PARDA: proportional allocation of resources for distributed storage access. In *Proceedings of FAST*, pages 85–98, San Francisco, 24 - 27 February 2009.

[62] C. Guo, G. Lu, D. Li, H. Wu, X. Zhang, Y. Shi, C. Tian, Y. Zhang, and S. Lu. BCube: a high performance, server-centric network architecture for modular data centers. In *Proceedings of the ACM SIGCOMM*, pages 63–74, Barcelona, 16-21 August 2009.

[63] C. Guo, H. Wu, K. Tan, L. Shi, and Y. Z. S. Lu. DCell: a scalable and fault-tolerant network structure for data centers. In *Proceedings of the ACM SIGCOMM*, pages 75–86, Seattle, 17-22 August 2008.

[64] S. Ha, I. Rhee, and L. Xu. CUBIC: a new TCP-friendly high-speed TCP variant. *Proceeding of ACM SIGOPS Operating Systems Review - Research and developments in the Linux kernel*, 42(5):64–74, 2008.

[65] J. A. Hartigan. *Clustering Algorithms.* John Wiley, New York, 1975.

[66] Y. Hatano and M. Mesbahi. Agreement over random networks. *IEEE Transactions on Automatic Control*, 50(11):1867–1872, 2005.

[67] T. Heath, A. P. Centeno, P. George, L. Ramos, Y. Jajuria, and R. Bianchini. Mercury and Freon: Temperature Emulation and Management for Server Systems. *In Proceedings of ASPLOS*, pages 106–116, 21-25 October 2006.

[68] D. Hinchcliffe. The year enterprises open thier SOAs to the Internet? *Enterprise Web 2.0*, Jan, 2007.

[69] R. E. Horton. Rational study of rainfall data makes possible better estimates of water yield. *Eng. News-Record*, pages 211–213, 1917.

[70] Internet World Stats. Internet world stats usage and population statistics. http://www.internetworldstats.com.

[71] A. Jadbabaie, J. Lin, and A. S. Morse. Coordination of groups of mobile autonomous agents using nearest neighbour rules. *IEEE Transactions on Automatic Control*, 48(6):988–1001, 2003.

[72] A. Jain, J. M. Hellerstein, S. Ratnasamy, and D. Wetherall. A wakeup call for internet monitoring systems: The case for distributed triggers. In *Proceedings of HotNets-III*, pages 1–6, San Diego, 15 - 16 November 2004.

[73] R. Jain. *The Art of Computer Systems Performance Analysis: Techniques for Experimental Design, Measurement, Simulation, and Modeling.* John Wiley and Sons, INC., 1991.

[74] M. Ji, A. Muhammad, and M. Egerstedt. Leader-based multi-agent coordination: Controllability and optimal control. In *Proceedings of ACC*, pages 1358–1363, Minneapolis, 14 - 16 June 2006.

[75] D. Kempe, A. Dobra, and J. Gehrke. Gossip-based computation of aggregate information. *In Proceedings of IEEE FOCS*, 2003.

[76] C. Kim, M. Caesar, and J. Rexford. Floodless in Seattle: A Scalable Ethernet Architecture for Large Enterprises. In *Proceedings of the ACM SIGCOMM*, pages 3–14, Seattle, 17-22 August 2008.

[77] F. Knorn, M. J. Corless, and R. N. Shorten. Results in cooperative control and implicit consensus. *International Journal of Control*, 84(3):476–495, 2011.

[78] D. Knuth. *The Art of Computer Programming III: Sorting and Searching.* Addison-Wesley, Reading, Mass, 1973.

[79] E. Kohler, R. Morris, B. Chen, J. Jannotti, and M. F. Kaashoek. The click modular router. *ACM Transactions on Computer Systems*, 18(3):263–297, 2000.

[80] N. Kothari, R. Mahajan, T. Millstein, R. Govindan, and M. Musuvathi. Finding protocol manipulation attacks. In *Proceedings of SIGCOMM*, pages 26–37, Toronto, 16 - 18 August 2011.

[81] J. Kurose and K. Ross. *Computer Networking - A Top-Down Approach (4thth ed.)*. Addison Wesley, 2008.

[82] J. R. Lawton, R. W. Beard, and B. J. Young. A decentralized approach to formation maneuvers. *IEEE Transactions on Robotics and Automation*, 19(6):933–941, 2003.

[83] D. T. Lee and C. K. Wong. Voronoi diagrams in the $L_i(L_inftY)$ metrics with 2-dimensional storage applications. *SIAM Journal on Computing*, 9:200–211, 1980.

[84] M. Lenzen. Life cycle energy and greenhouse gas emissions of nuclear energy: A review. *Energy Conversion and Management*, 49(8):2178–2199, August 2008.

[85] B. C. Levy, D. A. C. non, G. C. Verghese, and A. S. Willsky. A scattering framework for decentralized estimation problems. *Automatica*, 19(4):373–384, 1983.

[86] M. A. Lewis and K.-H. Tan. High precision formation control of mobile robots using virtual structures. *Autonomous Robots*, 4(4):387–403, 1997.

[87] A. Li, X. Yang, S. Kandula, and M. Zhang. CloudCmp: Comparing Public Cloud Providers. In *Proceedings of IMC*, pages 1–14, Melbourne, 1 - 3 Novermber 2010.

[88] G. Linden. Marissa Mayer at web 2.0. online at:. http://glinden.blogspot.com/2006/11/marissa-mayer-atweb-20.html.

[89] Z. Liu, M. Lin, A. Wierman, S. H. Low, and L. L. Andrew. Geographical load balancing with renewables. In *Proceeding of GreenMETRICS*, pages 1–5, San Jose, 7 - 11 June 2011.

[90] Z. Liu, M. Lin, A. Wierman, S. H. Low, and L. L. Andrew. Greening geographical load balancing. In *Proceeding of SIGMETRICS*, pages 233–244, San Jose, 7 June 2011.

[91] Luiz André Barroso and Urs Hölze. The case for energy-proportional computing. *IEEE Computer*, 40(12):33–37, 2007.

[92] R. C. Luo and M. G. Kay. Multisensor integration and fusion in intelligent systems. *IEEE Transactions on Systems, Man, and Cybernetics*, 19(5):901–931, September/October 1989.

[93] P. Mahadevan, S. Banerjee, and P. Sharma. Energy proportionality of an enterprise network. In *Proceedings of ACM GreenNet*, pages 53–60, New Delhi, 30 August 2010.

[94] Z. M. Mao, C. D. Cranor, F. Bouglis, M. Rabinovich, O. Spatscheck, and J. Wang. A precise and Efficient Evaluation of the Proximity between Web Clients and their Local DNS Servers. In *Proceedings of USENIX*, pages 229–242, Monterey, 10 - 15 June 2002.

[95] V. Mathew, R. K. Sitaraman, and P. Shenoy. Energy-aware load balancing in content delivery networks. In *Proceedings of IEEE INFOCOM*, pages 954–962, Orlando, 25 - 30 March 2012.

[96] M. MEsbahi and F. Y. Hadaegh. Formation flying control of multiple spacecraft via graphs, matrix inequalities, and switching. In *Proceedings of CCA*, volume 2, pages 1211–1216, Kohala, August 1999.

[97] Microsoft. Azure services platform. www.microsoft.com/windowsazure/.

[98] Mikko Pervilä and Jussi Kangasharju. Running servers around zero degrees. In *Proceedings of ACM SIGCOMM Workshop on Green Networking*, pages 9–14, New Delhi, 30 August 2010.

[99] Mikko Pervilä and Jussi Kangasharju. Cold air containment. In *Proceedings of ACM SIGCOMM Workshop on Green Networking*, pages 7–12, Toronto, 19 August 2011.

[100] R. Miller. Microsoft goes all-in on container data centers. http://www.datacenterknowledge.com/archives/2008/12/02/microsoft-goes-all-in-on-container-data-centers/.

[101] F. F. Moghaddam, M. Cheriet, and K. K. Nguyen. Low Carbon Virtual Private Clouds. In *Proceedings of IEEE International Conference on Cloud Computing*, pages 259–266, Washington DC, 4 - 9 July 2011.

[102] J. Moore, J. S. Chase, and P. Ranganathan. Consil: Low-cost thermal mapping of data centers. In *Proceedings of the Workshop on Tackling Computer Systems Problems with Machine Learning Techniques (SysML)*, Saint-Malo, 27 June 2006.

[103] J. Moore, J. S. Chase, P. Ranganathan, and R. Sharma. Making scheduling "Cool": Temperature-aware workload placement in data centers. In *Proceedings of USENIX*, pages 61–75, Anaheim, 10-15 April 2005.

[104] L. Moreau. Stability of multiagent systems with time-dependent communciation links. *IEEE Transactions on Automatic Control*, 50(2):169–182, 2005.

[105] D. Mosberger and T. Jin. httperf-a tool for measuring web server performance. *Performance Evaluation Review*, 26:31–37, 1998.

[106] A. G. O. Mutambara. *Decentralized Estimation and Control for Multisensor Systems.* CRC Press, Inc, Boca Raton,FL,USA, 1998.

[107] R. N. Mysore, A. Pamboris, N. Farrington, N. Huang, P. Miri, S. Radhakrishnan, V. Subramanya, and A. Vahdat. Portland: a scalable fault-tolerant layer 2 data center network fabric. In *Proceedings of the ACM SIGCOMM*, pages 39–50, Barcelona, 16-21 August 2009.

[108] T. Norvig. Consensus of subjective probabilities: A convergence theorem. *The Annals of Mathematical Statistics*, 38(1):221–225, February 1967.

[109] A. Odlyzko. Internet pricing and the history of communications. *Computer Networks*, 36:493–517, 2001.

[110] R. Olfati-Saber. Flocking for multi-agent dynamic systems: Algorithms and theory. *IEEE Transactions on Automatic Control*, 51(3):401–420, 2006.

[111] R. Olfati-Saber. Distributed kalman filtering for sensor networks. In *Proceedings of CDC*, pages 5492–5498, New Orleans, 10 - 11 December 2007.

[112] R. Olfati-Saber and R. M. Murray. Consensus protocols for networks of dynamic agents. In *Proceedings of American Control Conference*, volume 2, pages 951–956, Denver, 4 - 6 June 2003.

[113] R. Olfati-Saber and R. M. Murray. Consensus problems in networks of agents with switching topology and time-delays. *IEEE Transaction on Automatic Control*, 49(9):1520–1533, 2004.

[114] R. Olfati-Saber and J. S. Shamma. Consensus filters for sensor networks and distributed sensor fusion. In *Proceedings of IEEE Conference on Decision and Controland European Control Conference (CDC-ECC)*, pages 6698–6703, Seville, 12 - 15 December 2005.

[115] L. E. Parker. Alliance: an architecture for fault tolerant multirobot cooperation. *IEEE Transactions on Robotics and Automation*, 14(2):220–240, 1998.

[116] L. Parolini, B. Sinopoli, and B. H. Krogh. Reducing data center energy consumption via coordinated cooling and load management. In *Proceeding of HotPower Workshop on Power Aware Computing and Systems*, San Diego, 7 December 2008.

[117] L. Parolini, N. Tolia, B. Sinopoli, and B. H. Krogh. A cyber-physical systems approach to energy management in data centers. In *Proceedings of the 1st ACM/IEEE International Conference on Cyber-Physical Systems*, Stockholm, 13 - 14 April 2010.

[118] C. D. Patel, C. E. Bash, and R. Sharma. Smart cooling of data centers. In *Proceedings of IPACK*, pages 128–137, Maui, 6-11 July 2003.

[119] M. Pathan, C. Vecchiola, and R. Buyya. Load and Proximity Aware Request-Redirection for Dynamic Load Distribution in Peering CDNs. In *On the Move to Meaningful Internet Systems: OTM*, volume 5331 of *Lecture Notes in Computer Science*, pages 62–81. Springer Berlin / Heidelberg, 2008.

[120] M. K. Patterson. The effect of data center temperature on energy efficiency. In *Proceedings of IEEEE ITHERM*, pages 1167–1174, Orlando, 28 - 31 May 2008.

[121] M. K. Patterson and D. Fenwick. The state of datacenter cooling. Available at http://download.intel.com/technology /eep/data-center-efficiency/state-of-date-center-cooling.pdf.

[122] R. Perlman. An algorithm for distributed computation of a spanningtree in an extended lan. *SIGCOMM Computer Communication Review*, 15(4):44–53, September 1985.

[123] A. Qureshi, J. Guttag, R. Weber, B. Maggs, and H. Balakrishnan. Cutting the electric bill for internet-scale systems. In *Proceedings of ACM SIGCOMM*, pages 123–134, Barcelona, 17-21 August 2009.

[124] B. Raghavan, K. Vishwanath, S. Ramabhadran, K. Yocum, and A. C. Snoeren. Cloud control with distributed rate limiting. In *Proceeding of ACM SIGCOMM*, pages 337–348, Kyoto, 27-31 August 2007.

[125] L. Rao, X. Liu, L. Xie, and W. Liu. Minimizing electricity cost: Optimization of distributed internet data centers in a multi-electricity-market environment. In *Proceedings of IEEE INFOCOM*, pages 1–9, San Diego, 15 - 19 March 2010.

[126] W. Ren and R. W. Beard. Consensus seeking in multiagent systems under dynamically changing interaction topologies. *IEEE Transactions on Automatic Control*, 50(5):655–661, 2005.

[127] C. W. Reynolds. Flock, herd and schools: A distributed behavioral model. In *Proceedings of SIGGRAPH*, pages 25–34, Anaheim, July 1987.

[128] H. Rodrigues, J. R. Santos, Y. Turner, P. Soares, and D. Guedes. Gatekeeper: supporting bandwidth guarantees for multi-tenant datacenter networks. In *Proceeding of the USENIX workshop on I/O virtualization*, pages 1–8, Portland, 14 June 2011.

[129] C. A. Rogers. *Packing and Covering.* Cambridge University Press, London,New York, 1964.

[130] P. Rumsey. Overview of liquid cooling systems, 2007. http://hightech.lbl.gov/presentations/Dominguez/5_LiquidCooling_101807.ppt.

[131] S. Savage, N. Cardwell, D. Wetherall, and T. Anderson. TCP congestion control with a misbehaving receiver. *SIGCOMM CCR*, 29(5):71–78, 1999.

[132] L. Schenato and G. Gamba. A distributed consensus protocol for clock synchronization in wireless sensor networks. In *Proceedings of CDC*, pages 2289–2294, New Orleans, 12 - 14 December 2007.

[133] M. I. Shamos. Geometric complexity. In *Proceedings of ACM STOC*, pages 224–233, Albuquerque, May 5 - 7 1975.

[134] R. K. Sharma, C. E. Bash, and C. D. Patel. Balance of power: Dynamic thermal management for internet data centers. *IEEE Internet Computing*, 9(1):42–49, 2005.

[135] H. Shi, L. Wang, and T. Chu. Virtual leader approach to coordinated control of multiple mobile agents with asymmetric interactions. *Physica D: Nonlinear Phenomena*, 213(1):51–65, 2006.

[136] A. Shieh, S. Kandula, A. Greenberg, C. Kim, and B. Saha. Sharing the data center network. In *Proceedings of NSDI*, Boston, 30 March - 01 April 2011.

[137] S. N. Singh, P. Chandler, C. Schumacher, S. Banda, and M. Pachter. Nonlinear adaptive close formation control of unmanned aerial vehicles. *Dynamics and Control*, 10(2):179–194, 2000.

[138] D. E. Snyder. Urban places in uruguay and the concept of a hierarchy. festschrift: C. f. jones. *Northwestern University Studies in Geography*, 6:29–46, 1962.

[139] Social Bakers. Social bakers the recipe for social marketing success. http://www.socialbakers.com/facebook-statistics/.

[140] T. Sridhar. Cloud computing - a primer part 1. 12(3):2–19, 2009.

[141] R. Srikant. *The Mathematics of Internet Congestion Control*. Birkhäuser, Boston, 2004.

[142] R. Stanojević and R. Shorten. How expensive is link utilization? In *Proceedings of NET-COOP*, pages 54–64, Avignon, 5-7 June 2007.

[143] R. Stanojević and R. Shorten. Fully decentralized emulation of best-effort and processor sharing queues. In *Proceedings of ACM SIGMETRICS 2008*, pages 383–394, Annapolis, 2-6 June 2008.

[144] R. Stanojević and R. Shorten. Generalized distributed rate limiting. In *Proceedings of IWQoS*, pages 1–9, Charleston, 13 - 15 July 2009.

[145] R. Stanojević and R. Shorten. Load balancing vs. distributed rate limiting: an unifying framework for cloud control. In *Proceedings of IEEE ICC 2009*, Dresden, 14-18 June 2009.

[146] R. Stanojević and R. Shorten. Distributed dynamic speed scaling. In *Proceedings of IEEE INFOCOM*, pages 1–5, San Diego, 14-19 March 2010.

[147] M. Stokely, J. Winget, E. Keyes, C. Grimes, and B. Yolken. Using a market economy to provision compute resources across planet-wide clusters. In *Proceedings for the international Parallel and Distributed Processing Symposium*, pages 1–8, Rome, 23 - 29 May 2009.

[148] R. F. Sullivan. Alternating cold and hot aisles provides more reliable cooling for server farms. *Uptime Institude*, 2000.

[149] H. G. Tanner, A. Jadbabaie, and G. J. Pappas. Stable flocking of mobile agent part i:fixed topology. In *Proceeding of CDC*, pages 2010–2015, Maui, 9 - 12 December 2003.

[150] H. G. Tanner, A. Jadbabaie, and G. J. Pappas. Stable flocking of mobile agent part ii:dynamic topology. In *Proceeding of CDC*, pages 2016–2021, Maui, 9 - 12 December 2003.

[151] D. Thaler and C. Hopps. Multipath Issues in Unicast and Multicast Next-Hop Selection - RFC 2991. November 2000.

[152] The Uptime Institute. http://uptimeinstitute.org.

[153] A. H. Thiessen. Precipitation average for large area. *Monthly Weather Rev.*, 39:1082–1084, 1911.

[154] J. N. Tsitsiklis, D. P. Bertsekas, and M. Athans. Distributed asynchronous deterministic and stochastic gradient optimization algorithms. *IEEE Transaction on Automatic Control*, 31(9):803–812, 1986.

[155] A. K. Uht and R. J. Vaccaro. TEAPC: Temperature Adaptive Computing in a Real PC. In *Proceedings of the Second Workshop on Temperature-Aware Computer Systems (TACS-2)*, Madison, 4-8 June 2005.

[156] United States Environmental Protection Agency. eGRID. http://www.epa.gov/cleanenergy/energy-resources/egrid/index.html.

[157] M. Veloso, P. Stone, and K. Han. The CMUnited-97 robotic soccer team: Perception and multi-agent control. *Robotics and Autonomous Systems*, 29(2-3):133–143, 2000.

[158] T. Vicsek, A. Czirók, E. Ben-Jacob, I. Cohen, and O. Shochet. Novel Type of Phase Transition in a System of Self-Driven Particles. *Physical Review Letters*, 75(6):1226–1229, 1995.

[159] G. Wang and T. E. Ng. The impact of Virtualization on Network Performane of Amazon EC2 Data Center. In *Proceeding of INFOCOM*, pages 1–9, San Diego, 15 - 19 March 2010.

[160] P. K. C. Wang. Navigation strategies for multiple autonomous mobile robots moving in formation. *Journal of Robotic Systems*, 8(2):177–195, 1991.

[161] A. Weissel and F. Bellosa. Dynamic thermal management for distributed systems. In *Proceedings of TACS Workshop*, Munich, 20 June 2004.

[162] S. C. Weller and N. C. Mann. Assessing rater performance without a "gold standard" using consensus theory. *Medical Decision Making*, 17(1):71–79, February 1997.

[163] P. Wendell, J. Wenjie, M. J. Freedmand, and J. Rexford. DONAR: Decentralized Server Selection for Cloud Services. In *Proceedings of SIGCOMM*, pages 231–242, New Delhi, 30 August - 3 September 2010.

[164] B. White, J. Lepreau, L. Stoller, R. Ricci, S. Curuprsad, M. Newbold, M. Hibler, C. Barb, and A. Joglekar. An integrated experimental environment for distributed systems and networks. *SIGOPS Oper. Syst. Rev.*, 36:255–270, December 2002.

[165] E. Wigner and F. Seitz. On the constitution of metallic sodium. *Phys. Rev.*, 43:804–810, 1933.

[166] C. Wilson, H. Ballani, T. Karagiannis, and A. Rowstron. Better never than late: Meeting deadlines in datacenter networks. In *Proceedings of SIGCOMM*, pages 50–61, Toronto, 16 - 18 August 2011.

[167] R. L. Winkler. The consensus of subjective probability distributions. *Management Science*, 15(2):B61–B75, October 1968.