DEPARTMENT OF COMPUTER SCIENCE

Pricing The Cloud
An Investigation into Financial Brokerage for Cloud Computing

Philip James Clamp

A dissertation submitted to the University of Bristol in accordance with the requirements of the degree of Master of Engineering in the Faculty of Engineering.

Thursday 23rd May, 2013, CS-MEng-2013

Declaration

This dissertation is submitted to the University of Bristol in accordance with the requirements of the degree of MEng in the Faculty of Engineering. It has not been submitted for any other degree or diploma of any examining body. Except where specifically acknowledged, it is all the work of the Author.

Philip James Clamp, Thursday 23rd May, 2013

Contents

List of Figures

List of Tables

List of Algorithms

Executive Summary

Abstract

Cloud Computing is the latest paradigm backed to be the realisation of the long sought dream of supplying compute resource as a utility. In 2009, a team of researchers at UC Berkeley's RAD lab unveiled a paper claiming that Cloud Computing "*has the potential to transform a large part of the IT industry, making software even more attractive as a service and shaping the way IT hardware is designed and purchased*." [1] In the few short years since, it appears that their predictions were accurate. Many of the world's leading technology companies are now offering 'Cloud' services - Amazon with their Web Services, Google with AppEngine, Microsoft, IBM along with new players such as Salesforce, who have achieved remarkable feats of growth in a short period of time.

In 2012, Rogers and Cliff published a paper [37] proposing that a third party, acting as a broker, could be introduced that would not only benefit the providers and consumers of the cloud resource, but could also make a profit for itself. The broker acts in a similar way to other financial markets, matching provider supply to consumer demand, creating a source of liquidity. This is achieved through offering options, a type of financial derivative contract, with the compute resource as the underlying asset. The model gives the provider a better idea of upcoming demand, allows the users to potentially purchase the resource at a rate lower than the pay-on-demand offerings and also allows the broker to charge a fee, benefiting all parties involved.

In this project, that research is extended further. This was achieved by implementing the model in CReST, an open-source data centre simulation platform. Several interpretations of the model were possible from the original paper and the intended meaning is further clarified while carrying out verification of the original results. The performance of the model is analysed in a multitude of situations, using both Rogers' strategies and further original scenarios. Finally, a method is proposed to allow the fully autonomous operation of the agent to remain profitable in any market scenario with no *a priori* knowledge of the domain.

Achievements

- This is the first project to utilise the CReST platform that is still in development under the LSCITS (Large Scale Complex IT Systems) initiative. The model proposed by Rogers was implemented in the platform in Java, comprising of backend modules, GUI components and configuration interfaces. Additionally tested, fixed bugs and released the work completed for further research.
- The model was thoroughly tested under a variety of market situations, optimising and modernising the parameters to determine the performance adjustments.
- Constructed a suite of analytical scripts in Python to post-process the vast amount of data outputted by experiments.
- Critically analysed previous research and proposed a Widrow-Hoff [46] inspired automation extension mechanism for the model.
- Experimented with the model in a wide variety of market situations, analysing performance and discussing its real-world application.
- Conducted over 30,000 experiments totalling over 250 hours, run on Amazon Web Services.

Supporting Technologies

- The open source CReST software was employed, forming the foundations for the implementation of this project. `http://sourceforge.net/projects/cloudresearch/`
- The Eclipse IDE (`www.eclipse.org`) was the main development environment for the project, making use of the subversion source control, debugging and Java development toolkits.
- The Python 2D graphics environment *matplotlib* was used to produce graphics for analysis. `http://matplotlib.org/index.html`
- The *SciPy* statistics library was used to help in the automation of statistical testing. `http://www.scipy.org/`

Notation and Acronyms

CReST	:	Cloud Research Simulation Toolkit
LSCITS	:	Large Scale Complex IT Systems
SLA	:	Service Level Agreement
VM	:	Virtual Machine
SaaS	:	Software as a Service
PaaS	:	Platform as a Service
IaaS	:	Infrastructure as a Service
AWS	:	Amazon Web Services
WZH	:	Creators of the Truth-Telling price model - Wu, Zhang and Huberman.
AAT	:	Autonomous Adaptive Thresholding

Acknowledgements

First and foremost I would like to thank Dr John Cartlidge, who not only managed to suggest a topic that catered for the range of interests I wanted to cover in my project but also consistently provided sound advice and guidance throughout its course. Additionally, the aid of Owen Rogers was greatly appreciated when technical difficulties relating to implementation of his work were met.

Finally, I would like to take this opportunity to thank my family and friends for their encouragement throughout the year. The positive reinforcement I received has helped me to produce a piece of work that I am immensely proud of.

Chapter 1

Introduction

1.1 Technology: The Big Picture

Technological revolutions have played a significant role in shaping the world as it exists today. Beginning in 18th century Britain, roughly every 40-60 years the technological landscape is transformed once more, each change diffusing worldwide from an initial core location. A key characteristic of these revolutions is the introduction of new technologies, infrastructures and organisational principles that drive development and have the capability of modernising and increasing the productivity of the world's economy. [32]

The current technological revolution, the fifth of its kind outlined by Carlota Perez, came into prominence in the early 1970s. Dubbed the age of information, cheap microelectronics have enabled the mass production of computing equipment which has subsequently not only found its way into offices, but also into the modern day home. The invention of the World Wide Web by Tim Berners-Lee while at *CERN* [3] is an example of a generic technological tool that has fundamentally altered the way we communicate, shop, work, learn and generally live our lives on a day to day basis. As this particular revolution matures further, the underlying infrastructure of the web, the Internet, continues to provide opportunities to alter the way businesses operate and in turn, influence the evolution of the world economy. With this in mind, the information technology industry is undeniably going through a period of change once again. Although one might argue that it is constantly evolving, advancements in the past decade suggest a more significant transformation is afoot.

Some part of this is embedded in the self-fulfilling prophecy known as Moore's Law, the observation made by Intel co-founder Gordon Moore in 1965 that the number of transistors on integrated circuits would approximately double every two years. [29] To date, the law has remained accurate, although in 2005 he stated that the trend would not continue forever due to limitations in the miniaturisation of transistors. [13] Compounded against the physical limitations being experienced by the chip manufacturers is the increasing hunger from firms to take advantage of vast amounts of electronic data generated on a day-to-day basis. Eric Schmidt, Chairman of Google, gave a speech where he claimed: "*Between the birth of the world and 2003, there were five exabytes of information created. We [now] create five exabytes every two days.*" [41]

For firms to take advantage of the growing amount of information at their disposal, methods are required to process, analyse and visualise it in a suitable amount of time. The compute resource required to perform these tasks cannot be provided by a regular desktop workstation. As a consequence, the benefits of using the data has been limited to organisations that have the capital to invest heavily in information technology infrastructure. An example of an enterprise that has comparable requirements is the animation company, *Pixar*. Rendering animations is an incredibly compute intensive task and to make their movies, *Pixar* owns a huge bank of computers known as their "render-farm". Having access to these kinds of assets has historically been a barrier to market entry, due to the enormous cost associated with purchasing and maintaining such equipment. [11] It is clear that in order to promote innovation and drive technology to the next level, this barrier to entry needs to be eradicated.

1.2 Computing as a Utility

Nicholas Carr, in his 2009 book '*The Big Switch*' [6] explores the similarities of current trends in the computing industry with events that occurred almost 150 years ago. In the mid 19th century, factory owners were on the precipice of revolutionising the manufacturing industry. Machinery was being employed to produce goods more efficiently than ever before, but power was a difficult resource to attain. The factory owners themselves were the power producers. Henry Burden, a Scottish engineer that had emigrated to the United States in 1819 was one such example. He was an inventor, originally working for a farming-tool manufacturer and later managing and owning an Iron and Nail factory which he renamed the Burden Iron Works. Burden quickly established himself as a market leader, thanks to key geographical positioning near the Hudson river and a magnificent water wheel. The energy provided by the water wheel gave Burden a significant competitive advantage, allowing more machinery to be employed that not only retained the quality of product, but greatly improved productivity. Fifty years later, Burden's water wheel and many other private generators had been rendered obsolete. This was due to scientific and engineering breakthroughs that allowed electricity generation to be centralised and distributed over a network of wires. Economics did the rest. The economies of scale that could be achieved by supplying many clients from the same pooled resource meant that the prices dropped so low that it was affordable to everybody - soon electricity was available to almost every business and household in the country. Carr argues that a similar movement is underway within the computing industry today, with the broadband Internet forming the underlying infrastructure required to make the idea a reality. Throughout the book, he forms a convincing defence by drawing parallels with the events that led to the uptake of electricity as a utility.

Offering compute resource as a utility, not unlike electricity and gas, is not a new idea; in fact there are references suggesting the concept that date back to the early 1960s:

> "*If computers of the kind I have advocated become the computers of the future, then computing may someday be organized as a public utility just as the telephone system is a public utility... The computer utility could become the basis of a new and important industry.*" [15]
> John McCarthy (MIT Centennial 1961)

With the significant advances made in the information and communications technology industry in recent years, the vision that computing will one day be the fifth utility has become increasingly recognised. As with electricity before, there are limiting factors which means that the dream is yet to be reality. The technology to support such a utility is not wholly available yet, but a number of paradigms have been proposed that could form the basis of delivering compute resource remotely. Over time different paradigms have gained and declined in popularity, with some significant examples including *Cluster Computing*, *Grid Computing* and most recently, *Cloud Computing*. [34] This work will focus on the latest paradigm, Cloud computing, which appears to have the strongest claim for moving computing to the utility domain.

Cluster Computing

"*A cluster is a type of parallel and distributed system, which consists of a collection of inter-connected stand-alone computers working together as a single integrated computing resource.*" [5, 33]

Grid Computing

"*A grid is a type of parallel and distributed system that enables the sharing, selection, and aggregation of geographically distributed 'autonomous' resources dynamically at runtime depending on their availability, capability, performance, cost and users' quality-of-service requirements.*" [34]

Cloud Computing

"*A Cloud is a type of parallel and distributed system consisting of a collection of inter-connected and virtualised computers that are dynamically provisioned and presented as one or more unified computing resource(s) based on service-level agreements established through negotiation between the service provider and consumers.*" [16]

1.3 Project Challenges

Multiple challenges are presented in this work in the domains of research, design, implementation and testing. The project will involve working with an existing simulation platform in order to implement, optimise and investigate the performance of Rogers' proposed brokerage model. The area of research is young and has the compounding issue of firms being secretive about their findings. The result is relatively little in the way of relevant research level material. In order to compensate, reputable news and online article archives are leveraged to bridge the gap.

The primary platform employed to perform the experiments in this work is known as CReST (Cloud Research Simulation Toolkit), an open-source project overseen by LSCITS. The project has been developed over the last year by a number of contributors, including student interns over the summer for the purposes of research. The software is complex and of significant size, yet is still in a fairly early development stage. A consequence of this is the existence of bugs and a lack of certain features. Seeing the codebase for the first time when taking on this project meant that a significant technical challenge involved understanding the design of the application and the patterns in use. Furthermore, adapting and adding features along with debugging the platform were requirements to not only successfully perform the experiments for this project, but to ensure that the platform would be an effective research platform for the future.

Design and implementation of the brokerage component for CReST was not a straightforward task, with no concept of pricing originally present in the application. Additionally, the model proposed by Rogers involved two distinct periods and this had to be adapted to fit into the event-based nature of the framework. To recreate the original experiment, two additional modules were required, one to handle pricing and the second to implement the brokerage component. Development was completed in the primary language of CReST, Java, following the same design patterns as previously implemented to aid code uniformity and understandability for new contributors.

1.4 Project Aims & Objectives

1. Replicate and verify the results of Rogers' original experiment through implementation in the CReST framework.
2. Discover whether brokerage is a profitable venture given the current pricing strategies of Cloud Computing providers.
3. Perform a sensitivity analysis, determining the scale of the effect of extrinsic environmental alterations on the performance of the Broker.
4. Improve the broker model through automated threshold optimisation, which should enable profit maximisation given any demand profile.
5. Thoroughly test both static and adaptive implementations of the Broker over different demand profiles and analyse the behaviour when the market suddenly changes, i.e. a shock occurs.

Chapter 2

Technical Background

2.1 Cloud Computing Overview

Cloud computing is an exciting concept poised to be the next step in the delivery of computing services as a utility - a model for enabling ubiquitous, convenient, on-demand network access to a shared pool of configurable computing resources. This essentially involves shifting the location of the compute infrastructure to the network, enabling the reduction of costs commonly associated with managing hardware and software assets. [20, 27]

In order to achieve this feat, a variety of technologies are required to collaborate which has made the overall picture a confusing one. [22] Furthermore, with the long standing dream of computing as a utility at the forefront of many a mind, the hype that has built up surrounding the introduction of the paradigm has caused further uncertainty as to what the term captures. [28] According to Gartner's 'Hype Cycle', cloud computing is arguably sitting somewhere between the initial *trigger* stage and the so called '*trough of disillusionment*', a period where over enthusiasm is still rife and expectations are extremely high. [19, 43]

The term itself encapsulates both the applications delivered as a service and the underlying hardware and software infrastructure in the ultra-modern data centres that makes the concept viable. [1] This infrastructure is commonly known as a Cloud, while an application delivered to end users is referred to as *Software as a Service*. Thanks to a set of services with common characteristics provided by a number of important industry players, a useful stratification has been devised that describes the different layers involved in the cloud computing stack. See figure 2.1 on page 6 for a visual representation.

2.1.1 The 'as a Service' (aaS) Stratification

IaaS - *Infrastructure* as a Service. Underpinning the stack, IaaS offers a developer remotely accessible raw infrastructure, normally virtualised, which they can configure to their own desires and requirements. The computing resources owned by a Service Provider is pooled, with virtual machines utilised to split and dynamically resize it in order to provide ad-hoc systems demanded by customers. [43] This is useful for users who wish to perform batch processing tasks using the cloud, or prefer to have more control over the environment in which their applications reside.

PaaS - *Platform* as a Service. Cloud systems can offer an additional abstraction level - instead of offering virtualised infrastructure, PaaS offers developers all the middleware functionality required to rapidly develop and deploy applications - the software platform on which the systems run, in a remotely accessible package. One example of a current PaaS offering is Salesforce's *Force.com* which allows developers to build and deploy multitenant applications on their servers as a service. [40]

SaaS - *Software* as a Service. At the highest level of the stack lies SaaS, the end user applications or software delivered to the users over the Internet. An example of this could be full-functionality word processing software accessed using a standard web browser such as Google Docs - all required processing is performed server-side at the content provider's data centre. In other words, SaaS includes software traditionally purchased on physical discs and installed onto a user's machine.

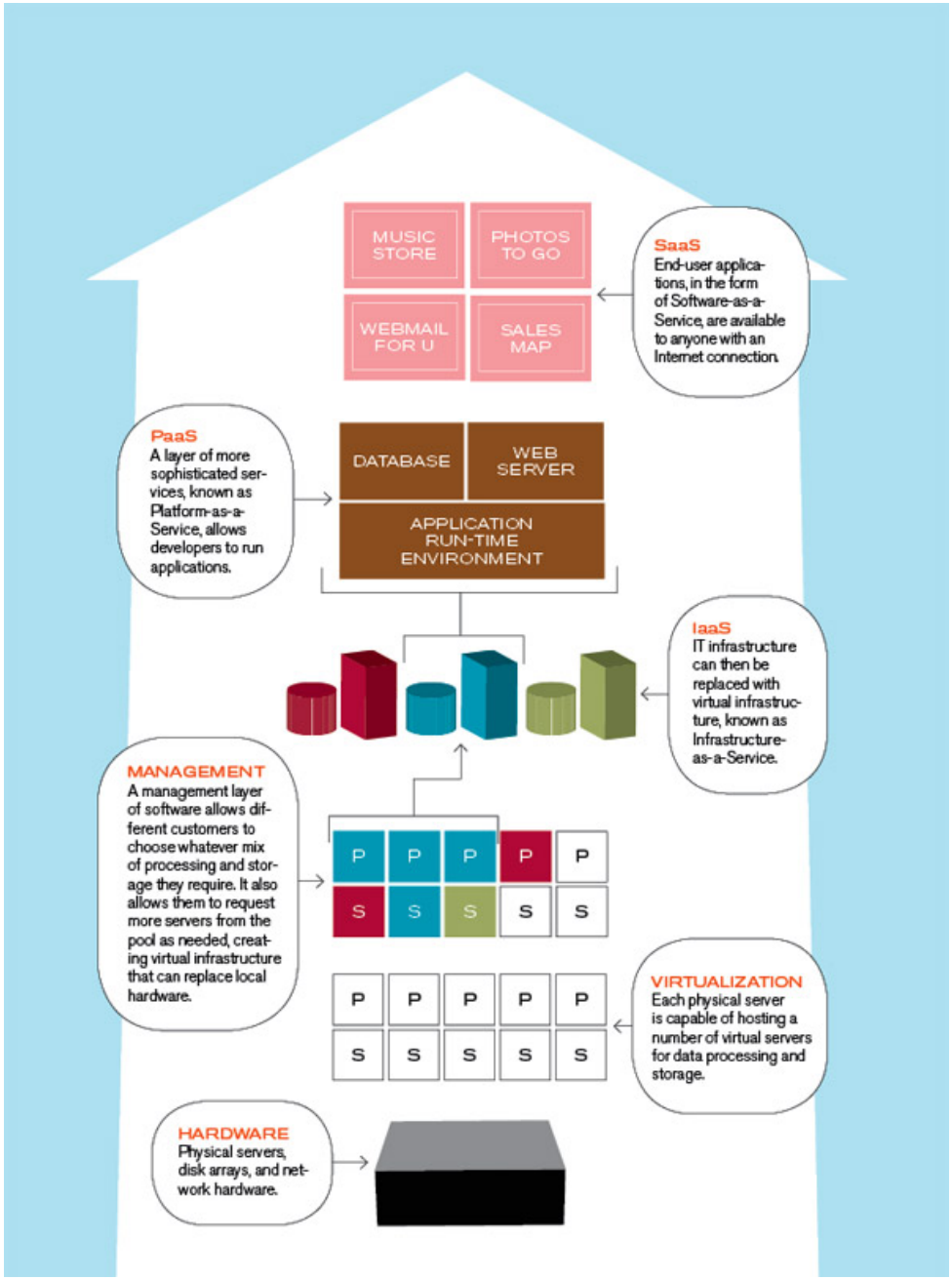

Figure 2.1: The Cloud Stack. Created by Author of Technology Overview: Conjuring Clouds [30]

2.1.2 Public and Private Clouds

It is not the case that every company can afford to invest in their own data centre in order to deliver their software offerings as a service. The cost to operate such a facility is extremely high, requiring constant monitoring and maintenance. For the handful of companies fortunate enough to possess the capital to invest in the appropriate infrastructure, there is opportunity available to offset the cost through offering spare servers out to the public in a pay-as-you-go manner. Other companies or even individuals can choose to host remotely accessible applications on this infrastructure and pay a certain rate for the compute usage per hour. This offering is known as a *Public Cloud*, an early manifestation

of computing as a utility. Current examples of such services include Amazon Web Services, Google AppEngine and Microsoft Azure. It is important at this stage to recognise that there are three key parties that need to be recognised as part of the cloud computing paradigm. The *Cloud Provider*, *SaaS Provider* and *End User*. The SaaS Provider builds the software that is hosted on the Cloud Provider's infrastructure and is also delivered to the End User. [1]

This shift to remotely hosting services on infrastructure owned by another corporate entity is a meteoric change to the prior culture of companies building up their own mammoth systems. Understandably, there is an uncomfortable stance towards the idea of hosting private data on external services, especially when it is sensitive in nature or gives the entity a competitive advantage. To combat these fears, some corporations have begun to invest in their own cloud infrastructure, internal data centres that are not made available to the public. These are known as *Private Clouds*, and are often constructed in such a way that if extra capacity is required at peak consumption, the private systems can easily integrate into the public offerings, an approach known as *cloudbursting*. Normally the term cloud computing does not include Private Clouds, referring instead to the pay on demand public offerings.

The impact of this change is not only limited to the software developers who have to be mindful of the platform that they are designing applications for, but also to the major IT systems suppliers. The strategy of the hardware manufacturers is already evolving, announcing plans for new products aimed at corporations interested in investing in their own cloud facilities. Furthermore, plans are afoot to produce terminals or thin clients, computers with good displays and peripherals but lacking in processing hardware which makes them very cheap. [11] These offerings are the perfect combination to cloud computing, where all processing is done elsewhere and the output is delivered through the Internet. This plays into the dream that in the future, there will be no need for a machine at the desk - the only compute power required will be that to run a web browser, everything else will be done in the cloud.

2.1.3 Paradigm Driving Technology

The creation of the Internet is the main enabler towards achieving the ultimate goal of computing as a utility, creating a worldwide system of interconnected computer networks that enables communications between devices located all around the planet. [34] The cloud computing model has no particular need for powerful machines, in fact it has been known for companies to opt for the opposite - low-powered, cheap and abundant machines that are easy to replace. [17]

Advances in hypervisor technologies such as Virtual Machines (VM) provide a significant driving force behind the recent successes in the industry. VMs allow for a single server to run multiple independent instances of a desired resource, be it a hardware platform, operating system, storage device or anything else. This means that Cloud Providers can run multiple virtual machines on a single server, each configured and rented out to different customers and as such creating several streams of income. [11] The utilisation of virtual machines also means if an instance crashes, another matching instance can instantly be started up on another physical machine, minimising human input and downtime and in turn allowing providers to better keep to their contractual Service Level Agreements (SLA) that are agreed with customers beforehand.

On top of these important technologies is the required stack of company specific software infrastructures, built over a number of years to enable the autonomous running of their data centres. Without such technologies, the vast complexity of the data centres would be impossible to manage. An important step forward has been the willingness of some of the companies to reveal and open-source designs and technologies to enable a more competitive market in the future. This open-sourcing has led to the formation of an organisation called the Open Compute Project, originally a project out of Facebook that has growing support in the industry. [21]

2.1.4 Benefits for Providers

On first inspection, it is not abundantly clear why a business entity would want to become a Cloud Provider. Building a facility such as a data centre is a serious investment, typically costing hundreds of millions of dollars to build, provision and launch. [12] In addition to the infrastructure itself, intricate and scalable software to manage the data centre and automate the process of provisioning virtual machines needs to be developed. On the other side of the coin, during the early 21st century multiple prime information technology companies such as HP, IBM, Amazon, Google and others had already began to develop their own solutions. In the case of Amazon and Google, this was to support their core businesses which were undergoing a period of extraordinary growth. Other companies had already realised the visions of utility computing and were preparing their own solutions for the idea of remotely hosted computing services. As already well established multinationals, these companies had the capital required for such a mammoth investment along with the operational expertise at hand to make them a success. [11]

If a company happens to be in such a position, there are several benefits that may influence a decision to enter the market as a Cloud Provider. [1]

Increasing Profitability. It is possible through a process of statistical multiplexing that a single server can be rented to multiple clients with little to no risk involved, through the technology of virtual machines. This process allows a choice of specifications to be made for an instance and the increase of profits per box in operation. Typically companies are only charging around $0.1 per hour for compute resource and $0.12 per gigabyte-month for storage making it difficult to comprehend how the income outweighs the costs. However, firms with such a huge infrastructure can purchase hardware, network bandwidth and power at a vastly reduced rate - in some cases at around 15% of the price that a common medium sized corporation could pay. Leveraging these economies of scale, the services offered come at low cost and as such allow the provider to profit significantly.

Leveraging Existing Investments. In the cases, such as Amazon, where significant investment is required in hardware and software infrastructure to support the computing operations of the main area of business, there are periods of high activity and conversely low activity. This is because the data centres have to support the high amounts of traffic experienced in peak busy periods such as Christmas, yet for the rest of the year it may not be needed. Werner Vogels, Amazon's CTO, claimed that Amazon Web Services (AWS) was initially created for Amazon's internal operations. Offering these services to the public and other organisations adds a new revenue stream, likely at a fairly low cost; helping to offset the initial investments. [44]

Defend and Expand existing Franchises. As the world continues to move towards the utility computing model, with many different providers offering solutions that synchronise users data to the cloud or perform expensive computations remotely, vendors with an established franchise may be forced to update their offerings in order to keep in line with current trends. This could be achieved by extending current applications with a cloud based component, in order to help future-proof or facilitate a migration to SaaS in the future, which may be required given the momentum of the uptake of cloud based services.

Preventing a Monopoly. Some companies may choose to enter the market purely to prevent a competitor from monopolising. In practice this has led to multiple different methods for SaaS developers to deploy - they can choose to have more control on an IaaS platform such as Amazon Web Services, or choose a more PaaS related offering like Google's AppEngine, which provides automation for scalability and load balancing that developers otherwise might have to deal with themselves.

Leveraging Client Relations. In the business world, earning the trust of clients is an extremely important and laborious task, especially when vast sums of money are involved for service offerings.

If a company has an existing relationship with an IT services provider such as IBM, it may well be the case that they would prefer to use their services than a competitor whom they may have no prior experience with. For this reason, it is important for the Service Provider to appease their clients by offering modern services that they may require and as a consequence, investment in a cloud platform helps to maintain the already established relationship.

Forging a Platform. Some Internet firms, for example Facebook, provide specific functions and want users to engage with their service as much as possible, providing plenty of opportunity to advertise to them. One method to do this is to allow them to perform different every day tasks or play games using the service. This is often achieved through a plug-in environment, which in turn is a great fit for cloud computing. Through offering their service as a platform so other companies and developers can build plug-ins, Facebook can increase profitability in several different ways through engaging users with their service for longer. Moreover, additional revenues could also be generated through charging developers to deploy on the platform, the advantage for them being the instant accessibility to a large user base.

2.1.5 Benefits for Clients

There are significant advantages to SaaS over traditional software distribution methods, both to the end user and to the software provider. Service providers no longer have to concern themselves about supporting multiple different versions of their software, instead having centralised installation and control over versioning. Their clients connect to the applications at any time and from any location, enabling them to use any device with an Internet connection to hand. In turn, this enables software providers to supply mobile versions of their applications, granting even more flexibility to their user base. Within this framework, the user data is stored remotely and as a result collaboration and sharing within applications is not only viable, but provides a significant incentive to use remotely hosted services.

Prior to the introduction of cloud computing, delivering a new web based application typically involved predicting the potential size of the user base and growth. This information could then be employed to plan and build a system optimised to support peak usage. Furthermore, additional variable costs are incurred such as the employment of staff to support the IT facilities, storage space, cooling for the servers and maintenance for inevitable hardware failure. Clearly, this is a high-commitment model for the application developer, requiring constant evaluation of usage in order to ensure that a high level of service is maintained for the clients. It is desirable for the service provider to purchase new equipment at the latest possible point, due to the speed at which new technology is developed and falls in price. The rise to popularity of social networking and the sharing culture only served to make this model even more difficult to execute, where applications had the potential to go viral, meaning that a user base could swell incredibly overnight. [45]

Employing a public cloud yields a multitude of advantages for application developers, in particular those that are predicting a strong uptake of new users. There are three key new aspects from a hardware perspective that offer significant rewards over building out a bespoke system. [44]

1. The illusion of infinite computing resources available on demand, eliminating the requirement to predict usage patterns and plan hardware upgrades in advance.

2. No requirement of an up-front commitment by Cloud users, allowing a company to start small with relatively little capital requirements to release an initial application. Hardware resources need only be increased when it suits their needs.

3. The ability to pay for computing resources on a short-term basis as and when needed. These can be increased, and more importantly decreased when needed, conserving energy when machines are no longer useful.

Case Study: Animoto

Animoto, a small American web-based business that produces videos from a set of user-uploaded photographs and audio, is a prime example of the benefits that cloud computing can provide for small start-up businesses. Animoto's service analyses the audio and synchronises the stream of photos with the track in order to create an entertaining music video. The service itself was built on top of Amazon's Web Services (AWS) framework, enabling them to benefit from the scalability and on-demand pricing that the cloud provides. On April 19th 2008, Jeff Bezos, the founder and CEO of Amazon gave a public lecture in which he explained one phenomenal weekend that Animoto had experienced. In short, the service went viral after introducing a new application on the Facebook platform. Animoto had been running for some time on around 50 Elastic Compute Cloud (EC2) instances, but following the sharing of the new, free service over the social networks, the application became instantly popular. 750,000 new users signed up over the course of the weekend and in order to support the explosive growth, the service scaled up to 3,500 instances of EC2. Without the elasticity of cloud computing, the site simply would have fallen over, unable to support the significant increase in traffic. [4]

2.1.6 Other Cloud Usage Opportunities

This new paradigm not only offers a favourable environment for SaaS providers. The nature of renting virtual machine instances allows for more general computing work to be performed, that could be of benefit to many different types of organisations. [1]

Mobile Interactive Applications. One of the major benefits of modern mobile devices is the ability to consume information on the move. Many services exist that gather news and other data from different sources around the web, mash them up and provide them to the user in a simple, highly usable interface. There are many different devices on the market, offering hugely variable amounts of processing power. Coupled with a data connection that may not be reliable and the requirement to collect and process large datasets makes the utilisation of the Cloud for the majority of processing a sensible option. The mobile application then only needs to render the interface and download the articles to display.

Parallel Batch Processing. The ability to rapidly scale up the number of machines provides a significant opportunity to entities that need to perform computation-heavy batch processing jobs. Due to the pay-on-demand nature of cloud computing, it costs the user the same amount to rent 10,000 instances for 1 hour as renting 1 instance for 10,000 hours. Under the right set of circumstances where there is sufficient data-parallelism within the application, utilising the Cloud can offer throughput unrivalled by all but a minority of institutions with the necessary resources to afford this kind of infrastructure for themselves. Popular programming abstractions exist in the form of Google's MapReduce and open-source equivalent Hadoop to aid developers while constructing these kinds of applications to hide the inherent complexity of parallel computing.

Analytics. Since the rise of the information revolution, companies have been gathering and storing data about everything, but it hasn't been until recently that it has started to become useful. Business analytics is an expensive, compute-intensive process of understanding customers, buying habits, ranking and much more. Harnessing the terabytes of data collected can enable firms to create models, spot trends and generally improve business practices and operational efficiency. As the Internet continues to intertwine itself further and further into our everyday lives, more data will continually be produced leading to further requirement of compute resource to perform business analytics. The cloud is placed as an ideal factory to achieve this kind of batch processing, driving the marketplace of the future.

Case Study: New York Times

In 2007, The New York Times (NYT) newspaper was undertaking the task of digitising all of the 11 million articles published between 1851 and 1980. The files had been promised to customers in the industry standard portable document format (PDF), but the scanned originals were unfortunately stored in the image format TIFF. In some cases, multiple TIFF images needed to be stitched together to present the full article as a PDF and as a consequence, a conversion procedure needed to take place. Originally, this was completed in a dynamic fashion - the articles only went through the process of conversion when a customer requested to read the article. This solution appeared to work well, but as the number of requests rose it was clear that a substantial amount of compute power would be required to keep their readers satisfied with the service. A member of staff, Derek Gottfrid, decided that pre-generating and storing the articles would be a better idea. Unfortunately generating 11 million articles sequentially would be a very slow process and could take some time using the resources he had available at the time. Instead he opted to generate the articles using AWS, leveraging 100 EC2 instances and storing the 1.5TB of data using S3. This process only ended up taking 24 hours and it meant that customers could access the articles instantly rather than having to wait each time for a program to convert the images. [18]

2.1.7 NIST - The Essential Characteristics of Cloud Computing

The National Institute of Standards and Technology (NIST) neatly summarise the essential characteristics and requirements of a standardised cloud computing service under the following categories [27]:

On Demand Self-Service. A user can provision computing resources as they require, in an automatic fashion.

Broad Network Access. Access to the resources are available over a network and can be accessed through standard mechanisms, available to a variety of client platforms.

Resource Pooling. The provider pools resources so that they are available to multiple customers with different physical and virtual resources dynamically assigned according to demand.

Rapid Elasticity. Capabilities can be elastically increased or decreased, in some cases automatically, according the the users demand. To the user, capabilities appear to be limitless.

Measured Services. Systems are automatically controlled and optimised using a metering capability, such as pay-per-hour or charge-per-use, relevant to the service being used. Usage can be controlled, monitored and reported providing adequate transparency for both provider and user alike.

2.2 The Present and Future Market

2.2.1 The Current Market

At present, the dream of trading compute resource in a similar manner to the more traditional utilities is a way off. This is in part due to the rate at which cloud computing has come to the forefront of the industry, with several large companies offering bespoke solutions in order to cement early dominance in a huge market. Early adoption is extremely important for modern, complex technologies as the more they are adopted, the more experience is gained using the tools and in turn, the more they are improved. [2] In other words, adoption breeds further adoption.

Unfortunately for the consumer, this series of events can have the side-effect of creating a vendor lock-in situation. The products that are initially popular will improve further, creating a barrier to market entry as customers may not be willing to try new products with less mature or differing functionality. Additionally the interfaces to each of the technologies is likely to differ and once a consumer has established themselves as using one particular solution, it may be difficult to switch providers.

Certainly this rings true for the current cloud computing market. Although there are several significant entities offering solutions [26], the offerings vary wildly and there is no common interface that allows a consumer to easily switch provider. This is because the standards are still being constructed and the consumers aren't yet concerned with provider independence. [25] At some point in the near future, if cloud computing is truly the manifestation of the fifth utility, industry standards will be required to allow consumers to easily switch between providers.

2.2.2 The Federated Cloud: Moving Towards a Market Mechanism

At present, the market has a limited number of Service Providers, each with proprietary interfaces to their services and an inflexible pricing structure. A standardised interface for utilising cloud resources could make the providers interoperable. A federated cloud could then in theory pave the way for the creation of a market infrastructure for trading units of computing resource as a commodity. This would allow the prices for the resources to smoothly vary, whilst the market mechanism could match consumer demand to provider supply. [34]

2.2.3 An Example Market, Involved Parties and their Role

The following is a market system that could potentially be used to trade cloud compute resources, modelled on current real world exchanges.

The Order Book. The market order book is a directory of the prices submitted by the buyers and sellers of the resource that is maintained by the trading venue.

The Auctioneer. Auctioneers periodically clear the bid and ask prices submitted by the market participants.

The Banking System. The banking system is an intermediary which ensures that financial transactions pertaining between the participants in the marketplace are carried out.

Brokers. Brokers mediate between the consumers and the providers through purchasing resources from the providers and sub-leasing to the customers, essentially matching demand the buyer's demand to the seller's supply. A single broker can accept orders from many consumers, whom in turn have the option of submitting their requests to different brokers. The market participants are bound to requirements and compensation through Service Level Agreements (SLAs). These SLAs detail the

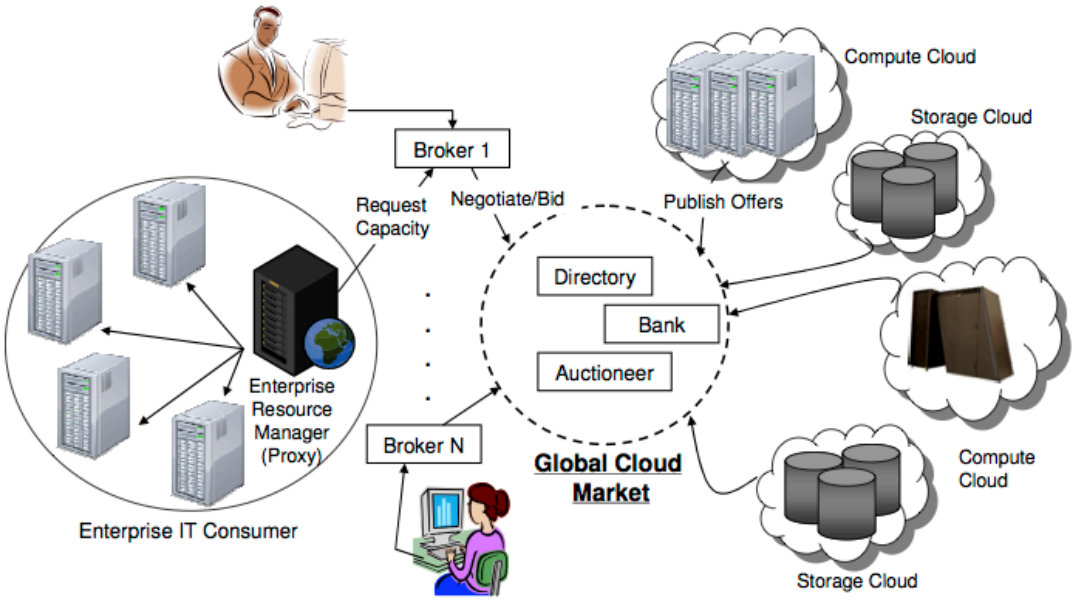

Figure 2.2: Cloud Exchange for Trading Compute Resource. Created by Authors of cloud computing and Emerging IT Platforms [34]

services that will be provided and are agreed upon by all involved parties, with penalties in place for non-adherence. Typically, a broker turns a profit in one of the following ways:

1. Charging a commission fee for any completed deal.
2. By varying the *spread*, the difference between the price the broker buys the resource at and the price at which it is sold on to buyers.
3. Some combination of both commission and spread.

Because the supplied goods would theoretically not be able to be differentiated due to the standardised interface, the only basis for differentiation of the services would be price. All that matters in this market is that both buyers and sellers are comfortable with the price the commodity is traded at. [31] Typically the extra cost of using a broker is tolerated by the market as they provide liquidity, neither the provider or consumer have to spend time finding potential counter parties at the right price. The broker essentially has to satisfy the requirements of the buying and selling parties, whilst simultaneously turning a profit.

Providers. Providers in the market are equipped with price-defining mechanisms which sets a dynamic price for compute resource dependent on market conditions, demand and the current level of utilisation. Providers negotiate with brokers, communicating with the resource management systems in order to ensure that SLAs can be upheld.

Consumers. Consumers have functions that calculate the optimal amount of resource that they require to maximise their profits, whilst taking into account any constraints they may have, such as a limited budget. Once the requirements are realised, negotiations take place to develop SLAs with a range of brokers, in order to attempt to cut the best possible deal. Once the chosen broker delivers the resource, the consumer is free to deploy their services on the leased infrastructure.

2.3 Pricing Models

In 2008, a group of researchers at HP Labs released a paper highlighting the common issue of compute intensive applications often suffering from bursty usage patterns. This had the unfortunate side-effect of sometimes resulting in demand for IT infrastructure resources to exceed the installed capacity within an organisation. At the time, the cloud computing paradigm was beginning to gain traction and it was noted that the additional demand could be satisfied by 'providers of IT services who satisfy demand for a given price.' [49]

It was recognised that there was a need for a pricing scheme for a utility form of IT provisioning, due to the uncertain nature of demand for computing requirements. On the one hand, providers need to encourage the usage of their offerings whilst gaining an insight into usage patterns in order to enable effective statistical multiplexing. On the other hand, the customer requires a simple way to anticipate and hedge the need for uncertain demand and costs that it will add to their operations.

2.3.1 Swing Options

In a previous piece of work [9], Clearwater and Huberman proposed that a swing option mechanism could be utilised for efficient pricing of IT resources. Through the payment of an upfront strike price, an option is a type of financial derivative that gives the holder the right, but not the obligation to use a resource when the option contract matures. If the option is taken, the remaining cost is paid. This is a flexible method of purchasing infrastructure, particularly favouring entities whose demand is difficult to predict as it was originally developed for trading energy.

Despite the simple nature of the mechanism, it turns out that pricing a swing option is a difficult task. Two important problems needed to be resolved:

1. The user needs a method of predicting future resource requirements.
2. The provider requires a method of encouraging the user to be truthful when stating the likelihood of exercising their options.

Although the first issue can be resolved through users estimating costs of reservations through utilising historical data, the second is more difficult to solve. Cases such as new users make it difficult to predict demand, and some users may be untruthful in order to gain a pricing advantage.

2.3.2 The WZH Model - Truth Telling Reservations

In order to overcome the pitfall of untruthful consumers when using swing options, Wu, Zhang and Huberman designed an option contract with a pricing structure that encourages customers to reveal the true likelihood that they will use a resource. They showed that a nonlinear pricing scheme leads to both truthfulness and profitability for both the consumer and provider. Henceforth, this shall be referred to as the WZH Model.

This is presented as a two period model. In the first period, the consumer knows the probability of using the resource in the second period. A reservation is then purchased, where the price depends on the probability submitted. A third party *coordinator* then aggregates the reservations from all consumers and purchases the corresponding quantity from the provider. The resources are purchased in the second period.

Consider n users who live for two discrete periods. Each user is able to purchase a unit of resource from a provider to use in the second period, either at a discounted rate of 1 in period 1, or a higher price $C > 1$ in period 2. In period 1, each user i only knows the probability p_i that they will need the resource in the next period. The requirement is not known for certain until the second period arrives. An assumption is made that the distribution of the consumers needs are independent.

The Coordinator Game

A third agent, the coordinator, is introduced who can profit from the aggregation of the consumer's probabilities while absorbing their outstanding risk. This is achieved through the following two period game. The terms described are completely transparent to all parties, prior to the first step.

1. (Period 1) The coordinator asks each user to submit a probability q_i, which does not have to be the real probability p_i, that the user will need one unit of resource in period 2.
2. (Period 1) The coordinator reserves $\sum q_i$ units of resource from the resource provider at the discount price for use in Period 2.
3. (Period 2) The coordinator delivers the reserved units to consumers who claim them. If the amount reserved is not sufficient to satisfy the demand, additional resources are purchased from the provider at the higher price C in order to meet the demand.
4. (Period 2) Consumer i pays:
   $f(q_i)$ if resource is required
   $g(q_i)$ if resource is not required
   where $f, g : [0,1] \to \mathbb{R}^+$ are two functions specified later.

In order for the Coordinator to profit, the following conditions must be satisfied:

- *Condition A.* The Coordinator can make a profit by providing this service.
- *Condition B.* Each of the consumers prefers to use the service provided by the coordinator, rather than deal with the resource provider directly.

Furthermore, the following truth-telling conditions are not completely necessary, but are useful for conditions A and B to hold.

- *Condition T1.* Each user submits their true probability p_i in step 1, so that they expect to pay the least later in step 4.
- *Condition T2.* In step 3, if a user does not require a resource in period 2, it is reported to the coordinator.

The following specific case was proved to meet these criteria, where k is a constant chosen to alter the price paid by the customer. In the proofs, the value of k was set to 1.5, while C was set at 2.

$$g(p_i) = \frac{kp_i^2}{2} \tag{2.1}$$

$$f(p_i) = 1 + \frac{k}{2} - kp_i + \frac{kp_i^2}{2} \tag{2.2}$$

The model can be used to forecast demand as it encourages users, through a reduction in cost, to submit an honest estimate of future requirements. If the user is submitting honestly (which Wu *et al* proved to benefit the user) they will expect to pay:

$$w(p_i) = p_i f(p_i) + (1 - p_i)g(p_i) \tag{2.3}$$

If they instead choose to purchase resources directly from the provider, they will expect to pay $w(p_i) = Cp_i$ where C is the on-demand cost of a resource.

The contract can be considered an option because $g(p_i)$ is the minimal amount the user has to pay in any event, which can be requested in period 1. The user then only has to pay $f(p_i) - g(p_i)$ in period 2 if the resource is needed at that time. Therefore, by paying the initial amount $g(p_i)$, the user receives the right, but not the obligation to purchase a unit of resource at $f(p_i) - g(p_i)$ in the second period.

2.3.3 The Effects of Truthfulness and Demand in a Simulated Market

While Wu *et al.* provided a theoretical description and analysis of their model, which they claimed promoted truth-telling amongst the resource buyers population, no empirical results existed to reinforce its effectiveness. Rogers and Cliff addressed this issue through a detailed exploration of a more heterogeneous instantiation of their model. [35] Using methods similar to replicator dynamics, which is commonly employed when studying evolutionary processes, they determined whether the model actually favoured honesty, encouraged agents to become more honest and whether honesty actually benefits the coordinator in a simulated setting meant to model a real-world, multi-user scenario. The heterogeneity of probabilities and levels of honesty can be used to justify the work as an extension of the original theoretical foundations and supporting analysis.

An algorithm was developed that assigns an 'honesty' value H_i to each user, describing the accuracy with which a probability of future resource requirement is provided to the coordinator. $H_i = 1$ describes a user who is considered 'honest', while $H_i < 1$ describes a user who is 'dishonest'. A replicator dynamics approach was employed, mutating a random user to alter their honesty value every two timesteps. Each user was also supplied with a probability p_i of requiring the resource in the next period. The cases of a user underestimating and overestimating its future requirements were taken into account, submitting the probabilities $q_i = H_i \cdot p_i$ and $q_i = (1 - H_i)(1 - p_i) + p_i$ respectively. The users are charged as per the WZH options model, paying a reservation premium in the first period and the outstanding charge if they wish to utilise the resources in the second period. If the coordinator did not reserve adequate resources, additional resources must be purchased for the on-demand price of C. The steps were repeated a number of times to ensure that each user provided a range of probabilities to the coordinator using the same honesty level.

Does the Model Favour User Honesty?

The aim of the first experiment was to determine whether a dishonest user would benefit more than an honest user. The most likely scenario to occur is a user submitting a probability of less than they predict they will require, so that they pay a lower premium but still have the right to access the resource should they actually need it. The results showed that users who are more honest generally pay a smaller price per resource than those who lie. The mean cost over infinite samples was found to be 1.25 when $H_i = 1$ and $p_i = 1$, and 1.75 when $H_i = 0$ and $p_i = 0$, results that were matched through the simulation.

Furthermore, the simulation found that the mean cost per utilised resource is always cheaper than going directly to the resource provider, therefore fulfilling *Condition B* of the model by giving incentive to users to always use the coordinators services through a mean saving.

Does the Model Encourage Users to Change Behaviour?

In order to test whether users alter their honesty over time, the simulation was rigged to alternate between mutating H_i such that user i becomes more honest and mutating H_i such that user i becomes less honest. This aims to model a real world scenario where a user may spontaneously decide to become more or less honest to see if they financially benefit as a result.

The simulation results conclusively showed that over time the user base as a whole appears to become increasingly honest as further mutations occur. Two experiments were conducted, one with the user base starting with an honesty of 0, and another with an honesty of 1. When starting with 0, the honesty of the population grows steadily over time, appearing to show steady progression to 100%. When starting with 1, the level of honesty remains stable near 100%, leading to the conclusion that users

converge to a stable state of 100% honesty. As the users are compelled to submit true probabilities, *Condition T1* of the WZH Model holds.

Do Changes in User Behaviour Benefit the Coordinator?

The simulation was altered to record the coordinators profit as the users gained honesty, to see how the behaviour affected them. The results showed that as the honesty of the user base increases, so too does the coordinators profit. *Condition A* therefore holds, as the coordinator can make a profit from the model.

The Effects of Market Demand on Truthfulness

In research furthering the original empirical analysis [36], Rogers and Cliff aimed to determine the performance of the WZH Model in a market where demand for resources is both variable and unpredictable. This was achieved through the extension of the simulation to include heterogeneous variations, forcing the users to make decisions based on scarcity or abundance of resources. The experiment aimed to discover the extent to which the model continues to hold as the simplifying assumptions are relaxed. They found that the coordinator benefits more when resources are abundant, although profits did not always increase as the honesty of the users increased. Additionally, it was found that an optimum honesty occurs when there is no surplus or deficit of resource purchased by the coordinator.

2.3.4 Cloud Brokerage: Rogers and Cliff's Modified WZH Model

Following the positive results obtained in their previous research efforts to verify the validity of the WZH Model, Rogers and Cliff recognised an opportunity for an extension that could potentially increase the viability for such a model for real-world application. [38] In Wu *et al*'s original proposal, the provider has the ability to predict future demand through users reserving resources one period in advance of purchasing them. Although it can be argued that this aids the resource provider in managing variable costs such as staffing, it is unlikely to be of use for planning larger investments. If the information is used to plan additional capacity in the next period, it could cause the provider to make a technology investment with no guarantee of its future utilisation. In order to help offset some of this risk which is taken on by the coordinator, a new model is proposed, leveraging use of commonly used provider pricing methods.

- (Period 1) The coordinator has a choice to purchase a provider reserved instance. A reserved instance provides the coordinator access to resources for a fixed time (12 or 36 months), charging an initial up-front fee followed by a lower unit time cost.
- (Period 2) The coordinator can purchase an on-demand instance, with no upfront fee but an increased unit time cost.

Provider Benefits

This approach gives the provider a long term view of future demand. Additionally, the information gained on likely utilisation in the next period could serve to efficiently schedule workloads on servers to promote optimal usage of resources at the providers disposal. [42] Another benefit is the payment of costs up front as this demonstrates that the coordinator is confident of future usage, prompting the provider that investment in new infrastructure could be justified.

Coordinator Benefits

Access to a resource for a longer period of time allows the coordinator to act as a kind of wholesaler - allowing them to provide their reserved instances to any customer who needs them on a month to

month basis, in turn reducing wastage.

User Benefits

The user now has the ability to reserve a resource without the issue of having to pay full price, should they not need it in the future, reducing total expenditure and lowering risk. One downside of this model however is that the user must anticipate that the resource will be utilised for the full month in order to gain the financial benefit.

The Modified Model

1. (Period 1) Each user i submits a probability, q_i to the coordinator, which does not have to be the real probability p_i, that they will require a unit of resource in Period 2.
2. (Period 1) The coordinator needs to reserve $\sum q_i n_i$ units of resource from the resource provider to be executed in the next month.
   (a) If the coordinator has previously purchased enough reserved instances for the predicted demand, no further instances are purchased.
   (b) If there are not enough reserved instances to cover the demand, additional reserved instances may be needed. Considering the performance of additional instances over the previous 36 months:
   $$\textit{Previous Demand Profile } \boldsymbol{A} = [d_{t-36}...d_t]$$
   $$\textit{Future Capacity Profile } \boldsymbol{B} = [c_t...c_{t+36}]$$
   $$\textit{Deficit Profile } \boldsymbol{C} = \boldsymbol{A} - \boldsymbol{B}$$
   For each resource required, the *Marginal Resource Utilisation (MRU)* is the ratio of items in $\boldsymbol{C} > 0$. The *MRU* is a fraction of the life of an additional reserved instance that will be utilised over the next three years based on past performance. The *Threshold* (from now on denoted θ) is a ratio determined by the coordinator to maximise profit.
   (c) If $MRU > \theta$, the coordinator will buy a new reserved instance, as it is likely to be used enough to make a return on investment.
   (d) If $MRU < \theta$, no instance will be purchased, as it will likely be cheaper for the coordinator to purchase an on-demand instance next period.
3. (Period 2) The coordinator delivers the reserved units to consumers who claim them. If the amount reserved is not sufficient to satisfy the demand, additional resources are purchased from the provider at the higher price D_h in order to meet the demand. For reserved instances, the reduced cost of R_h is paid.
4. (Period 2) Consumer i pays:
   $f(q_i)$ if resource is required
   $g(q_i)$ if resource is not required
   where $f, g : [0,1] \rightarrow \mathbb{R}^+$

Model Parameters and Performance

Once more, a simulation was constructed in order to test the performance of the newly suggested alterations to the pricing model. In an attempt to replicate the likely real-world scenarios that would be encountered, datasets were obtained from the UK National Statistics Office on the Non-Seasonally Adjusted Index of Sales from 1988 to 2011, using market segments with a strong relationship to IT usage. The data used to predict demand varies differently over the period, allowing the model to tested across a range of market conditions. Simulated prices from the provider were taken from the cost of an Amazon Web Services small EC2 instance.

Users were charged by the provider based on Wu *et al*'s $f(q_i)$ and $g(q_i)$, although because the standard monthly on-demand cost of the service provider was around $60, the coordinator can scale the values so long as Condition B of the model is met. For their original experiments, a scale factor of 60 was used, although in later work [37] it was found that a scale factor of 35 seemed to be optimal. The threshold is another parameter that can be performance altering. In [37], brute force testing showed that on average, a threshold value of 0.8 appeared to be optimal, although clearly this was not the case in all instances, potentially opening up an opportunity for further work in this area.

The results of the simulation showed that the coordinator was able to profit, even when not optimising across the different market profiles provided, indicating that it has an ability to prosper in a variety of market conditions. If the threshold was set to 0, annual profit generally varied with market demand, but often led to reservations being made that were not utilised in the future. However, using the optimal threshold for each market saw improvements in every instance, in varying amounts. This is due to the coordinator only buying reserved instances when it believes that it will be utilised enough to pay back its cost, reducing expenditure and maximising profits.

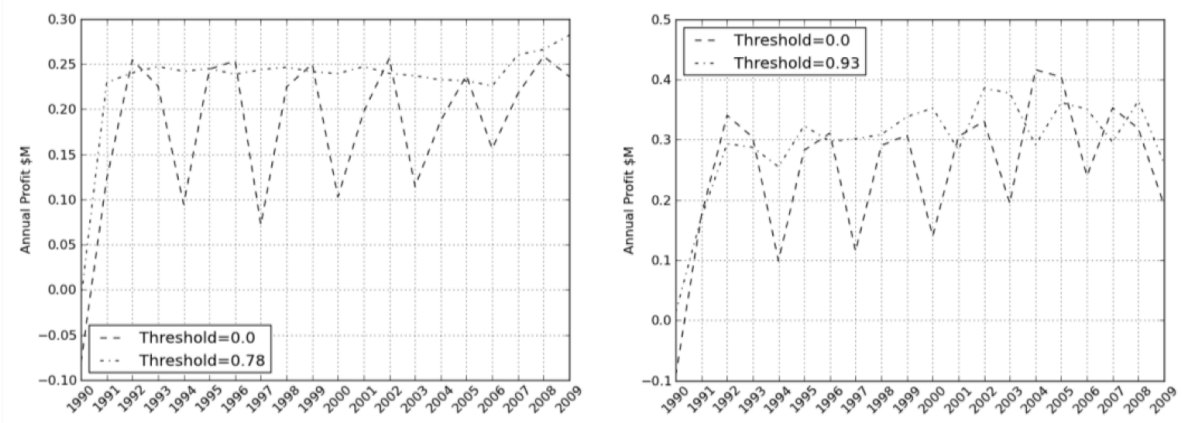

Figure 2.3: Left: Non-Store Retail. Right: Computer and Telecoms Equipment.

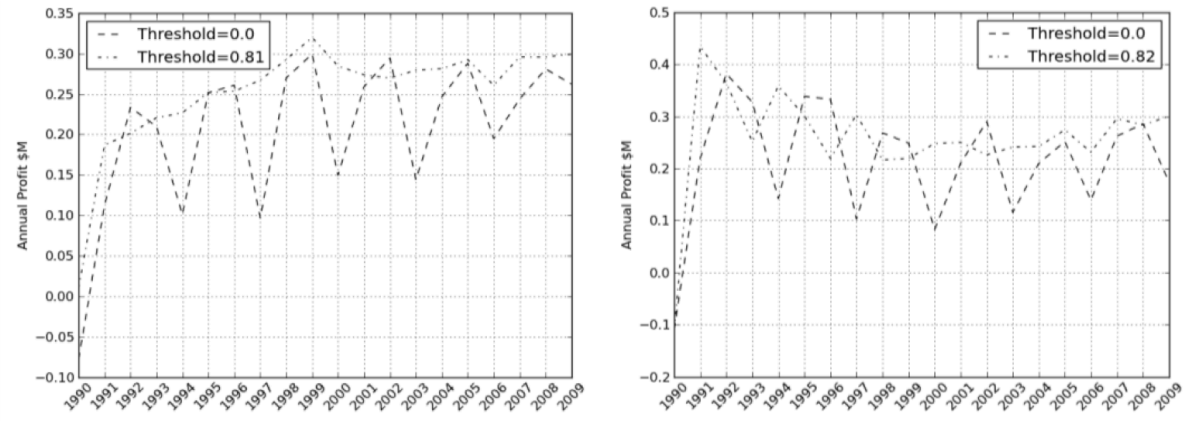

Figure 2.4: Left: Non-Store Retail: Small Business. Right: Non-Store Retail: Large Business.

2.4 Simulating Cloud Provision

As cloud computing gains popularity and traction as an effective computing platform in a multitude of industries, it becomes increasingly essential to identify methods that increase efficiency, especially considering the vast amount of energy required to power a modern data centre. Research interests are being piqued in the area, although due to the inherently secretive and proprietary of the current market solutions, there are no industry-standard tools available for experiments to be performed. Due to the size and costs associated with owning and running data centres, the only viable research approach is through simulation. An effective simulator can model the complexity of a real data centre, allowing tweaks, changes and additions to easily be tested and reproduced with no risk to real equipment. Despite the lack of official tools, there have been several frameworks developed by universities and other research organisations, including CReST from the University of Bristol.

2.4.1 Cloud Research Simulation Toolkit

CReST [23] is an cross-platform open-source framework for simulation and modelling of ultra-large scale data centres, which has been in development at the University of Bristol through the LSCITS initiative over the last two years. It is primarily built in the Java programming language, but has a comprehensive XML based configuration system. Designed to be customisable and with extensibility in mind, CReST is a modular, event-based framework that provides users with the ability to design their own data centres, along with the options to turn features on and off for each simulation. The opportunity to help further develop an existing open-source simulation platform made CReST a good choice for performing brokerage experiments.

At the time of developing the brokerage components for CReST, multiple other modules existed at various stages of completeness. These included data centre configuration, utilisation, including demand and scheduling of job instances, server failure and replacement, energy and cost, thermal and middleware. Despite the majority not being particularly useful for the experiments required within this project, their existence may well be significant in future research. Each of the modules run over different timescales and therefore for simulations to run in a suitable amount of time, modules that operated at a minute by minute level, such as thermal, were disabled for the brokerage experiments, which operated at a monthly level.

CReST Builder

The framework is formed from two different tools, the builder and the simulator. In order to generate customised simulations, the CReST Builder GUI can be used. This allows the simulation parameters to be specified, the data centres to be generated and customised and the desired modules to be toggled on or off independently. The builder interface is comprehensive, allowing the user to drill down to the level of detail required to accurately perform their experiments. Once the settings are established, the parameters for the simulation are saved as a gzipped XML file ready to experiments to be performed. One of the many different customisation views available in the builder can be seen in Figure 2.5.

CReST Simulator

The simulator itself is another interface that utilises the configurations designed using the builder. Once more, there is a plethora of information at the users disposal with customised views available for many of the modules, graphing and different overlays of the data centre to monitor performance and events that are occurring. Each of the modules enabled for the simulation has the ability to output logs to allow the user to perform further analysis at a later date. One of the views of a simulation can be seen in Figure 2.6.

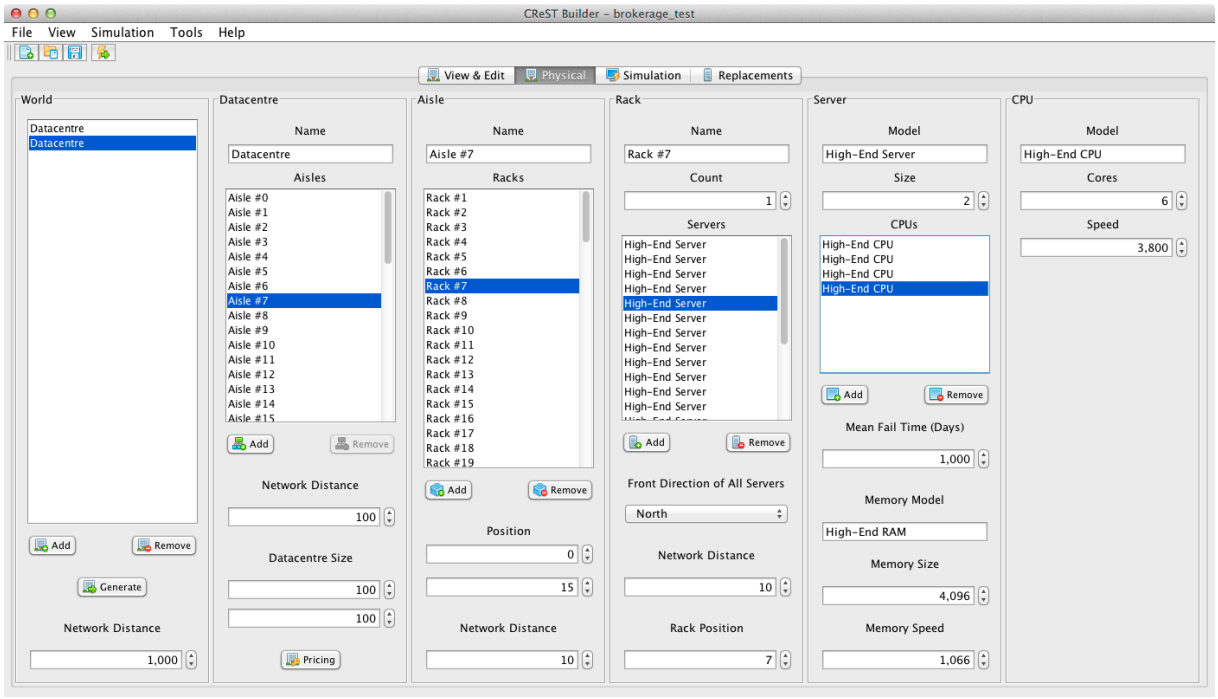

Figure 2.5: View of CReST Builder Data Centre Specification

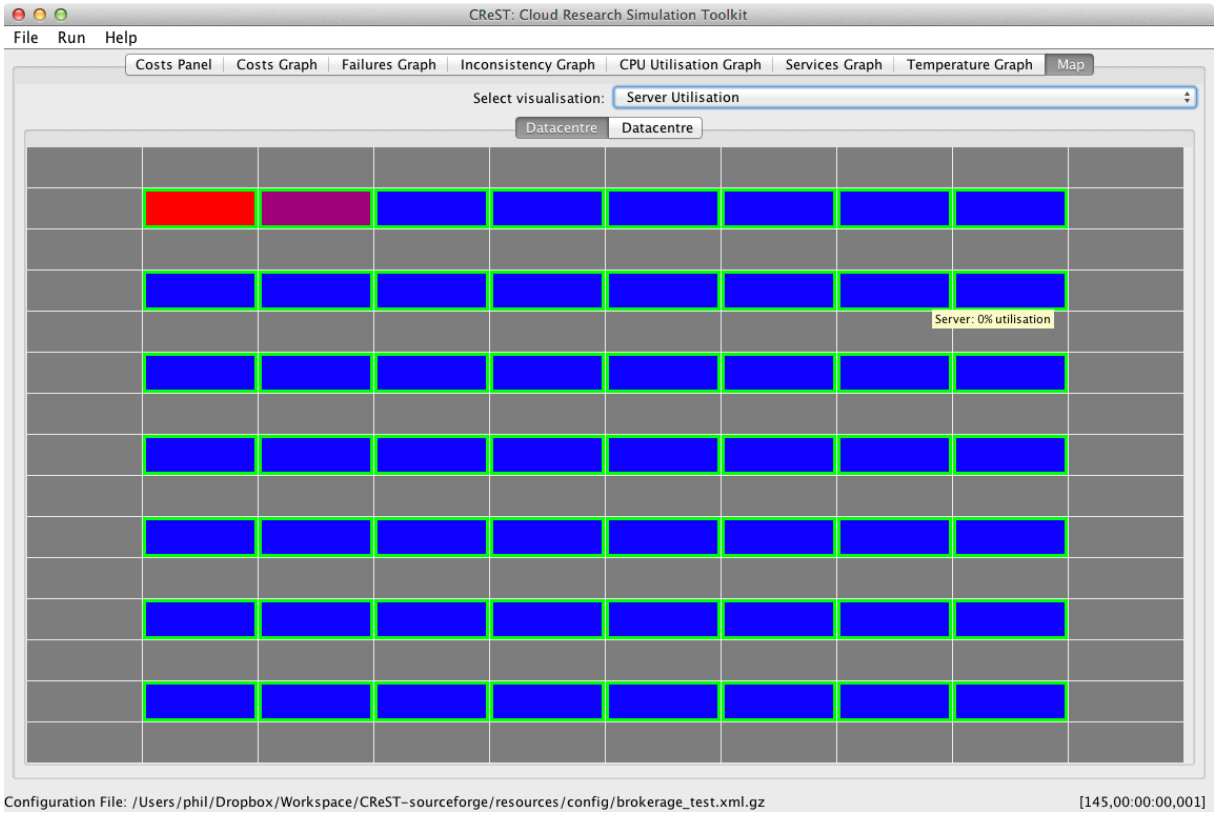

Figure 2.6: View of CReST Simulation Data Centre Server Utilisation

It is important to note that CReST is at an early stage of development and many of the features are currently incomplete. A significant portion of the technical challenge of this project involved working with this system, tracking down issues in existing code and developing further components, both for the benefit of this project and future research. Due to the young nature of the platform, this will be the first research performed employing the platform for experimentation. Once finished, any work completed on the platform for this project will be released in order to help its development and encourage further use in research.

Chapter 3

Project Execution: Initial Investigation

3.1 CReST Module Design

The initial stage of the project involved the adaptation and realisation of Roger's enhanced WZH model as a brokerage module within the CReST framework. In order to accomplish this task, a full understanding of the methodologies utilised to build CReST was required, along with the identification of dependencies that the component would be constructed upon.

CReST employs a custom events based architecture and in order for the brokerage component to correctly integrate and interact with the other pre existing functionality, it would need to be modelled in the same manner. Major components include the *Simulation Runner* and *Event Queue*, which together allow the different elements of the environment to perform their tasks and interact. The behaviour of these abstractions is summarised in Figure 3.2 on page 24.

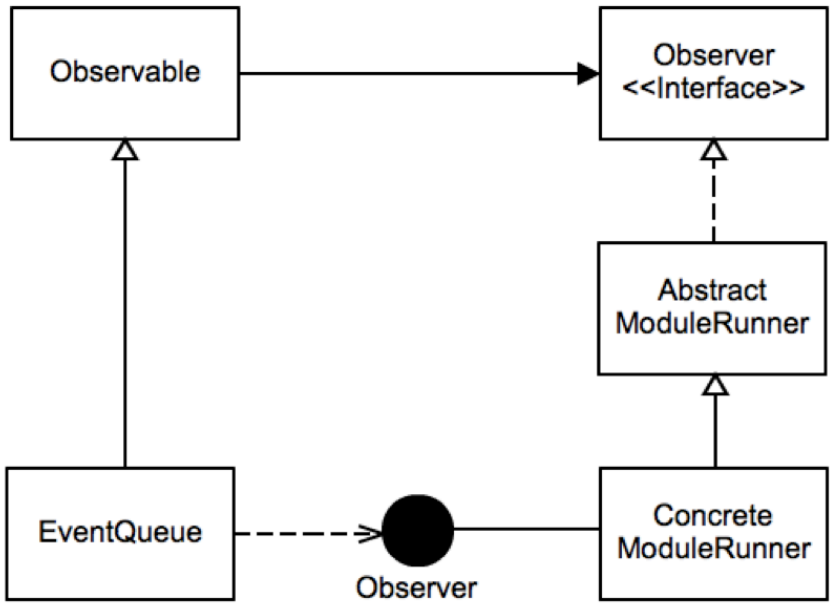

Figure 3.1: Event Queue Module Interface. Reproduced from [8]

The patterns in use include an `Observer/Observable` interface, which help to forge a strict separation between the Model (which includes the Simulation Runner, Event Queue and generally the underlying application functionality) and View (the Graphical User Interface). The interfaces can be interpreted as in Figure 3.1.

In order to create a new module, the following components are a requirement:

1. Concrete Module Runner - The main Observer for each module, it receives simulation Events popped from the Event Queue, acting if appropriate and potentially adding new Events.
2. Configuration Parameters Interface - CReST employs a system of configurable XML options for defining the parameters of a simulation. Each module has its own possibilities and therefore an interface is required to assemble the values in the simulation environment.
3. Module Event Thread - The purpose of the event thread is to initialise the module, generating and adding any events to the Event Queue that may be required to perform its task and kick start the simulation loop.
4. Other Relevant Module Events - Any interactive activity taking place within a simulation is performed through an Event. An Event could be anything from a server failure to writing a log of current performance in a particular module.
5. Other Relevant Concrete Components - These are module-dependent components and basically encompass any behaviour that is not already covered by the required classes.

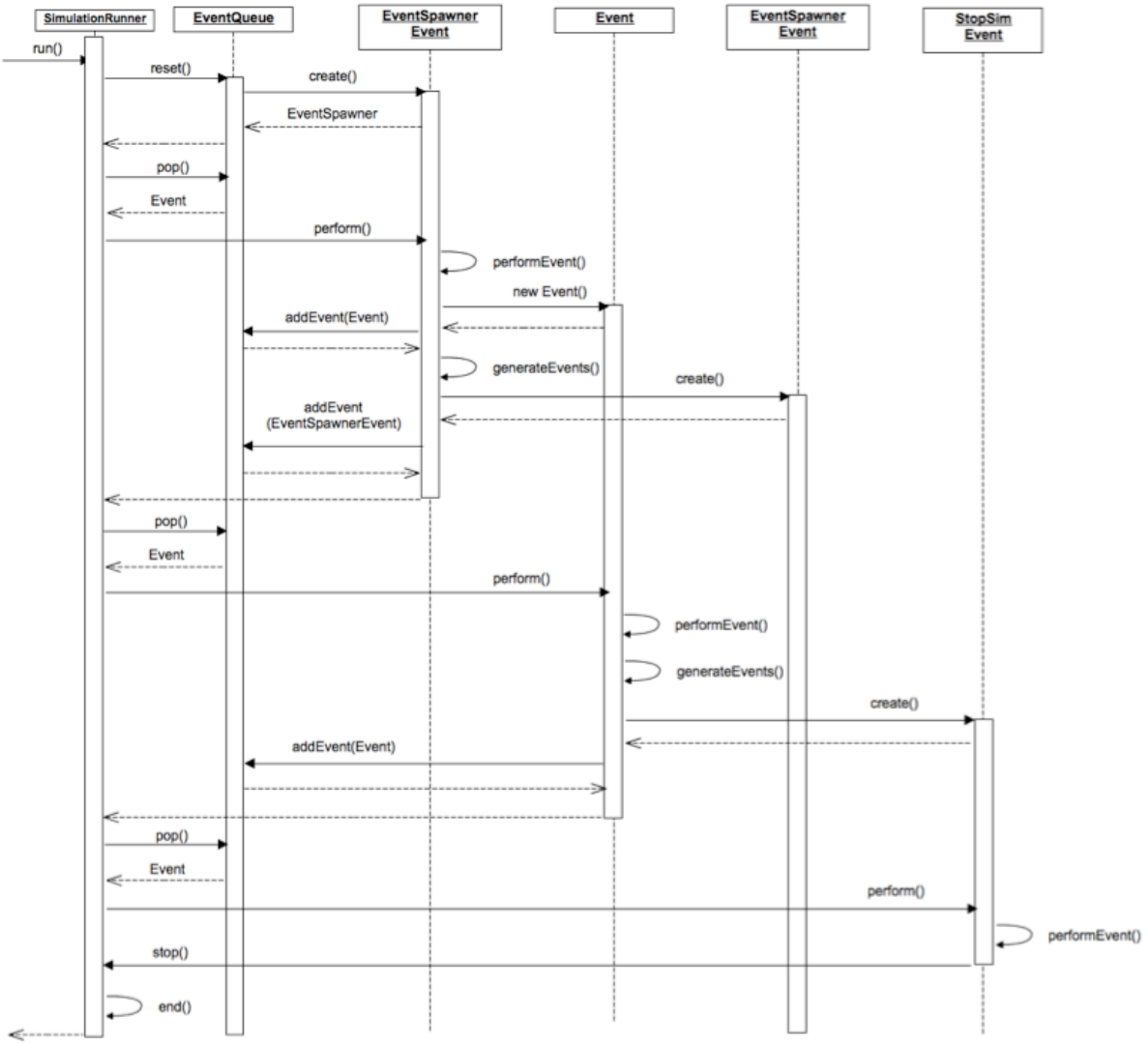

Figure 3.2: Interaction of the SimulationRunner with Events in the CReST Framework. Reproduced from [8]

3.2 Brokerage Components

The event based structure utilised by the framework appeared apt for the periodic nature of the WZH Model. Two significant entities are a requirement for the implementation, representing the behaviour of a broker (`BrokerAgent`) and a cloud instance consumer (`UserAgent`). The periods in the model are dubbed the 'reservation' stage and the 'execution' stage, referring to periods 1 and 2 respectively.

3.2.1 The Broker

As in the abstract model, the broker agent has two distinct stages at which numerous events occur that are essential to its role within the market. Several parameters are defined which can have a dramatic, yet non-linear, effect on its performance, including the marginal resource utilisation threshold and cost factor, the fee multiplier required to bring the WZH fee calculation into an appropriate range corresponding to the charges of the cloud provider.

Reservation Stage. The primary objective of the reservation stage is to ask each of the users to submit a list of the instances they are likely to require, along with a likelihood estimate that they will require it in the next month. The requests are charged and probabilities summed followed by the calculation of the number of required reservations. The model at this point could be interpreted in a number of different ways, potentially leading to differing results. Two main methods were identified and implemented in this simulation. Although on the face of it they appear to be very similar, ultimately the performance diverges significantly. Once the number of instances to purchase is established with either method, a Quote Request Event is created and submitted to the Event Queue. Finally, a Broker Execute Event, which initiates the following Execution period is created for the next month.

Execution Stage. The execution stage equates to stage two of the WZH Model, taking place in the month following the reservation stage at the point when the users need the instances to run their jobs. At this juncture the concern is providing the users with the number of instances that they require, utilising the cache of reservations at the broker's disposal and making up any deficit with on-demand instances. To this end, each user is solicited for their present instance requirement, are charged in kind using price $f(q_i) - g(q_i)$ (see equations (2.2) and (2.1) on page 15) and the total demand is tracked. Using the demand figure in conjunction with the known amount of instances at its disposal, the on demand requirements are computed and are purchased through a Quote Request Event. At the end of the execution stage, the broker is charged for the instances by the provider. Once the execution steps are completed, the simulation moves on to the reservation period for the following month.

Reservation Method 1: Full Reservation Hedging using the Single, Current MRU

This is achieved through subtracting the outstanding reservations available next month from the number of reservations requested by the users in total. In order to make a decision about purchasing the further instances, the current MRU is computed and tested against the defined threshold. If the MRU is above the threshold, the deficit number of reservations are purchased. This may also lead to a situation where there is no deficit in the next month, but the MRU has also crept above the threshold, in which case no further instances will be procured. The pseudocode can be seen in Algorithm 1.

Reservation Method 2: Iterative Hedging of Individual Reservations

This interpretation involves the broker determining if purchasing another instance is likely to be fully utilised over its lifetime, up to a maximum defined by the totalled probability of the requested reservations from the users. This leads to purchasing one reservation at a time, recomputing the MRU after each purchase (although in this simulation, temporary forecasts are used to indicate the number required and they are purchased after the process has ended).

Algorithm 1 Interpretation of Reservations Hedging - Method 1.

```
1:  procedure HEDGERESERVATIONS(totalProbability, priorRequired, capacity)
2:      reservationsToMake ← ceil(totalProbability) − capacity[0]
3:      if reservationsToMake < 0 then
4:          reservationsToMake ← 0
5:      end if
6:      deficit ← newList()
7:      totalDeficit ← 0
8:      for i ← 0, priorRequired.length() do
9:          monthDeficit ← priorRequired[i] − capacity[i]
10:         deficit.add(monthDeficit)
11:         if monthDeficit > 0 then
12:             totalDeficit ← totalDeficit + 1
13:         end if
14:     end for
15:     margin ← totalDeficit/deficit.size()
16:     if (margin > threshold) then
17:         EventQueue.addEvent(QuoteRequest(reservationsToMake))
18:     end if
19: end procedure
```

Algorithm 2 Interpretation of Reservations Hedging - Method 2.

```
1:  procedure HEDGERESERVATIONS(totalProbability, freeReservations, capacity)
2:      numHedge ← ceil(totalProbability)
3:      reservationsToMake ← 0
4:      for i ← 0, numHedge do
5:          if freeReservations[month + 1] >= 1 then
6:              freeReservations[month + 1] ← freeReservations[month + 1] − 1
7:          else
8:              mru ← 0.0
9:              if month >= resLen then
10:                 monthsDeficit ← 0
11:                 for j ← month − resLen + 1, month + 1 do
12:                     deficit ← nAgents ∗ getDemand(j + 1) − capacity[j + resLen]
13:                     if deficit > 0 then
14:                         monthsDeficit ← monthsDeficit + 1
15:                     end if
16:                 end for
17:                 mru ← monthsDeficit/resLen
18:             end if
19:             if mru > threshold then
20:                 reservationsToMake ← reservationsToMake + 1
21:                 for k ← month + 1, month + resLen + 1 do
22:                     freeReservations[k] ← freeReservations[k] + 1
23:                 end for
24:             else
25:                 break
26:             end if
27:         end if
28:     end for
29:     EventQueue.addEvent(QuoteRequest(reservationsToMake))
30: end procedure
```

Therefore the process involves iterating through the total amount that could be hedged, first of all swallowing up the amount of free reservations that are already owned, then performing the calculations to check if another reservation is desired. After a reservation is purchased, the next reservation to hedge is swallowed. The algorithm itself leads to significantly fewer reservations being purchased than the alternative interpretation, although takes a significant amount of time to perform all the calculations. Optimisation for running speed was one of the first concerns noted when implementing this method. The pseudocode can be seen in Algorithm 2.

The Hedging Interpretations: A Quick Comparison

As previously mentioned, despite the apparent likeness of the two approaches, the behaviour between the two implementation methods bifurcates noticeably. This can be attributed to both the volume of reservations purchased at each time step and the spread of purchases over time. The trend observed with method 1 was a tendency to purchase a significant number of reservations in a single month in an attempt to cover the demand for the following 36 months. This occurred for a few reasons. Because the number of instances to hedge was limited by the capacity for the next month, which will always have the least deficit due to reservations running out over time, less the anticipated demand (using past data for prediction), there were a significant number of months where the MRU threshold was breached but there was sufficient current resources in the next month to cover the demand. This theme tended to repeat for a number of months with the owned reservations slowly running out (in turn creating some very profitable years), but all of a sudden a month of high demand causes a significant purchase of reservations and in turn with such a high upfront fee, a huge loss spike. This problem persisted regardless of the threshold in use and even using techniques such as averaging the demand over a number of months, intended to stabilise the purchasing over a number of months rather than all at once.

On the other hand the second method, which after consultation with Rogers was found to be the original intention, results in a much smoother spread of reservation purchases when employing a suitable threshold. It also provides the additional benefit of a more consistent profit margin year-on-year. Although it is likely that using method 1 will indeed result in cumulated profit over time, as a business proposition a more consistent result would be desirable. The results using the same demand profile and simulation settings can be seen in Figure 3.3. Further work in the project will utilise the second method, now it has been established to be the original intention of the model's author.

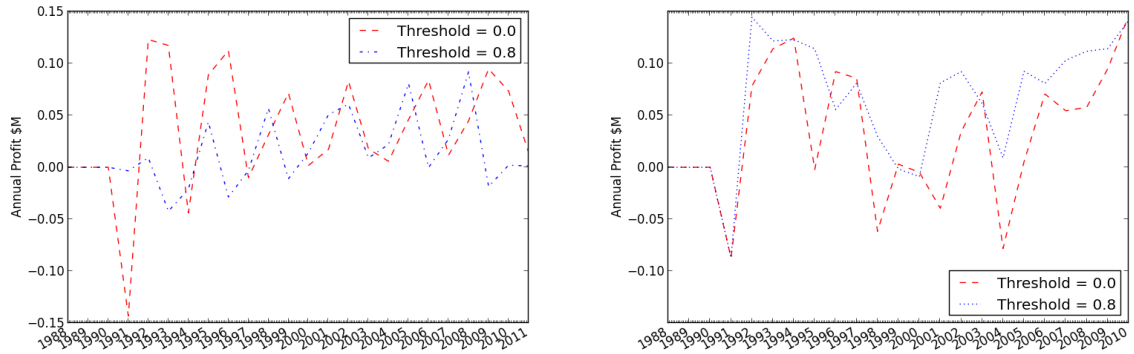

Figure 3.3: Comparisons of Interpreted Performance of Methods 1 and 2 Respectively.

In order to help understand the difference, a simple numerical example can be of aid. The following assumptions are made using both examples:

- There are 10 users submitting probabilities to the broker, for this particular month the **demand** is 0.6 and therefore 6 users will submit probabilities of requiring a resource.
- The broker is using a MRU threshold of 0.8 to determine how many reservations to purchase.

- Looking through the users histories, the average probability of them needing a reservation for that month are calculated and result in the following probabilities being submitted to the broker: 0.35, 0.5, 0.72, 0.16, 1.0, 0.96 and 0.49.
- The broker currently has an inventory of 0 reservations and the previous requirements and future capacity are as such for a span of 36 months:

  Previous Requirements = [4,6,2,1,1,3,4,6,6,7,8,9, 1,5,4,3,3,7,0,0,1,6,9,9, 3,7,2,2,1,5,8,1,1,4,10,8]

  Future Capacity = [0,0,0,0,0,0,0,0,0,0,0,0, 0,0,0,0,0,0,0,0,0,0,0,0, 0,0,0,0,0,0,0,0,0,0,0,0]

Using method 1, the algorithm proceeds as such. With the probabilities summed, the number of *reservations to make* equates to (0.35 + 0.5 + 0.72 + 0.16 + 1.0 + 0.96) - 0 = 3.69. Therefore $ceil(3.69) - 0 = 4$. From here, the deficit profile is computed which in this instance is identical to the previous requirements, as all entries in the future capacity are 0. Deficit = [4,6,2,1,1,3,4,6,6,7,8,9, 1,5,4,3,3,7,0,0,1,6,9,9, 3,7,2,2,1,5,8,1,1,4,10,8] This profile in turn means that the total deficit is 34 out of 36 months, leading to an MRU of 34 / 36 = 0.94. Because this value is higher than the broker's threshold, the broker will proceed to purchase the *reservationsToMake*, which in this instance is 4.

Method 2 on the other hand produces a slightly different chain of events. Once again, the number to hedge ends to being 4 (ceil(3.69)). Because no reservations are currently owned, there are no free reservations at the start and therefore the algorithm proceeds to test if the first instance is needed. Again, the deficit profile is computed which in this instance is the same as method 1; the MRU ends up being more than the broker's threshold and therefore the first reservation is purchased. In the second iteration of the algorithm, there is now a 'free' reservation and so the second reserved instance is not purchased. The third iteration finds that no reservations are free and proceeds to test if a further reservation will be fully utilised if purchased. The computation of the deficit profile is now different, as the purchase of the first reservation has altered the future capacity profile (which is now filled with 1s as the instance purchased will be owned by the broker for the next 36 months). The new deficit = [3,5,1,0,0,2,3,5,5,6,7,8, 0,4,3,2,2,6,-1,-1,0,5,8,8, 2,6,1,1,0,4,7,0,0,3,9,7] which leaves 27 months in reservation deficit. The MRU therefore is computed as 27 / 36 = 0.75, which is in fact less than the broker's threshold and therefore the reservation is not purchased. Furthermore, as no further reservations will be purchased due to this constraint, the algorithm does not proceed to perform hedging the final instance. Method 2 ends up purchasing only 1 reservation in total.

3.2.2 The Users

In correspondence to the two stages distinguished by the broker, the user agents have two main functions. During each time step, a number of the agents in tune with the relevant demand profile are selected to submit probabilities to the broker for a reservation. The probabilities submitted by each user is dependent on its personal execution history, which is built throughout the simulation. The past requirements for that particular month is averaged and submitted, as this is likely to give a fair indication of the likely usage of the resource, considering seasonal trends and other contributing factors. During the execution stage, a certain number of users are again selected based on the current demand profile (which has now changed as it is the following month). The designated users submit to the broker that they require a unit of resource and update their execution histories to reflect this. Other parameters such as the amount spent on resources are also kept track of in order to facilitate any experiments with a user focus.

3.2.3 Dependencies

Pricing

One of the most important dependencies of the broker is obtaining prices from the provider, as it enables the purchasing of reservations and, in turn, profit computations. This was a feature that was not originally built into CReST and therefore a custom module was needed. A simple mechanism was

required that allowed the broker to submit a request for a certain number and type of instance. Once the request is received, a price is calculated and a quote is returned to the broker.

For the initial broker implementation, static prices were used and therefore the price calculation was simplistic. The module was built to allow the prices for each simulation to be defined through configuration files, a common feature of existing CReST components. This allowed the definition of upfront and monthly prices for both reserved and on demand instances. In anticipation that further experimentation may incorporate markets where prices move, which is feasible if IaaS became standardised and allowed users to easily switch between providers, the module was built in such a way that building out a dynamic pricing component would be straightforward.

Demand

Due to the young age of the cloud computing paradigm, there appears to be nothing in the way of public domain real world data for resource usage that would be appropriate for simulating the WZH Model, due to the timescales required. It can be expected for there to be fluctuations in the demand of the users and it is also likely that there will be correlations between users in a similar sector. If the data was not assumed to be correlated, then it is likely that it would form a constant rate and provisioning for demand would be a much easier task.

Therefore this investigation utilised public domain data based on several different real world markets over a period of around 20 years, the same data set utilised in Roger's initial experiments [37]. The argument was made in the original paper that the data served as a good proxy for real-world demand for cloud computing resources in different market sectors. The profiles in question are known to exhibit peaks and troughs throughout the year. The data was obtained from the UK National Statistics Office and four market profiles were chosen based on knowledge of a strong link to Information Technology usage. The demand patterns were normalised values between 0 and 1 and in the simulation this value corresponds to a percentage of n users submitting requests for resources.

The market profiles in use are:

1. Rapid Growth (Non-Store Retailing: All Business). Figure 3.4.
2. Steady Growth (Non-Store Retailing: Large Business). Figure 3.4.
3. Recession and Recovery (Non-Store Retailing: Small Business). Figure 3.5.
4. Steady Market (Retail of Computer and Telecoms Equipment). Figure 3.5.

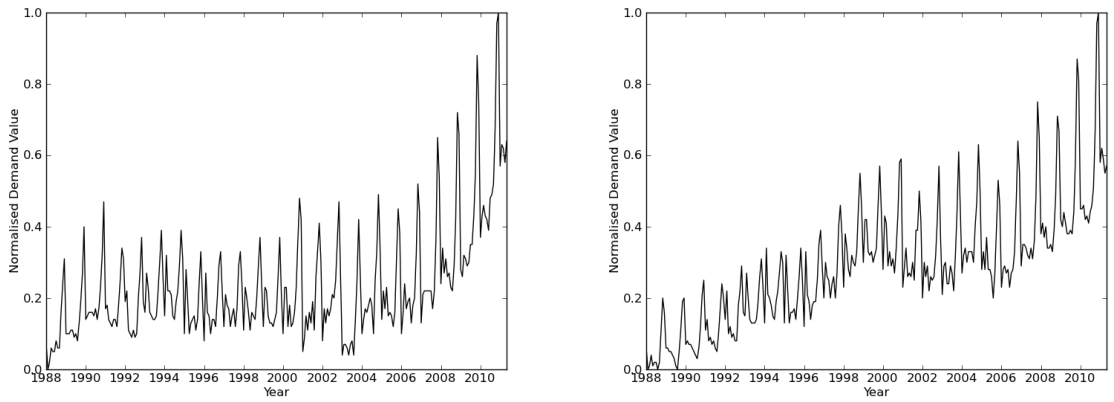

Figure 3.4: Rapid Growth and Steady Growth Market Profiles

In CReST, a module exists that handles the demand profiles loaded from an external comma separated value file. This module has the responsibility of maintaining the current month's demand for a certain profile, advancing the demand when required and stopping the simulation once demand for the simulation has been exhausted. The module was implemented in CReST prior to this investigation, but was

modified during this project to offer alternative methods of attaining the demand of a certain month when the implementation of the broker changed.

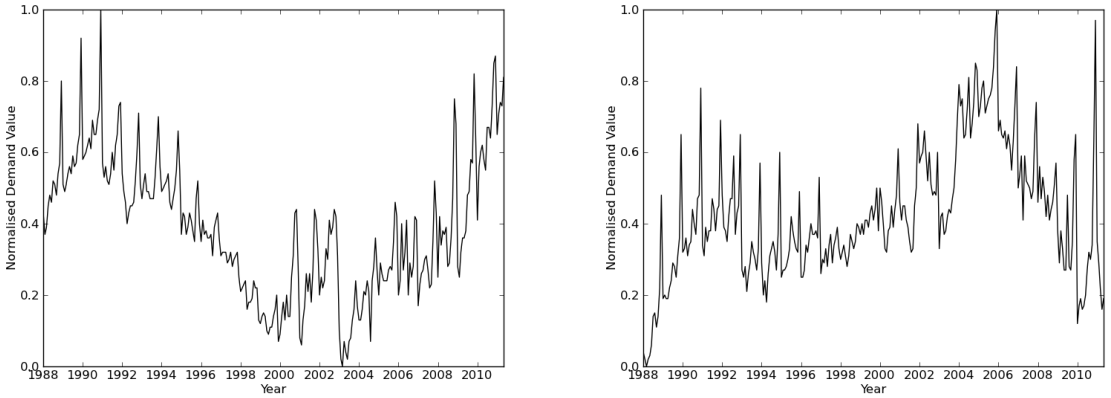

Figure 3.5: Recession & Recovery and Steady Market Profiles

Services

Because CReST is designed to simulate data centres, it is desirable for the brokerage module to interact with the simulation in some way in order to provide a means for further investigations in the future. This is achieved when the broker purchases reservations of either type. A 'service' is started in the data centre that the instance was purchased from in the simulation, utilising server resources. This gives a more rounded, realistic simulation. In the future, if high usage in a data centre could affect the cost of purchasing instances, experiments could be performed to see how this in turn affects the broker.

3.2.4 Events

As CReST is an event-based framework, the modules interact with each other using events that are inserted in the timeline to be executed. For the implementation this means that the broker submits requests for prices to the provider through *PriceRequestEvents* and replies are sent in the same manner. Therefore, each of the modules constructed during the project required a series of events to be designed when cross communication was required. Abstract thinking was therefore required to design them in such a way so that execution occurs as desired.

3.3 Experimental Setup

3.3.1 Simulation Parameters

For all the experiments performed in the project using CReST, a set of configurations were required in order to specify the required modules and parameters for each particular simulation. The fundamental set of these are specified in a gzipped xml configuration file that can be generated and edited using the CReST builder interface. Due to the nature of the event timings within the framework, it is not desirable to simply leave all of the modules on. For example, the module designed to simulate temperature within the data centres updates every few seconds. A consequence of this is that the simulations take a great deal of time. The brokerage module requires years to be simulated and because temperature is an internally managed variable, it is not a desirable contributing factor to the experiments.

The Active Modules used in all of the Brokerage Simulations are:

1. Broker
2. Pricing
3. Events
4. Services
5. Simulation

An additional feature of CReST is that individual simulations can be modified through the use of configuration files which override the default set parameters. Configuration files were used extensively throughout the experimentation stage of the project for automation purposes, allowing many experiments to be set up and left to run in sequence.

Many different variables make up the parameter space for Brokerage simulations, including:

1. **Running Time.** The simulations each ran for a period of 276 months, the period of time for which the demand data utilised in Rogers simulation was available. Initially, experiments took around 5 minutes to run which was a severe bottleneck given the large amount of tests needed to exercise useful analysis. Fortunately, a series of optimisations and bug fixes relating to locked dependencies in the framework massively reduced this figure to around 10 seconds each.
2. **Number of User Agents.** In order to keep the profit margins at around the same levels as those seen in Rogers' experiments, the number of users was set to 1000.
3. **Prices Used.** Prices for cloud computing instances are changing all the time, due to numerous factors such as hardware prices, specialised hardware, competition and others. When recreating Roger's experiment, the same prices are used as were stated in [37]. In later experiments, prices were taken from the Amazon Web Services website. At this point in time, Amazon offer three different kinds of 3 year reservations - Light, Medium and Heavy usage. Each of these offers a tradeoff between up-front price and the hourly cost. For the experiments, the medium usage reservations were used as this is likely to find the best balance between the user's requirements.
4. **Reservation and Learning Period Length.** Amazon Web Services offer reservations of varying length - either 12 or 36 months. Rogers explored both of these options, showing that using 36 month reservations proved to produce more profit. In this study, because the main interest is verification and improvement of the model, only the 36 month instances are used. The Learning period in the simulations matches the reservation length and therefore also stands at 36.
5. **Cost Factor.** The WZH charging model is based on reservations with a cost of 1 or 2 and therefore needs to be scaled up in order to be applied to AWS pricing. In different papers this has been set at different levels e.g. 60 in [38] and 35 in [37]. Rogers claims that the optimal Cost Factor is 35 in [37], which is the most recent paper, as this both inspires truthfulness from users and results in the broker profiting. Therefore a cost factor of 35 is employed in the simulations.

6. **Demand Profiles.** The governmental data utilised in the study contains demand from many different industries and a range of these are required to be tested in order to determine the effectiveness of the broker in different markets. The same market profiles from the original experiments [38, 37] are used, as these are the most likely to apply to users wanting to exploit the advantages of Cloud Computing - Rapid Growth, Steady Growth, Recession and Recovery and Steady.
7. **Marginal Resource Utilisation Thresholds.** One of the most important parameters in the simulations, the optimal threshold could be changed by the slightest variation in any of the parameters. In the experiments that follow, a range of thresholds are utilised in order to find the most appropriate.
8. **Threshold Automation Parameters** In a later stage of the project, a system for automating the choice of the MRU threshold is proposed, adding two additional variables - μ (*Momentum*) and α (*Learning Rate*). A range of values for these are tested to find the best combination in due course.

3.3.2 Data Logging

In order to analyse the results of a simulation, appropriate logging of events and values during a run was a fundamental requirement. Fortunately, CReST supplies a logging system based on the open source Java logging library log4j [14]. This meant that all the significant intermediary data of a simulation could be captured and used for debugging, optimisation and ultimately analysis of a broker's performance under certain conditions. It was found that logging on a monthly basis made the results difficult to interpret, largely due to the offsetted reservation and execution stage of the algorithm and also due to the long running time of a simulation. Instead, the application was modified to accumulate monthly values and log yearly, trading off simplified analysis and graphing of results for more difficult debugging capability.

The simulation parameters that are typically logged are:

1. Date (Year).
2. Summed Reservation Capacity over the year.
3. Summed probabilities submitted by users over the year.
4. The number of reservations purchased during the year.
5. The actual number of reservations required by users in the execution phase.
6. The number of on demand instances purchased during the year.
7. The ongoing balance of the broker (reset every 12 months).
8. The average yearly threshold utilised by the broker.

3.3.3 Test Domain

All of the experiments conducted used the CReST framework discussed at length. In order to verify that any results obtained during testing were not anomalous, each simulation was repeated 30 times and the mean of the results was computed. When required, statistical tests were employed to test the significance of an experiment run against the others.

CReST is built using the Java programming language and as a result is cross platform. With the considerable amount of experiments that needed to be processed in order to obtain the results, the cloud was aptly employed. Using 10 identical AWS 'micro' instances loaded with a custom image containing CReST on an Amazon linux installation, the experiments were run in parallel on machines that were running no other applications that could potentially interfere. This demonstrated the power of the cloud, allowing the experiments to be processed off-site and automatically using a set of custom scripts written in Bash and Python. Once the scripts had run their course, the results were retrieved back to a personal machine - a 2011 MacBook Pro running Mac OSX in order to perform analysis.

3.4 Experimental Stage 1: Replication

In order to judge the feasibility and direction of the investigation, an important initial step was to replicate the original research performed by Rogers for the purpose of verifying the validity of the model design. The cloud computing industry is very fast paced and as such many of the variables used in the original experiments have shifted, a prime example being the pricing of instances. In later experiments this change will be exploited in order to test the flexibility of the model, demonstrate whether the results obtained by Rogers were merely a result of chance or if cloud brokers truly are a viable business venture. Furthermore, recreating the original set of experiments gave an opportunity to test the suitability and stability of the CReST framework for further development and experimentation of the model.

Although the CReST application is at an early stage of development, finding a stable version suitable for the development of the brokerage model was not too difficult. This was mainly due to its detached nature from the other major components of the application, which focus more on the mechanics and running of the simulated data centre. The main dependencies of the new module are the core framework mechanics and modules that are created or modified purely to aid the broker. In turn this means that any features that aren't fully operational or have bugs are very unlikely to interfere with the experiments. The latest stable version from subversion was employed as the foundations of the implementation. This version came with a very basic outline of a brokerage implementation originally started by Owen Rogers prior to opting for a sequential Python implementation, along with tools to interpret the demand files written by Alex Sheppard. Ultimately, the existing components needed to be scrapped and rewritten and the demand module heavily modified to meet the needs of this implementation.

Several key issues arose when initially building the broker solution within the CReST framework. As described previously, there are several ways in which the proposed model could be interpreted. This was discovered through an initial understanding and implementation along the lines of method 1. After obtaining a series of unexpected and unsatisfactory results, the second method was obtained through contact with the author. Analysis of the algorithm reveals what may be a mistake - line 22 (Algorithm 2, Page 26) involves adding a 'free' reservation upon the purchase of a reservation. However, the model suggests that the reservation being purchased would simply be added to the total capacity and would not be 'free' as suggested. This leads to much fewer reservations being purchased than one would have initially expected. While this may indeed be a 'bug', it does not necessarily invalidate the results as it simply means that the number of reservations purchased is just a function of the number of instances to hedge, most likely offsetting the time at which instances are purchased. Because this is only a minor wrinkle and doesn't invalidate the model, the algorithm was kept in the original form for experimentation. Another issue to contend with was the fact that any minor change of any of the input variables for the model altered the optimal threshold - this includes slight variations in implementation that include when the broker is charged and even minor numerical alterations such as rounding differences which are likely to occur between the two different simulation platforms.

Having seen first hand how sensitive the threshold levels can be during the implementation of the brokerage module in CReST, the optimal thresholds Rogers identified for each market were not simply taken for granted. With each experiment being repeated 30 times for the benefit of being able to perform statistical tests, the granularity of 0.01 used originally was too small for the initial foray into generating results - instead thresholds were tested at intervals of 0.1. With 11 different thresholds for each of the four markets each having 30 simulations each led to a total of 1320 test runs being required. After the initial results were examined, there appeared to be the same turning point for a number of markets at around 0.9, disagreeing with the original findings of different thresholds being optimal for different markets. To experiment whether the true optimal threshold was at this point, the granularity at the higher end of the threshold scale was increased to 0.01 and 30 additional tests were run for the thresholds between 0.8 and 0.99.

Each simulation outputs a series of logs in the csv format and dealing with such a significant number of files is impossible to do by hand - in total the sum of the files was around 30MB. In order to post-process the vast amount of data into a form that is useful for the experiments, a complex Python script

was developed. This script had the job of reading each of the logs in turn, deciphering the simulation parameters utilised and outputting other more structured data files. These data files included:

1. A csv for each market stating the total profit made in each simulation for each threshold and sorted by the most profitable threshold on average.
2. A csv for each market giving the average number of reservations owned each year in the simulations for each threshold.
3. A csv for each market giving the average profit made each year in the simulations for each threshold.

This data processing gave a much better overview of the results and a further script was developed that made use of the Python matplotlib library in order to visualise the data.

3.4.1 Results

The data obtained from computing the total profit over time for different threshold values in each market demand profile reveals key performance characteristics of the model in different environments, which align with those discovered in the original experiments by Rogers and Cliff. 95% confidence intervals were calculated for the results[1] with a t-value of 2.045 taken from a student's t-test table.[2]

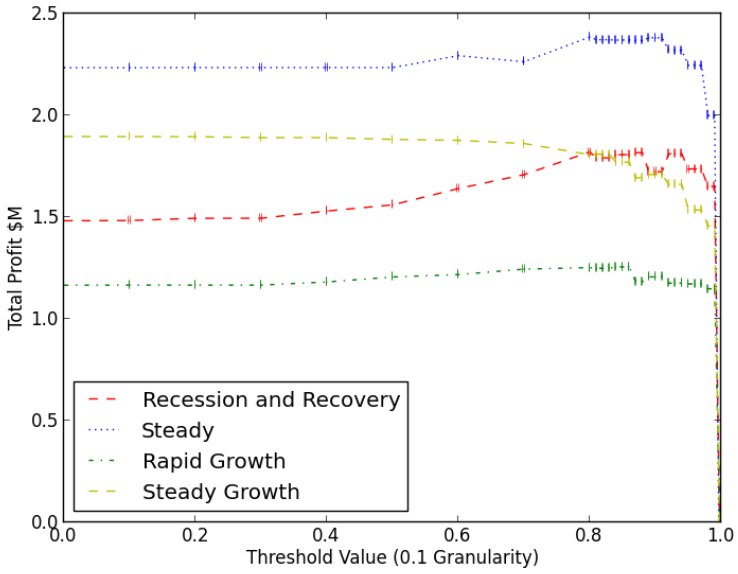

Figure 3.6: Total Mean Profit of 30 runs for each Market for different thresholds using 36 month reserved instances. The granularity between 0.0 and 0.8 is 0.1 and between 0.8 and 1.0 is 0.01. Vertical bars represent a 95% confidence interval.

Market	θ_{opt}	36 Month Reservations Profit $M		
		$\theta = 0$	$\theta = \theta_{opt}$	Change
Rapid Growth	0.84	1.17	1.26	7.7%
Steady Growth	0.00	1.89	1.89	N/A
Recession and Recovery	0.80	1.48	1.82	23.0%
Steady	0.91	2.23	2.38	7.1%

Table 3.1: Total Profits Observed in different Markets using 0 and optimal Threshold values.

[1] http://www.ehow.com/how_5933144_calculate-confidence-interval-mean.html
[2] http://www.stattools.net/tTest_Tab.php

1. **The broker is profitable in all but the non-trivial case.**
   Figure 3.6 shows that in all cases except for when θ = 1.0, the broker is profitable in aggregate. When the threshold is set to 1, the broker does not purchase reserved instances and thus will always make a loss due to charging less for on-demand instances than it is paying. Interestingly, the results exhibit that in the majority of cases a threshold nearer the top end of the scale results in higher overall profitability. The exception to this is the Steady Growth market, where the optimal threshold observed was 0.0 - although many of the thresholds resulted in very similar gains using that particular market. This can be attributed to the year-on-year similarity of the demand, which prompts the broker to purchase homogeneously regardless of the threshold value used.

2. **Considering past demand is beneficial.**
   The comparison of the total profits when using a threshold of 0, where the broker will purchase reserved instances as standard practice, and the 'optimal' threshold found revealed that the broker always benefits from using past performance. The results of the comparisons for each market can be seen in Table 3.1. Certainly this is more the case in some markets than others. For example, utilising the optimal threshold in the Recession and Recovery market culminates in a 23% increase in profitability. The other markets showed a more constrained increase when utilising the optimal threshold, on average across all markets the average increase was ≈10%.

   The vast difference in profitability in the Recession and Recovery market given the different thresholds can be explained by Figure 3.7. When using the threshold of 0, cycles begin to appear every 36 months where the broker needs to purchase a significant amount of new reservations. In between these cycles profitability is largely maintained as the broker owns enough reservations to cover the majority of demand. However, in the years where significant numbers of reservations need to be purchased, the outlay of up-front fees amounts to an overall loss. When the broker is more conservative with the higher optimal threshold, less reservations are purchased at the same time instead spreading investment over throughout the simulation, leading to a smoother profit over time and maintaining profit in the vast majority of years. This leads to a much higher profit in total, notably due to the high reduction in costs.

   The effect of choosing 'better' thresholds can be further exemplified by Figure 3.8. With θ_{opt}, the number of resources owned by the broker tracks much closer to the actual demand, therefore minimising the costs incurred by either owning too many reservations such that they are not used and owning too little, so extra on-demand are required. On the other hand, with θ set to 0, far more reservations are purchased than are required, increasing the costs significantly, leading to unused reservations and ultimately causing the losses observed.

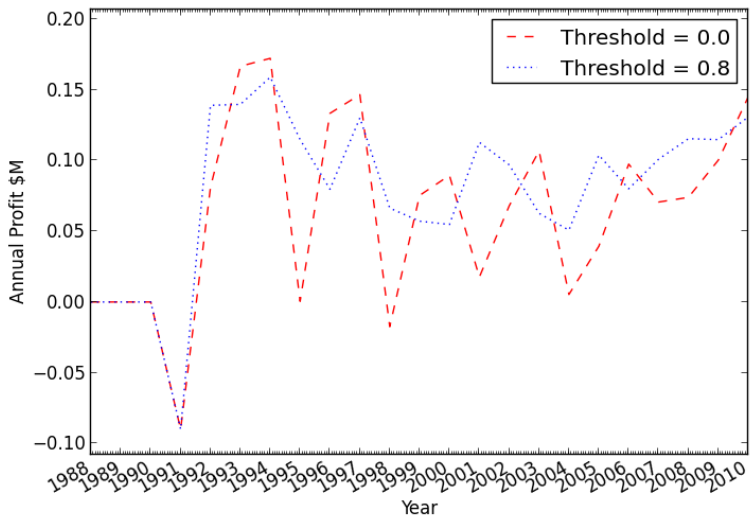

Figure 3.7: Annual Profit for Broker in Recession and Recovery Market at $\theta = 0$ and $\theta = \theta_{opt}$

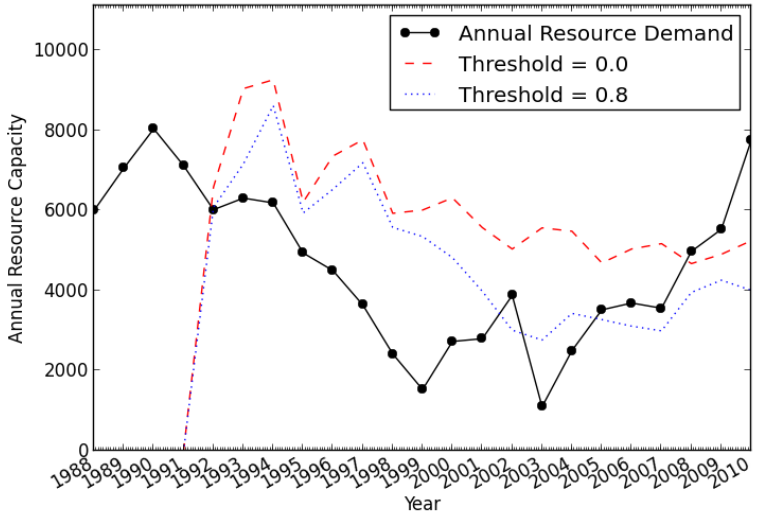

Figure 3.8: Annual Resources Owned by Broker in Recession and Recovery Market at $\theta = 0$ and $\theta = \theta_{opt}$

3.4.2 Statistical Hypothesis Testing

In order to confirm that changing the marginal resource usage threshold of the broker directly influences its profitability, statistical hypothesis testing in the form of the t-test was performed on the results of the simulations. This was achieved through performing paired t-tests between the samples where reservations are always purchased ($\theta = 0$) and sample sets where historical usage is taken into account ($0 < \theta < 1$). The paired t-test assumes that the observed data are from the same subject and are drawn from a population with a normal distribution, but does not assume that the variance of both populations are equal.

The goal of the testing is to determine that the two populations have means that are not equal. Due to the fact that only samples of the populations are to hand, a probability (p-value) is computed and compared to an arbitrary cut-off level. The null hypothesis of the testing is that the means of the samples are equal ($\mu_1 - \mu_2 = 0$) and the alternative hypothesis is that the means are not equal. If the calculated p-value is below the threshold, the null hypothesis can be rejected and the results are considered to be statistically significant.

In order to perform the large amount of t-tests required between each pair of thresholds for each market, a script was developed in Python using the scipy statistics library to automate the process. Running the tests on the total profit samples gave the following results:

θ	Markets							
	Rapid Growth		Steady Growth		Recession & Recovery		Steady	
	t	p	t	p	t	p	t	p
0.1	1.33	0.19	-0.68	0.49	1.47	0.15	-0.31	0.76
0.2	1.35	0.19	-1.82	0.079	23.5	$2.04e^{-20}$	0.87	0.39
0.3	1.01	0.32	-16.33	$3.6e^{-16}$	18.40	$1.54e^{-17}$	0.46	0.65
0.4	50.7	$6.9e^{-30}$	-15.59	1.06^{-15}	105.32	$5.03e^{-39}$	1.14	0.26
θ_{opt}	245.8	$1.1e^{-49}$	N/A	N/A	603.25	$5.45e^{-61}$	294.6	$5.78e^{-52}$

Table 3.2: An excerpt of T-Test T and P-Value Results for Each Market Comparing Profitability when $\theta = 0$ and $\theta > 0$. The data shows that the mean profits become statistically significant after a certain threshold change, but this varies between markets. For brevity, only a small number of the tested thresholds are shown here, up to the point where the majority become significant ($p < 0.05$).

The outcome of running the t-tests makes for interesting reading, providing concrete evidence to backup the assumptions made when analysing the graphical representations of the data. The significance level is set at 0.05. Several key observations can be made:

1. The t-values outputted by the process show that thresholds using past experience ($0 < \theta < 1$) do not always outperform purchasing reservations to cover demand ($\theta = 0$). When the t-value is negative, it indicates that a zero threshold has outperformed a non-zero threshold. Trivially, this is always the case when the threshold is set to 1, in which case reservations are never purchased and as a consequence the broker makes a loss. In the Steady market there is an example that unexpectedly shows that always buying resources is more beneficial than using past experience. However, the corresponding p-values in this case is found to be not significant (> 0.05), suggesting that the results overall are similar, but in the sample results theta = 0 led to a marginally higher profit. A key market that follows this pattern with significant results is the Steady Growth market, where 0.0 is found to be the optimal threshold. This is likely to be because reservations purchased in this market are more likely to be utilised in the future than in the other examples, leading to a threshold on the lower end of the scale to be optimal as this results in more reservations being purchased.
2. The results make it clear that selecting thresholds that take into account past experience in the vast majority of cases result in higher profits. Trivially, the p-values corresponding to this become increasingly significant as θ approaches the previously identified θ_{opt}. Across the different markets, it is plain to see that altering the threshold impacts the profitability in different ways, which is caused by the demand curve in each case, impacting the usage of the reservations purchased by the broker.
3. In the vast majority of cases, it is found that the mean profit of selecting $\theta > 0$ differs from the mean profit of selecting $\theta = 0$ with a high rate of significance in all tested markets. This result allows for the null hypothesis to be rejected and the alternative hypothesis, which concludes that the means are different, to be accepted. This is a promising and interesting finding, as it may allow for methods to be developed that can adapt the threshold to differing market conditions in order to maximise profitability.

3.4.3 Discussion

The optimal thresholds found for the majority of the markets simulated tend to be at the higher end of the scale, suggesting that protecting the broker from reservations purchased that are under utilised is less risky than having to potentially purchase a significant number of reserved instances in the execution phase.

The results obtained through the experiments on the whole emulated those found in the original paper [37]. As previously discussed, any small alterations in the simulations can cause the optimal threshold values and in turn the profitability to be different. Implementation differences such as when the provider charges the broker, rounding differences and handling of demand profiles could all have impacts on the output of the simulation. On the whole, however, both simulations agree that the model is a profitable endeavour and the CReST implementation goes some way to verifying Roger's results. Graphs of resource usage and year-on-year profits for the other market profiles from the CReST simulations can be seen in Appendix A.

3.5 Experimental Stage 2: The Effect of Pricing

The primary objective of experiment 1 was to verify the findings of the original experiments, in turn allowing for the implementation in the CReST framework to be tested. In order to emulate the original experiments, the same set of assumptions and parameters were used. However, since the time that Rogers performed the initial foray into the idea, the cloud computing market has moved once more. Prices have invariably altered as the industry has matured, with economic factors and market growth encouraging competitors to enter the promising market. Further choice for the consumer in the type of instances and even reservations have also become available in the time since, expanding the scope of the model. One of the aims for this project is to try to emulate the real world as closely as possible and in order to achieve this some of the assumptions and parameters made in the experiment 1 simulations were altered in order to investigate whether the model remains robust under changing conditions.

1. Prices were brought in line with current Amazon Web Services prices for a small instance in Virginia (March 2013). The options presented by Amazon currently range from light, medium and heavy usage reservation tiers, each combining different prices for up-front and hourly prices. The 'medium usage' reservations were chosen for this particular experiment as they appear to offer the most flexibility between customers needing different usage whilst renting the instances.

   The Prices used were:

   - Monthly On Demand = \$46.80.
   - Up Front Reserved = \$250.0.
   - Monthly Reserved = \$13.68.

2. The way the broker is charged is altered slightly. The broker now pays the normal up-front fees for reservations and the monthly charges only for the instances used, to bring it in line with the assumption that if there are surplus reservations, the virtual machines would not be utilised and therefore no charge incurred with the provider. This differs from the approach taken by Rogers, in which the broker was only charged for on demand instances in months when they were required, ignoring the reserved instances also utilised. This was a suspected bug that is rectified in this experiment henceforth.

3.5.1 Results

Market	θ_{opt}	36 Month Reservations Profit \$M		
		$\theta = 0$	$\theta = \theta_{opt}$	Change
Rapid Growth	0.4	1.50	1.51	0.67%
Steady Growth	0.1	1.93	1.93	0%
Recession and Recovery	0.6	2.19	2.21	0.91%
Steady	0.0	2.52	2.52	0%

Table 3.3: Total Profits Observed in different Markets using 0 and optimal Threshold values.

The outcome of the experiments reveals patterns that could be expected from reducing the costs associated with the model. For example:

1. Trivially, the reduction in costs whilst maintaining income results in increased profits throughout the combinations of market profiles and thresholds.

2. The sensitivity of θ_{opt} to changes in price is revealed. Each of the markets show a different optimal threshold to those in Experiment 1. This can be attributed to the fact that a lower reservation price reduces the risk involved with investment in the long term asset. In turn, hedging more reservations leads to increased profitability, which is cohesive with a lower threshold value.

3. The decrease in prices appears to have brought the profitability of the different threshold values to a very similar level in comparison to those seen in the first experiment. This can be seen in the percentage change between $\theta = 0$ and $\theta = \theta_{opt}$ (see Table 3.3), where the largest difference observed was less than 1%. Table 3.4 does show that in a lot of cases a threshold of 0.0 was not the worst performing, but the profitability between the different thresholds remains much smaller than with the higher prices. The large percentage differences observed previously were due to the fact that at some thresholds in some years, losses were made due to the high up-front price of the reservations purchased. Figure 3.10 shows that profitability is much more consistent throughout the simulations with lower prices, explaining the closeness of the profits for different thresholds.

Market	Profit ($MM) at Different θ									
	0.0	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9
Rapid Growth	1.503	1.502	1.503	1.502	1.501	1.505	1.499	1.479	1.408	1.346
Steady Growth	1.929	1.929	1.928	1.927	1.927	1.919	1.916	1.904	1.861	1.799
Recession Recovery	2.190	2.190	2.193	2.193	2.189	2.202	2.205	2.188	2.205	2.108
Steady	2.525	2.525	2.524	2.525	2.524	2.525	2.508	2.499	2.514	2.421

Table 3.4: Profitability in Different Markets using θ values at granularity of 0.1.

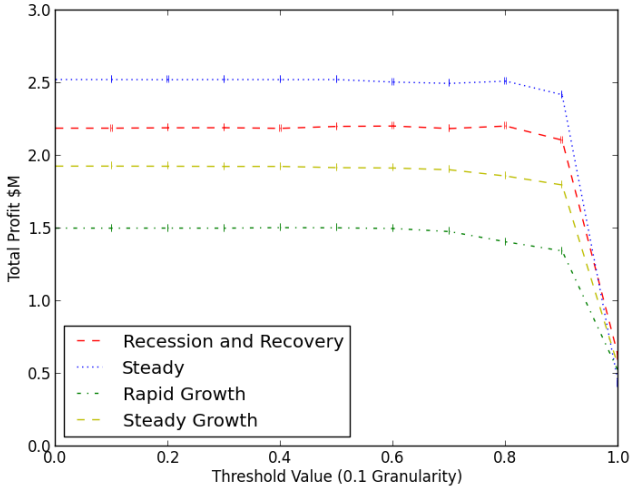

Figure 3.9: Total Profit for each Market for different thresholds using 36 month reserved instances.

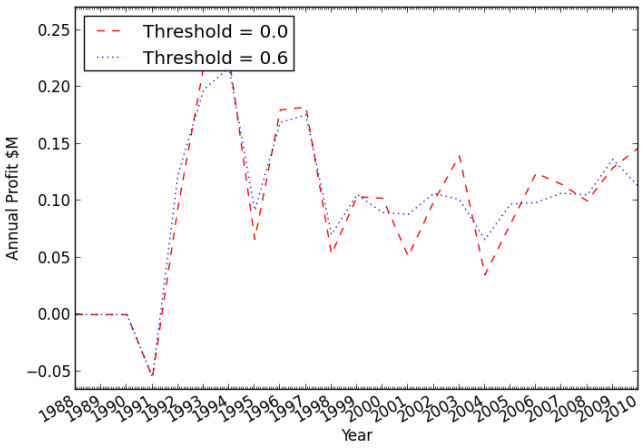

Figure 3.10: Annual Broker Profit in Recession and Recovery Market at $\theta = 0$ and $\theta = \theta_{opt}$

3.5.2 Discussion

As would be expected, the experiment shows that the reduction of the broker's costs in the form of the prices paid for reservation and on-demand instances, whilst maintaining the amount of income from users results in higher profits in all cases. The results show that the profitability gap between using different thresholds is vastly reduced and thresholds towards the lower end of the scale, which result in the hedging of more reservations, create more profit.

However, it is important to see these results in context and speculate on the real world implications. There are two points to consider.

1. One of the most important rules of the WZH Model is that users must prefer to use it over going to the provider directly. While in this particular example the cost of purchasing an instance option from the broker is still cheaper than going directly to the provider, if charges always remained constant it may not be beneficial to users to employ the broker. For this reason, passing on the savings of cheaper instances from the provider to the users would be a sensible option - it would enable a balance to be struck between appeasing the users and maintaining profitability. In the simulations, the charge to the user is calculated using the equations supplied by Wu *et al* [49] multiplied by a *Cost Factor* variable introduced by Rogers [38] in order to bring the prices in line with the monthly costs. Therefore, it would be pertinent to have some kind of link between the prices and the Cost Factor in order to maintain a balance between broker profitability and encouraging users to use the service. It is likely that a decrease in the Cost Factor could result in a larger influence from the thresholds once more.
2. In this particular set of experiments, the optimal thresholds for each market are lower than those observed before. However, it may still be advantageous for a broker to select a higher threshold, especially given the small difference in profits made. A higher threshold is less risky - it results in less reservations being purchased and as a consequence has less outstanding risk if the market suddenly changes. Clearly there are a lot of factors that affect the threshold chosen, but in general it is once more likely that some sort of compromise needs to be made between profitability and overall risk.

3.6 Experimental Stage 3: Parameter Space Exploration

Throughout the project it has become increasingly clear that the WZH Model's performance greatly depends on the parameters in use. Utilising different combinations of variables can cause the optimal utilisation threshold to change, directly influencing the amount of reservations purchased and in turn, the profitability of the broker. Clearly it is in the interest of the broker to optimise the intrinsic parameter settings, allowing cheaper prices to be offered to potential customers and therefore operating a successful business. Finding a pareto optimal solution is the ultimate goal - a group of settings that results in the highest profitability given a certain set of constraints. This set of experiments attempts to further investigate the sensitivity of the model to changes in the parameter space, thus allowing the analysis of the model in a wider context; hopefully providing the identification of opportunities to improve performance and robustness.

3.6.1 The Influence of Users in Simulations

An important component of the simulations of which the implications have not yet been fully considered is the number of users partaking. Thus far, a pool of 1000 customers submit for instances whereas in real life the number could indeed be many times that. A further consideration is that in the paradigm of cloud computing, customers tend to scale sideways, purchasing multiple instances that run in parallel. Although at the moment the CReST simulation does not support single users submitting for multiple instances, this can be emulated by simply adding additional customers to the pool.

One of the observations made after altering the prices to emulate Amazon was that the difference between utilising the optimal threshold and simply setting the threshold to always purchase reserved instances was now very small, around a 1% improvement. It is important to remember that although this is only a small improvement in comparison to those found with the old prices, when thousands of reservations are being leased to customers then this can equate to a significant amount of profit. In order to test this theory and witness the influence of the number of users on results, an experiment was set up employing the same variables as those in experiment 2, with the notable exception of utilising a pool of 10,000 users rather than the original 1000.

Results

Market	θ_{worst}	θ_{opt}	Profit $M With 10,000 User Pool			
			$\theta = 0$	$\theta = \theta_{worst}$	$\theta = \theta_{opt}$	Change (θ_{worst} to θ_{opt})
Rapid Growth	0.9	0.4	15.02	13.45	15.06	11.97%
Steady Growth	0.9	0.1	19.27	17.99	19.28	7.17%
Recession and Recovery	0.9	0.6	21.9	21.78	22.07	1.33%
Steady	0.9	0.0	25.23	24.2	25.24	4.3%

Table 3.5: Total Profits Observed in different Markets using 0 and optimal Threshold values, when using user pools of 1000 and 10,000.

Table 3.5 shows that altering the number of users directly manipulates the profit made by the broker and indeed with multiplying the number of users by 10, a 10-fold increase in profits is observed at the end of the simulations. This data highlights the importance of choosing a sensible threshold, as although the difference between selecting $\theta = 0$ and $\theta = \theta_{opt}$ is similar to that seen when testing only 1000 users, the gap in profits is accentuated greatly by the variable increase. Furthermore, a factor that was not considered in experiment 2 was that a threshold choice of 0 performed well in the majority of markets (Recession and Recovery being the notable exception) and in order to show the contrast in profit that can be made through choosing the wrong threshold, θ_{worst} is also shown in this table (the least-well performing theta value below 1). Indeed, the results show that a risk-averse broker who selects higher threshold values in order to minimise the reservations held could miss out on up to $1million profit in this particular scenario, which is clearly a very significant amount of money in a business scenario, showing that careful consideration needs to be places on the threshold selection.

3.6.2 Balancing Broker Profitability with Value for Consumers

One of the key benefits of the WZH Model is that it allows the broker to charge a smaller fee for accessing compute instances than it would typically cost a consumer to rent one on-demand directly from the provider. Indeed, this helps to maintain *Condition B* - that the users must prefer to use the brokers services over the provider. Because the product is identical, the only differentiator is price and therefore the broker has to offer the service at a cheaper rate. In the prior experiments, the price charged to the users - the *Cost Factor* - has been kept constant at 35, despite the fact that the second experiment utilised updated Amazon Web Service pricing, which has decreased in the time period since Owen Rogers' first simulations. Whilst this trivially increases the broker's profits, it also decreases the ratio between the price the user would have to pay. The aim of the broker is to keep the users truthful and one way to achieve this is to adjust the charges to a lower, but still profitable, level.

In order to resolve the effect on profitability and whether alterations in efficient thresholds occur, an experiment framework was constructed that tested the model under the typical 0.1 threshold granularity and a number of different Cost Factor settings ranging from 20 to 40 in steps of 5. The experimentation should allow an insight into the direct effect of the Cost Factor to the outputs of the simulation and allow a suitable updated broker charge to be selected to pair with the change to the provider prices.

Results

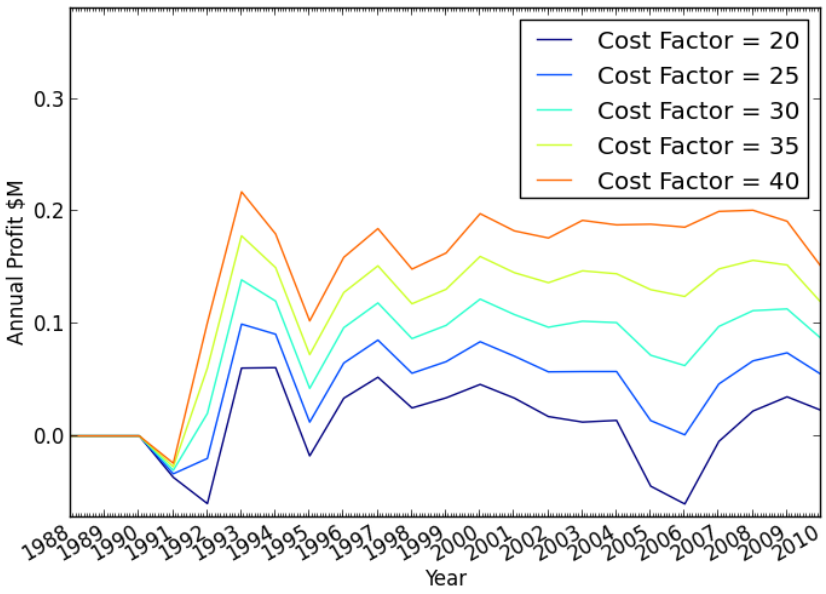

Figure 3.11: Annual Broker Profit in Steady Market at $\theta = \theta_{opt}$ for differing Cost Factors.

The results show first and foremost that altering the Cost Factor directly shifts the profit curve up and down. This can be seen in Figure 3.11 (and the other graphs, in Appendix A), where the shapes of the profit graphs remain homogenous. Since employing the more recent AWS prices, the annual profits for each of the markets has been shown to be much more consistent. When using a sufficient Cost Factor, this culminates in a steady yield year on year, essentially the main aim of the business potential that underpins the idea.

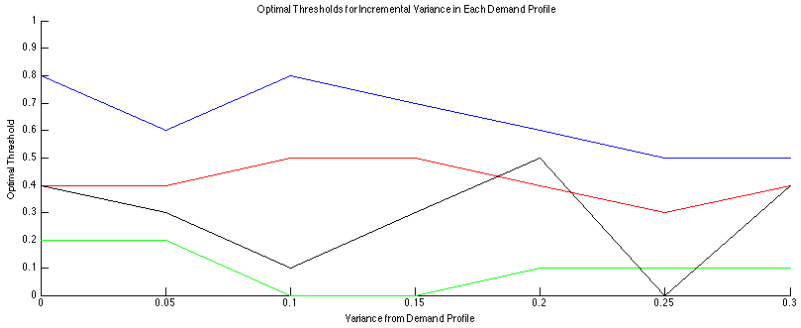

Figure 3.12: Optimal Thresholds of the Different Demand Profiles using Incremented Cost Factor

Once more, the requirement of a careful selection of threshold value becomes prevalent, with the adjustment of the Cost Factor variable causing the modification of the optimal threshold in each of the markets except Steady Growth.

The main question still remains: what Cost Factor should be chosen? This is not an exact science and some guesswork is required at this stage as it is not possible to gauge the opinions of the consumers. In a real life situation, the broker would be able to test the market by charging as much as they possibly could whilst retaining the user's custom. The experiments here have shown that a Cost Factor of as low as 25 can show consistent income year on year, but setting a lower price consequently means that less money is made than might be possible and a shock in the market may more easily cause a loss to be made.

For the purposes of this project, it seems sensible to keep the results in the same kind of profit range as Roger's original experiments. Therefore, the ratio of broker charge and provider charge in the original research was found and applied to the prices used in these examples - this results in a Cost Factor of just under 30 and therefore for the remainder of the project, this is the value that will be utilised.

3.6.3 The Effect of Variance in the Demand Profile

In all the experiments up until now, a varied set of real-world demand profiles have been leveraged for the use of testing the effectiveness of the broker over a large number of years. One of the main thrusts of the project is to test the model in a light that reflects the real world. While the data truly reflects a number of markets in the real world, one thing that needs to be taken into account is that the same results may not necessarily recur. The model has so far been shown to work given certain examples of markets, but what happens when the profiles vary slightly? The question here asks whether the model is robust to small changes in the expected demand.

In order to investigate, the parameters were set to the same values as used in Experiment 2 and an additional variable, variance, was added that alters the current normalised demand value by a random amount between $demand - variance$ and $demand + variance$. The advantage to testing against small variance in demand is that it allows changes in the optimal threshold to be observed, giving some indication of its robustness. Once more, 30 test runs were made for each experiment set up, which consisted of the four original demand profiles at threshold intervals of 0.1 and incremental changes in variance of 0.05, between 0 and 0.3.

Results

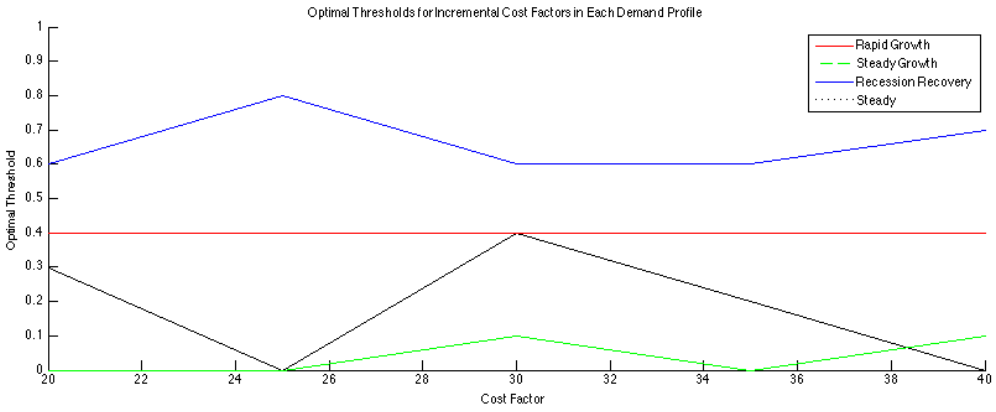

Figure 3.13: Optimal Thresholds of the Different Demand Profiles using Incremental Variance

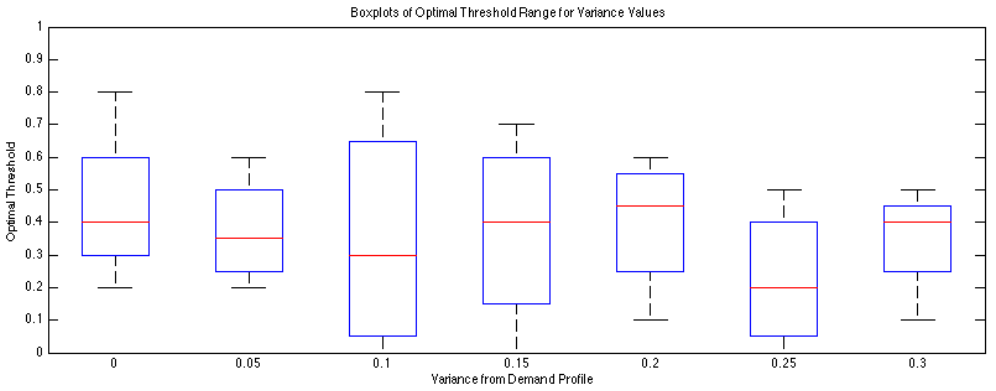

Figure 3.14: Boxplot of the Range of Optimal Thresholds in Different Markets under Differing Variance Factors

Market	θ	Avg Profit ($MM) at Different Variance Levels							% Diff Between
		0.0	0.05	0.1	0.15	0.2	0.25	0.3	Min and Max Profit
Rapid Growth	0.4	1.088	1.086	1.086	1.084	1.083	1.079	1.075	1.2%
Steady Growth	0.2	1.377	1.377	1.375	1.372	1.368	1.362	1.355	1.6%
Recession Recovery	0.8	1.600	1.571	1.590	1.556	1.551	1.545	1.534	4.3%
Steady	0.4	1.764	1.765	1.763	1.758	1.753	1.744	1.735	1.7%

Table 3.6: Average Profitability in Different Markets at different variance using θ_{opt}

Market	Avg Profit ($MM) at Different Variance Levels using θ_{opt}							% Diff Between
	0.0	0.05	0.1	0.15	0.2	0.25	0.3	Min and Max Profit
Rapid Growth	1.088	1.086	1.086	1.085	1.083	1.079	1.075	1.2%
Steady Growth	1.377	1.377	1.376	1.375	1.371	1.364	1.357	1.4%
Recession Recovery	1.600	1.594	1.590	1.589	1.575	1.585	1.575	1.6%
Steady	1.764	1.766	1.763	1.759	1.753	1.744	1.735	1.8%

Table 3.7: Average Profitability in Different Markets at different variance using θ_{opt}

This set of simulations continue to show the extent of the sensitivity of the brokers utilisation threshold, once again showing shifts when the market is slightly varied. Tables 3.6 (showing the results using the optimal threshold found when $variance = 0$) and 3.7 (results when using θ_{opt} for each particular variance level) show that employing the optimal threshold found for that particular variance level does indeed have an effect, with the Recession Recovery market showing the greatest divergence in profitability when variance in applied. This larger change is smoothed out by picking a 'better' threshold for that *variance* level.

Of course further questions can be asked here. For example, how to know what the incoming demand is going to be in order to select the suitable threshold. Furthermore, it is likely that the broker takes orders from a number of users in different markets and that also needs to be a factor in the threshold selection considerations. Figure 3.14 shows the range of optimal thresholds seen across different markets at each particular variance value. It is clear from these plots that selecting an appropriate threshold is a difficult task and the best case scenario may well to select an average over different markets. This in turn could be weighted by the number of users in each particular market.

3.6.4 Discussion

It can be said for certain that the result of this experiment is that the selection of an appropriate threshold level for the broker is no straightforward task. It has been shown that a number of external influences, including the number of customers and variations in demand, have a strong, unpredictable and non-linear influence on the ideal value, directly corresponding with the overall success of the broker. Moreover, even the alteration of parameters inherently intrinsic to the broker entity sway θ_{opt}, such as the Cost Factor - the multiplier to the prices paid by each of the users.

The sensitivity noted in across the tested markets is compounded by the fact that in a real world environment, the broker is likely to take orders from customers operating in a multitude of industries. This further hinders the broker in selecting a threshold value, making it difficult to predict the incoming demand. For this reason it seems sensible to abstract the threshold selection away from the demand of certain markets and focus on the information that the broker has learnt. Automating the process based on concrete evidence of market behaviour and the broker's current position could hold the key to solid performance in the long term, whilst balancing the number of reservations held against the likely demand in the future. The effectiveness of such a scheme will be investigated in the next stage of the project.

Chapter 4

Project Execution: Extending the Model

4.1 Autonomous Adaptive Thresholding

The evident sensitivity of the threshold parameter and its intrinsic contribution to the overall performance of the model presents a complication for the application in real world scenarios. Selecting the prime θ value enables the broker to balance its asset exposure to the providers in a favourable manner, ultimately reducing risk and maximising profits. The WZH Model leverages the data of past events in order to hedge risk appropriately. However, due to the nature of its operating environment it is not known *a priori* if the market will continue to follow the same pattern. Up to this point, the experiments conducted have been based on real world past data - however, the inherent unpredictability and vicissitudes of the world's markets could render forecasts made on previous demand meaningless. A market shock where demand for a resource in the community suddenly alters, perhaps caused by a new entrant to a market, could lead to the broker operating with a suboptimal threshold parameter, leaving it risk exposed in the number of reservations currently owned. Doubtlessly therefore it would be advantageous for θ to be automatically updated to reflect the current market circumstances during operation. Here, a versatile technique is presented that enables the broker to autonomously revise θ on the fly.

4.1.1 The Delta Rule

The Autonomous Adaptive Thresholding mechanism (AAT) is an extension to the model proposed by Rogers which utilises the Widrow-Hoff delta rule [46] in order to streamline the threshold selection between iterations of the hedging process. The approach is general and has been shown to be effective in several different domains such as Algorithmic Trading [10] and coevolutionary optimisation frameworks [7]. The delta rule is one of the simplest rules in Machine Learning, forming the basis of both adaptation algorithms [39] and reinforcement in classifier systems [47, 48]. The fundamental strategy of the method involves minimising the error between a real system output and a target output determined by some proxy appropriate to the problem. With the projected reservation utilisation as a proxy, AAT updates the θ value in each reservation stage of the model through the minimisation of the error between the current threshold and the determined target. If there is no difference between the system output and the desired output when using the delta rule, then no learning takes place. On the other hand, when there is a difference, the values in the system are altered to reduce the difference. The system can be explained with the following set of equations (the notation used is borrowed from [7], which in turn followed from [10]).

Let A_t be the actual output at time t and A_{t+1} be the actual output on the following time step.

$$A_{t+1} = A_t + \Delta_t \tag{4.1}$$

where

$$\Delta_t = \alpha(T_t - A_t) \tag{4.2}$$

Δ_t is the product of a learning rate (α) and the difference between the actual output at t (A_t) and the target output (T_t).

If the target value remains constant, A_t will converge to T_t at the rate determined by α. However, a non-fixed target can cause A_t to oscillate around the target value. In order to dampen the oscillations, an additional variable known as the *momentum* term (μ) can be introduced, transforming the equation to:

$$\Delta_t = \mu\Delta_{t-1} + \alpha(1 - \mu)(T_t - A_t) \tag{4.3}$$

4.1.2 Deriving the AAT Equations

The delta rules expressed above form the basis of the update rule for the MRU threshold. However, as with [7], the target threshold required at each time step is actually unknown and therefore needs to be derived from the data at the broker's disposal. An additional associated variable in the form of a normalised version of the projected resource utilisation rate is used, denoted τ. Remembering that a lower θ (close to 0) encourages the purchasing of reservations, while a higher θ (close to 1) encourages purchasing less reservations, τ can be determined (*NB. 1 is added to denominator for cases of no demand*):

$$\tau = \frac{reservationsOwned}{summedDemand + 1} \tag{4.4}$$

The ratio of reservations owned to the number of instances demanded appears to be the only viable proxy for determining a sensible movement for the threshold value. The logic behind the approach lies with the ultimate aim of the broker to maximise profit through the constant full utilisation of the reservations owned, in which case the more expensive on-demand instances would not be purchased and reservations would not go unused. The choice is not without its complications, however. Foremost amongst these are the fact that if the broker owns a relatively large number of reservations, say 100, and the demand for reservations is low, for example 10, the target becomes $\frac{100}{10+1} \approx 9.1$. This is clearly not suitable as a target threshold as it exceeds the maximum value of θ considerably. The proposed solution for this involves normalising the outputted value (see Equation (4.5)) by keeping track of the largest recorded raw target and normalising the values between 0 and 1. In this particular example, if 9.1 was the largest seen so far, it would be normalised to 1. If a raw target of 10 had been seen in a previous month, it would be normalised to 0.9 and so on. Further complications, for example a one-off huge target forcing subsequent months to all obtain indistinguishable normalised values, are investigated in the successive sections through a series of experiments in order to unearth the optimal operating conditions for the proposed augmentation to the model.

$$\tau = \frac{\tau - minTarget}{maxTarget - minTarget} \tag{4.5}$$

where *minTarget* and *maxTarget* are updated over time to determine the relative value of τ.

Then, letting θ_t and θ_{t+1} be the threshold at time t and $t+1$ respectively and substituting in τ as the target value, the AAT equations are obtained from equations (4.1) and (4.3).

$$\theta_{t+1} = \theta_t + \Delta_t \tag{4.6}$$

where

$$\Delta_t = \mu\Delta_{t-1} + \alpha(1 - \mu)(\tau - \theta_t) \tag{4.7}$$

and $\Delta_0 = 0$. The three parameter settings must all fall between a certain range: $0 \leq \tau, \alpha, \mu \leq 1$.

4.1.3 Selecting Robust AAT Parameters

The obvious downside to introducing the idea of automated thresholding to the model is the addition of yet more variables that provide a non-linear contribution to the overall performance. Although it may appear to just be adding unnecessary complications to the model, adapting theta on the fly yields several advantages for the broker. Firstly, it can be considered an autonomous entity, adapting as demand and other market factors vary over time in order to maintain profit margins. It also means that the model can be used immediately, with little to no prior knowledge of the market it will be operating in.

In order to determine a solid, core set of settings to use henceforth, a set of experiments were devised that tested the performance of different combinations of the learning rate and momentum parameters under different market conditions. The values tested ranged between 0 and 1 for both α and μ values in increments of 0.05. Prior investigations into similar automation implementations suggests a lower momentum term encourages a non-fixed θ (μ = 1 implies fixed θ) [7], which is theoretically an undesirable feature for this implementation. A slower learning rate is also likely to be a pertinent choice, preventing jumps to lower thresholds when the demand is very high, but might not stay there - leading to an over-purchase of reservations. If the α value is lower, it is likely to lead to a much smoother transformation of the threshold and is therefore more likely to keep in line with the flow of market demand. Despite these preconceptions, experiments were carried out for the full range of μ and α values, in order to ensure that every possibility was covered.

Rank	Markets							
	Rapid Growth		Steady Growth		Recession & Recovery		Steady	
	μ	α	μ	α	μ	α	μ	α
1	0.7	0.65	1.0	0.2	0.7	0.05	0.9	0.95
2	0.3	0.75	0.55	0.8	0.4	0.8	0.4	0.3
3	0.5	0.85	0.3	0.65	0.5	0.7	0.0	0.4

Table 4.1: Top 3 Ranking μ and α combinations for each market profile.

The ranks of the top 3 performing combinations in each market can be seen in Table 4.1, which can be interpreted to convey the idea that there is not one single blend that outperforms the alternatives, falling in with the consistent theme seen throughout the project that shows that the parameters have varying effects on the market profiles. In order to find the best combination across markets, a script was composed that ranked the μ and α combinations for each profile and then found the average rank. The outcomes of the average rank can be seen below:

μ	α	Avg. Rank (440 max)
0.7	0.05	400
0.8	0.85	393.75
0.1	0.8	387.25
0.45	0.8	377.5
0.4	0.3	371.25

Table 4.2: Highest Performing μ and α combinations across all markets, maximum possible rank of 440 (which would imply that the combination was the highest performing in all markets)

The underlying data from which the ranks were obtained showed that for the majority of the markets, choosing the optimal parameters did not yield a noteworthy increase in performance. Despite this observation, employing the data combined with the knowledge that no choice would be perfect in all scenarios, suggested that a selection of 0.7 for the μ parameter and 0.05 for the learning rate value was a sensible choice for the forthcoming experiments across markets.

4.2 Experimental Stage 4: Investigating AAT Behaviour

The first step in determining whether the AAT technique adds benefit to the model involved testing the renewed behaviour of the agent in the familiar market scenarios and drawing comparisons against the results obtained when utilising a static threshold variable. Several key questions stand out:

1. Does the threshold value converge over time, and if so, does it converge to θ_{opt}?
2. Does the starting threshold value affect the progress of the adaptation in any way?
3. Is the model still profitable and can it compete with simply setting θ_{opt} as a static threshold?
4. Is a situation created where a large *maxTarget* forces the threshold permanently down and can this be rectified if so?

In an bid to answer these questions a further two sets of simulations were prepared, fundamentally based on the same parameters found to be optimal in the prior experiments and using the AAT variables identified in the previous section. The second set of tests involved the addition of a yearly reset for the *maxTarget* value, whilst the initial set maintained the parameter throughout. For each demand profile, 3 starting thresholds were trialled - 0, 1 and the previously identified θ_{opt}. The routine 30 simulations were conducted in order to avoid any anomalous results and in order to enable visualisation in a similar manner to profitability, the logging operations within the CReST implementation was adjusted to output the mean yearly threshold.

4.2.1 Results

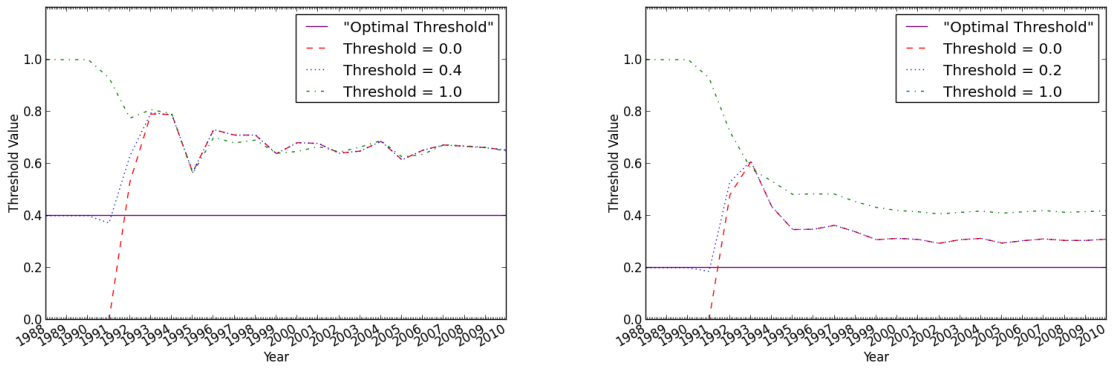

Figure 4.1: Yearly Mean Threshold in Rapid and Steady Growth Markets

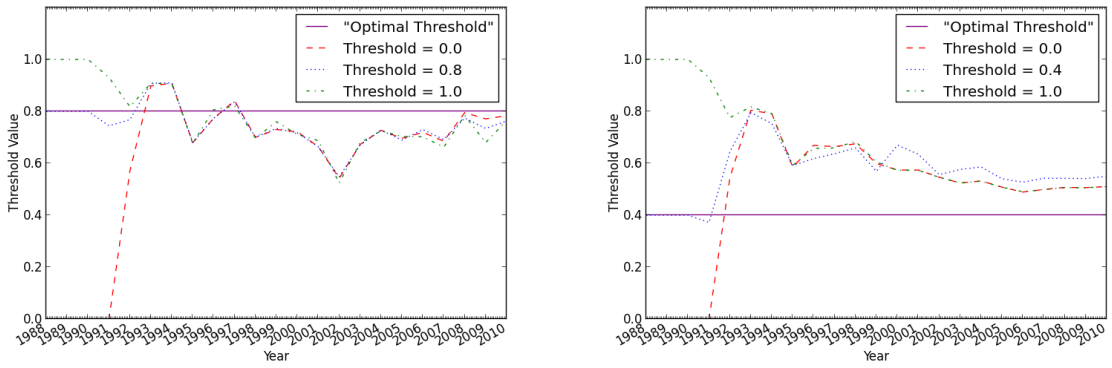

Figure 4.2: Yearly Mean Threshold in Recession & Recovery and Steady Markets

From the first inspection the mechanism does indeed appear to function in the sought-after manner. As can be seen in Figures 4.1 and 4.2, the threshold value converges to a similar value irrespective of the originally set threshold. It is important to remember that these plots only include the average values obtained and the underlying data reveals that particularly in the earlier stages of the experiments, the threshold varies considerably. This is revealed by the small dips that can be seen in all of the markets with starting thresholds above 0 which occurs around 1990. This dip is caused by AAT rapidly reducing the threshold due to the broker owning no reservations, initially to 0 and then gradually increasing as the reservation supply is balanced against the incoming demand. Furthermore, the slight differences seen between the movements when different starting thresholds are used can be explained by variances in the result sets. However, the difference noted in the Steady Growth market draws away from this argument. What is more likely is that the small number of reservations made when utilising the original threshold affects the model in the long term; after all, the number of reservations hedged each month has a dependency on the number of reservations currently owned. The combination of the number of reservations purchased and the overall demand shape clearly has a long term impact on the bearings of the adaptivity, which just so happens to have more of an impact in the Steady Growth market than its counterparts. This argument is also backed by the results of experiments where the *maxTarget* variable is reset every year - as can be seen in Figures 4.3 and 4.4. The fact that the convergence occurs for all thresholds in the Steady Growth example suggests that there are months in the simulations where a particularly large *maxTarget* is created, which is accentuated by lower starting thresholds; this is logical, as a lower start threshold would encourage heavier reservation purchasing at the start, in turn increasing *maxTarget*. Because the *maxTarget* variable has a higher value for those instances, the normalisation of the target threshold is reduced, which is why it converges to a lower value.

Another interesting point lies with the fact that for all markets with the exception of Recession Recovery, the adapted threshold converges to a value above the pre-calculated 'optimal' threshold. The overall objective of the adaptive thresholding is to achieve a level that balances supply against demand, achieving 100% utilisation of owned reservations, and this may not necessarily result in the optimal value. The fact that the threshold lies above suggests that the mechanism favours purchasing less reservations, resulting in the buying of more on-demand instances but simultaneously reducing the chance of purchased instances going unused. This is unlikely to be negative, as while profits may be reduced slightly, the overall risk of asset ownership is decreased which is a bearable trade-off.

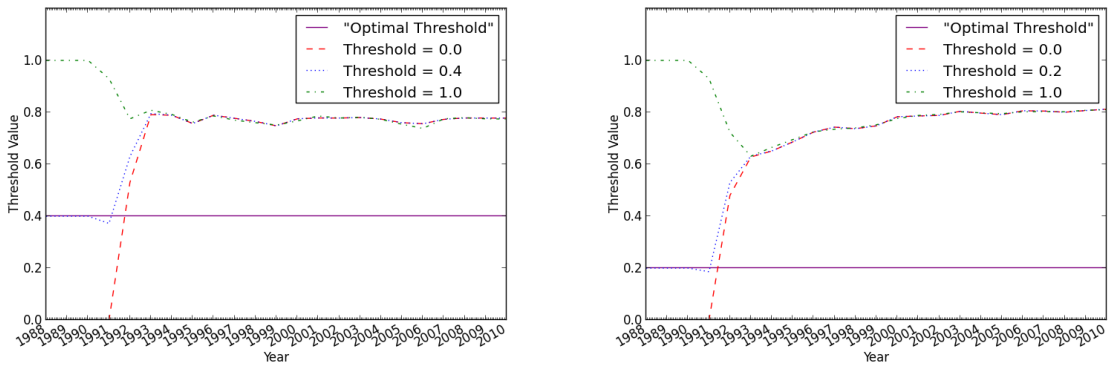

Figure 4.3: Yearly Mean Threshold in Rapid and Steady Growth Markets with *maxTarget* Resets

Market	Mean Profit ($MM) Using Different Configurations						
	Static θ	AAT			AAT (with Reset)		
	$\theta = \theta_{opt}$	$\theta = 0.0$	$\theta = 1.0$	$\theta = \theta_{opt}$	$\theta = 0.0$	$\theta = 1.0$	$\theta = \theta_{opt}$
Rapid Growth	1.088	1.0765	1.0789	1.0765	1.072	1.065	1.072
Steady Growth	1.377	1.367	1.362	1.367	1.367	1.362	1.368
Recession Recovery	1.600	1.610	1.594	1.614	1.606	1.605	1.595
Steady	1.764	1.739	1.735	1.783	1.739	1.736	1.782

Table 4.3: Profitability in Different Markets using AAT Mechanism with and without Resetting

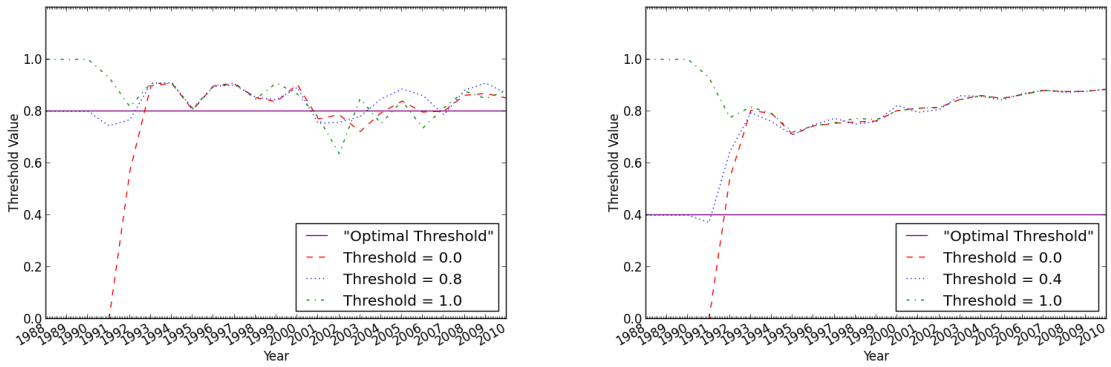

Figure 4.4: Yearly Mean Threshold in Recession & Recovery and Steady Markets with *maxTarget* Resets

Indeed, the profit data retrieved from the experiments in Table 4.3 shows that for the majority of markets, the broker makes less money by employing the AAT mechanism over setting the previously determined θ_{opt}. This comes at little surprise for markets that follow a certain pattern - for both the growth markets a constant lower threshold is going to be beneficial, favouring the purchase of reservations that are certain to be used at a later date. The more cautious thresholds seen through the application of AAT with and without reset will lead to fewer reservation being purchased and in turn reducing profitability. On the other hand, a more volatile market reveals the advantages of moving the threshold on the fly. In both implementations of AAT, 2 out of 3 of the configurations tested resulted in better performance over application of a static threshold. Potentially the most surprising result was that in a Steady market, both AAT and Reset AAT when configured to start from the static threshold outperform it over the course of the simulation. Certainly in that particular market, the expectation may be that a rigid threshold would yield higher profitability. Unfortunately, the results remained inconclusive regarding the ideal value to start the adaptation at, with the ideal varying between markets.

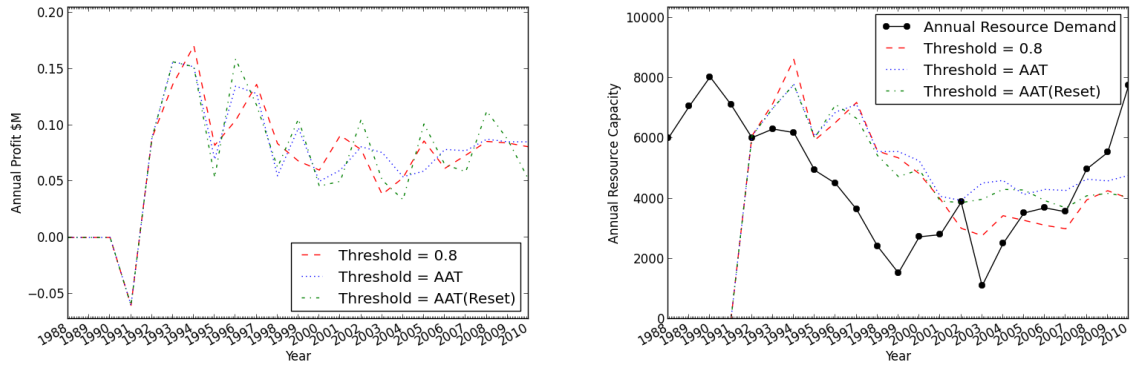

Figure 4.5: Yearly Profit and Resources for θ_{opt}, AAT and AAT (Reset) for Recession Recovery

Certainly in the Recession and Recovery example as can be seen in Figure 4.5, the profitability remains smoother when utilising the AAT mechanism without resetting, although there are a few examples where resetting *maxTarget* appears to yield an improvement. However, these improvements are very slight, and when taking the data as a whole into consideration it seems that resetting hinders overall performance.

4.2.2 Discussion

Initial experimentation with the system shows that it does behave in a desirable manner; converging to a suitable threshold value in all markets after a relatively short time. The convergence value appears to favour the purchase of less reserved instances than the optimal static threshold, but it is entirely possible that this may not always be the case for all parameter combinations; some refining through techniques such as genetic algorithms could allow the AAT method to outperform static methods in all cases. The experiments performed here have shown the technique to be favourable in markets of higher volatility and as discussed previously, it is shown to be weaker in growth markets where owning more reserved instances is likely to amount to higher profit margins due to continued usage month-on-month.

Initialisation of AAT is still a contested issue. Across markets the ideal start value varies widely and therefore a suitable tradeoff may be to start it at a mid-range value, for example 0.5. It needs to be kept in mind that this technique was developed as a solution to the fact that in scenarios in real life, the broker may have to deal with multiple markets simultaneously and they may well not follow the patterns shown here. Disruptions in certain industries may rapidly change the demand for instances, causing static thresholds to hedge unfavourably. Adapting the threshold automatically helps to minimise the risk associated with this and further experimentation will aim to resolve whether the technique succeeds in achieving the goal of effectively overcoming market shocks.

4.3 Experimental Stage 5: The Effect of Market Shocks

The final stage of this investigation looks at the performance of the broker under dynamic market conditions, or market shocks. Up to this point the assumption has been that the broker will witness a certain set of market conditions, taken from real world data. However, in a real world scenario this is less likely to be true as the broker will undoubtedly take orders from clients across different sectors which in all likelihood will follow independent demand flows. In an ideal world, the demand flows would remain as those seen, allowing the broker to set a threshold that could maximise profit. A number of external demand or supply side factors could cause this to unfortunately not be the case, causing the profiles to be more dynamic in nature. Therefore it is important to analyse the behaviour of the broker if the market moves.

This investigation simulates the broker acting on the same demand profiles as used in the prior experiments, only henceforth market shocks will occur. In this context, a market shock will involve the demand profile dynamically switching to an alternative within a simulation. This is achieved in CReST through a *MarketShockEvent*, which forces the user agents to forget their history and relearn the newly allocated profile in order to begin submitting requirements once more.

It is this kind of situation that AAT is expected to provide the most significant impact in performance for the broker, theoretically allowing the threshold to move to accommodate the change in market conditions. Of course the model is slightly more complex than a simple linear transformation and the jumps in demand between the different profiles utilised varies significantly, which in turn leads to a set of interesting results.

In order to ensure that the adaptive mechanism was not underperforming, a new set of experiments to determine appropriate μ and α values was performed for the market shock scenarios. Although not as clear cut as found in the previously experimented linear markets, the outcome from these experiments found that values of 0.55 for the momentum parameter and 0.45 for alpha were the most effective. Therefore, these are the values used in the following experiments.

4.3.1 Results

Original Market	Shock Market	Mean Cumulative Profit $(MM)		T-Test Results	
		Static $\theta = \theta_{opt}$	AAT	t	p
Rapid Growth	Steady Growth	1.395	1.386	23.93	$1.2e^{-20}$
Rapid Growth	Recession Recovery	1.174	1.168	16.6	$2.4e^{-16}$
Rapid Growth	Steady	1.369	1.362	17	$1.27e^{-16}$
Steady Growth	Rapid Growth	1.123	1.115	20.79	$5.73e^{-19}$
Steady Growth	Recession Recovery	1.185	1.179	20.2	$1.25e^{-18}$
Steady Growth	Steady	1.355	1.347	24.2	$8.72e^{-21}$
Recession Recovery	Rapid Growth	1.536	1.573	-18.36	$1.65e^{-17}$
Recession Recovery	Steady Growth	1.821	1.830	-11.59	$2.08e^{-12}$
Recession Recovery	Steady	1.817	1.913	-45.46	$1.65e^{-28}$
Steady	Rapid Growth	1.425	1.438	-38.34	$1.7e^{-26}$
Steady	Steady Growth	1.713	1.733	-50.63	$7.5e^{-30}$
Steady	Recession Recovery	1.511	1.527	-30.22	$1.79e^{-23}$

Table 4.4: Mean Profits $(MM) and T-Test results for Market Shock Environments with Static and AAT Thresholding

Several key trends emerged in the experiments, the most notable of which is that when presented with an initial growth market, the broker amassed more profit when utilising the static optimal threshold for that market, regardless of the market move after the shock. Conversely, when presented initially with a Recession or Steady profile, the adaptive algorithm showed to perform considerably better than its static counterpart, again regardless of the market move. These results are all shown to be statistically significant when using $p < 0.05$, although the trend does appear to show that the advantages of utilising AAT in the Recession Recovery and Steady markets outweigh the disadvantages of using it in the growth markets. In other words, the rift in the results is shown to be greater when AAT outperforms static thresholding.

These trends, as before, can be attributed to the fact that the growth markets tested in these scenarios favour lower threshold values which encourages more reservations to be purchased, which are guaranteed to be used as it's simulating growth, in turn producing higher profits. When employing AAT, after the initial purchase the threshold value tends to rise as the priority is to target a value which will balance the number of reservations owned against the incoming demand. Because the mechanism doesnt know that the demand is constantly rising, it does not react to keep the threshold low. Where a market is not constantly rising on the other hand, AAT supplies a significant advantage as a better balance is struck between the number of reservations hedged at any one time.

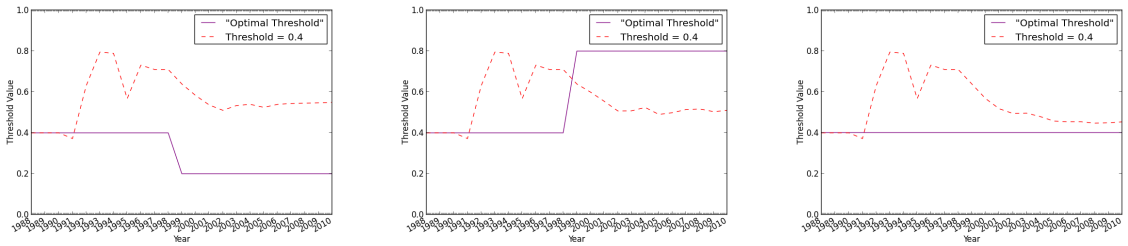

Figure 4.6: Threshold Values over time for a starting Rapid Growth Market moving to Steady Growth, Recession and Recovery and Steady profiles respectively.

As can be seen in Figure 4.6, after the initial reservation purchasing phase, the market shock does not tend to have a significant effect on the threshold value. There are subtle differences, but the threshold value does not appear to tend towards the perceived 'optimal' thresholds of the new market. This could be due to several reasons:

1. Because only half of the profile is now in use, θ_{opt} may no longer be accurate for that portion of the profile as the values calculated were the best on average in simulations of the whole profile.

2. At the stage that the market shocks occur, the broker already owns a significant inventory of reservations and given that there is the same number of user agents in the market, the number required is unlikely to vary too wildly. Even if there was a large variance, this would only cause the threshold to be lowered for a short time and afterwards raised again in order to attempt to average 100% utilisation. This causes the average yearly threshold value to be around the mid-range, which is portrayed by the graphs.

4.3.2 Detection of Growth Markets and Bullish Instance Reserving

One of the clearly established issues with the AAT approach is that it does not factor in whether incoming demand is in a growth cycle. In a market where demand is constantly increasing, profit can be maximised through setting a lower threshold value, as the broker can be sure that the extra reservations purchased through this approach will be used in the future. The next set of experiments aims to rectify this weakness in the model through the implementation of a growth detection mechanism.

The assumption is that if the broker can identify that it is in a growth cycle, the targeted threshold value computed can be adjusted to accommodate this observation, hedging a higher number of reserved instances and theoretically returning a higher profit margin over the long term. In the best case scenario, this detection would enable the preservation of the effectiveness of the AAT mechanism in mercurial markets, whilst adjusting the threshold to a more aggressive stance when suitable opportunities are presented.

In order to test this theory, a definition of a growth market had to be defined in order for the broker to test the incoming demand. As the idea of what could be construed as a growth market is fairly arbitrary, this extension currently provides just a proof of concept and thus presents an opportunity for further analysis and optimisation in further research opportunities. In this brokerage implementation, the test for growth was conducted prior to adapting the threshold for incoming demand and as such was conducted on a rolling basis. Two tests were used that both needed to pass in order to be judged that a bullish growth market is taking place:

1. In the past 12 months, at least 6 months resulted in a higher demand than the previous month.
2. The demand of the current month is higher than in the same month a year previous.

If both of these constraints were satisfied, the broker is moved to a more aggressive stance where the target threshold is halved. Some tweaking was performed to determine the difference made of different multipliers used under an aggressive outlook; but results found that it made little to no difference, positive or otherwise.

		Original Market			
		Rapid Growth	Steady Growth	Recession Recovery	Steady
Shock Market	Rapid Growth	-	1.121	1.569	1.407
	Steady Growth	1.386	-	1.819	1.700
	Recession Recovery	1.168	1.184	-	1.494
	Steady	1.362	1.354	1.927	-

Table 4.5: Mean Profits $(MM) for Market Shock Environments with AAT Thresholding and Growth Detection

The results provide yet further interesting behaviour. Some key observations are:

1. Adding the aggressiveness extension did not bring any benefits when the starting market is Rapid Growth, but the results were no worse than utilising plain AAT.
2. The profits made when starting in a Steady Growth markets are significantly increased and are almost on par with those achieved when using the θ_{opt}, regardless of the shock market in the simulation.
3. When starting in a Recession Recovery market, the results are still better than utilising a static threshold and only marginally less successful than using plain AAT in two of the simulations. When moving to the Steady market, aggressiveness was shown to increase performance.

4. When using the aggressive tactics, the profits are lower when starting in the Steady market than using either a static threshold or plain AAT.

The underperformance when starting in a Steady market can be attributed to the very definition of the market profile. If it's steady, then its likely that it will go up and down equally in the 12 month periods and is therefore likely to satisfy the initial growth constraint. Being aggressive in such a market is unlikely to yield a higher margin as it is likely the reservations will go unused and therefore leads to the inevitable disappointing results that were found. When starting in the Rapid Growth market, slightly better results were expected. However, on further inspection, in the first half of the profile it is steadier, with growth not really occurring till after the market shock period and this explains the similarity in performance to plain AAT.

What has become clear is that incremental improvements can easily be made to the algorithm in order to specialise for certain types of markets. However, it appears that for every improvement made, a trade-off in performance under other conditions is necessary. What's important to remember is that when employing all of these techniques, the broker has shown to be profitable; the small tweaks are just employed to squeeze every penny possible out of the operation.

4.3.3 Discussion

The experimentation with market shocks has revealed the importance of the early stages of reservation hedging for the broker's overall performance in addition to the strengths and weaknesses manifested when utilising the proposed autonomous thresholding algorithm.

With the reservations being a long-term investment and the broker not being able to see into the future, there is little that can be done in the short term to circumvent a situation where the broker suddenly owns significantly more or less reserved instances than required. After all, the threshold only controls the number of months that the broker has an instance deficit, which is calculated using previous months demand data. When a market shock occurs, this disrupts the technique as the previous demand data is no longer relevant. Theoretically, this is where the automated thresholding should benefit the broker as it is able to tweak the number of reservations purchased depending on the incoming demand, even if it is uncharacteristic given past events.

The underperformance of AAT in growth markets has been discussed previously, but a compelling observation from the dynamic market experiments shows that when the growth markets are the shock markets, AAT still outperforms a static threshold. The opposite is also true, where markets that AAT has been shown to surpass the static thresholding method prove the opposite as a shock market. This can be explained by the fact that it takes at least three years to adapt to the new market demand values and the previously owned reserved instances to run their course and in the meantime it is highly likely that profit is not maximised. An extension to the AAT algorithm that makes purchasing of reserved instances more aggressive in growth scenarios has been shown to increase performance in such situations. However, employing this technique at all times has the tradeoff whereby a more steady market profile could over purchase reservations, harming it's profitability in the long run.

There are undeniable merits to the addition of AAT to the Extended WZH pricing model, even in situations where it is not shown to outperform the optimal static threshold, it enables the broker to obtain consistent profits year on year. Future work could potentially improve the method further in order for it to more robustly adjust to growth markets, cementing it as a key component to the profitability of the model. It can be argued that regardless of the slightly poorer results obtained in such markets, the algorithm should still be employed, given that in the real world the broker would have no idea what the optimal threshold would be, forcing educated guesswork. The slightly higher resulting thresholds from AAT reduce the assets owned by the broker, in turn reducing risk if market shocks do occur.

Chapter 5

Conclusion

5.1 Main Contributions and Achievements

This project set out to investigate a niche opportunity in the fresh and exciting field of Cloud Computing, incorporating a blend of research, implementation, analysis and refinement. The WZH Model [49] introduces a coordinator into the Cloud Computing market, providing benefits for the resource provider and users whilst retaining a profit for itself. Rogers and Cliff investigated the idea further [37], taking the original mathematical theory and showing not only that it could potentially be applied to real world scenarios, but adding improvements that help to mitigate the risk associated with purchasing long term assets. The replication of that particular research in the open-source data centre simulation framework, CReST, served as the starting point for a series of detailed experiments, tweaks and improvements explored throughout this thesis. Robustness analysis was performed through a range of experiments intended to both re-create Roger's original scenarios and further probe into the impacts of altering details in the parameter space. Through scrutinisation of the original work and the results obtained in the early stages of this investigation, weaknesses of the model were identified and an autonomic extension based fundamentally on the Widrow-Hoff machine learning algorithm [46] was proposed. This involved original research into the effect of dynamic markets on the broker's ability to sustain profitability.

In summary, the ensuing findings were made during the course of the investigation.

1. The WZH Model was found to consistently profit given both the instance prices used in the original experiments in [37] and using up-to-date variants from Amazon Web Services, reinforcing the claims made by Rogers in the original paper. These findings were made in spite of adjusting some of the original assumptions made in the paper, which did not appear to emulate how such a scheme would work in the real world.

2. Statistical analysis, in the form of t-tests, was performed on the results of the simulations. In Rogers' work, each threshold value was only tested once. The findings in this project however indicate that the profitability can vary between simulations, casting doubts on the optimal thresholds originally found due to experimental anomalies not being accounted for. The results of the t-tests did serve to confirm the original observations that an alteration of the threshold parameter θ directly affects the profitability of the broker and consequently it is desirable to keep the variable at an optimal level.

3. An in-depth sensitivity analysis revealed the extent to which both intrinsic and extrinsic factors can affect the fruitful operation of the broker, demonstrating that a change in price, variations in market demand and adjusting the price multiplier can severely alter the potential profitability. This can be mitigated through careful selection of the MRU threshold introduced to the model by Rogers, which was also found to be extremely sensitive to the influence of market conditions. This justified the fundamental argument that the entity should have the ability to automatically adjust internal settings in order to adapt to environmental changes.

4. As a further step in improving the model, the Autonomous Adaptive Thresholding strategy was proposed as an answer to the sensitivity of the model to environmental changes. Inspired by implementations in both Algorithmic Trading [10] and Evolutionary Computing [7], the Widrow-Hoff method [46] was adjusted to allow the broker to adapt the threshold value dynamically in an attempt to balance the broker's supply of reserved instances against the incoming demand from users. This method was found to outperform using even the optimal static threshold in both steady markets and erratically changing markets, but fell short when presented with profiles that are constantly growing - a situation in which the broker would perform better by hedging more reservations than the risk-averse nature of the adaptive algorithm would allow.
5. Pioneering work into the performance in dynamic markets revealed the importance of the initial purchasing stages for the broker, due to the long ownership period of the reserved instance as an asset. Experimentation found that regardless of the market movements, the broker remained non-trivially profitable throughout when employing both static and adaptive thresholds, although the initial market determined which method resulted in producing a higher profitability.

All things considered, the broker was found to invariably profit when utilising the WZH Model in all market scenarios presented when using a threshold below 1, i.e. it actually invests in reserved instances. This could be seen to serve as a demonstration that a commercial implementation is feasible. The idea is not without its gambles, however. The cloud computing space is rapidly maturing and there are a number of external factors that could result in the purchase of long term reservations being seen as risky. Furthermore, contract restrictions on reselling could be a barrier to successful real-world execution.

5.1.1 Contributions to CReST

With CReST still in a young stage of development, this is the first research project to have employed it for use as an experimental platform. A considerable amount of time was spent learning the design of the system in order enable successful integration of the Brokerage module into the framework and great care was taken to ensure that no bugs were introduced to other components. Fortunately, the components required for the experimentation in this project were in a stable state when checking out the latest version of the framework, enabling development to be relatively painless. Issues did arise when attempting to simulate the inherently procedural nature of the model into the Object-Oriented, event driven system that is CReST, but a lot of perseverance and refactoring led to confidence in the final implementation. The following developments were made as a direct result of this project:

1. Around 2500 lines of Java code. Comprising of:
   (a) A Pricing module to simulate requests to the provider for prices.
   (b) A Brokerage module, simulating the broker and user agents.
   (c) Alterations to the Builder GUI and configuration parsers to allow for additional parameters to be specified in the experiments.
2. A suite of automation scripts written in Python totalling around 1000 lines of code, with the purpose of automatically parsing and analysing the output data from experiments. This was found to be one of the initial issues with the framework when running a significant amount of experiments - it is not possible to deal with over 30,000 csv files manually.

All the code written for this project has been contributed and as a result is open-source. It is sincerely hoped that further work can be conducted on what is an exciting area, with a great variety of possibilities for further investigation.

5.2 Current Project Status

Here is a summary of the initial aims and objectives for the project and the respective status of each:

1. *Replicate and verify the results of Rogers' original experiment through implementation in the CReST framework.* This was completed and the results echo those found by Rogers in his original work. However, interpreting the model for implementation was found to be challenging; two approaches were taken and these are covered in Section 3.2.1 on the interpretation of the broker.

2. *Discover whether brokerage is a profitable venture given the current pricing strategies of cloud computing providers.* It was clear right from the initial implementation that employing the WZH Model in the correct manner pretty much guarantees profit given the prices in the current market. When considering past demand, the hedging of the reservations results in a balance between reservations owned by the broker and the number required by the users, in turn mitigating the associated risk of selling on assets. This was a fairly abstract aim, but the results of the simulations suggest that it is indeed a profitable venture. Evidence can be seen throughout Chapters 3 and 4.

3. *Perform a sensitivity analysis, determining the scale of the effect of extrinsic environmental alterations on the performance of the broker.* Evidence for this analysis can be found in Chapter 3, where a number of intrinsic and extrinsic parameters were tested in order to determine their contribution to overall performance. This led to the conclusion that selecting the right θ value was extremely important and in turn led to the development of AAT in an attempt to link the threshold to full reserved instance utilisation.

4. *Improve the broker model through automated threshold optimisation, which should enable profit maximisation given any demand profile.* Autonomous Adaptive Thresholding was developed as a response to the sensitivity analysis performed in Chapter 3 and is covered thoroughly in Chaper 4. The results were not quite as originally expected, although it was shown to outperform even the optimal static threshold in both the Recession Recovery and Steady market profiles. In growth markets it still tended towards a balanced θ value on average and so resulted in lower profits than a static alternative that favoured purchasing increased amounts of reserved instances. Regardless, it was still shown to be profitable in such markets and higher so than many higher static threshold alternatives.

5. *Thoroughly test both static and adaptive implementations of the broker over different demand profiles and analyse the behaviour when the market suddenly changes, i.e. a shock occurs.* Market Shocks were built into the CReST framework, allowing the market profile to change to an alternative at any time during a simulation. Thorough testing of both the static and adaptive thresholding methods was carried out in Chapter 4 and the results showed that once an inventory of reserved instances had been built up to accommodate a certain profile, it took a little while for the broker to adjust. However, profitability was once more maintained.

5.3 Recommended Extensions and Unexplored Options

With the concept covered in the project being so fresh and with relatively little research coverage thus far, a multitude of options presented themselves throughout the execution phase that could have taken the investigation in different directions. Some of these ideas are shared here in the hope of inspiring further work in the area and the continued development of the CReST toolkit.

5.3.1 American Options

Also advocated by Rogers in the original paper [37], American Options are an alternative derivative to the European Options seen throughout this work where the holder has the right to use a resource at any time up until the expiry of a contract. An example of this could be if a user knows that they will require a reservation for a full month at some time during the year, but not the particular month it needs it. Through the purchase of an American Option from a broker, the customer could acquire an instance at a cheaper cost than a typical on-demand instance whenever it is needed. In terms of the broker hedging the appropriate number of reservations, there is a $\frac{1}{12}$ chance of the holder will require an instance in a given month which is added to the forecast computation. This kind of Option would

suit industries where demand over large periods of time is predictable, but peak demand times are unknown.

5.3.2 Further Investigation into the Parameter Space

One of the most frustrating issues within the project involved the huge amount of time taken to run the vast amount of experiments. Part of the reason for this was testing small tweaks in the parameter space in order to optimise the behaviour of the broker as much as possible. Ideally, the broker should behave in an optimal way across all markets whilst using the same parameters, but finding favourable combinations is a monumental task that simply cannot be completed manually. With adding yet more variables into the equation with the AAT technique, it would be desirable to find an automated method that could determine the best combination of intrinsic broker parameters moving forward. Due to the non-linear nature of the contributions made by each of the parameters, this could be a candidate for Evolutionary Computing techniques such as a Genetic Algorithm.

5.3.3 Analysing Performance in a Multi-Broker, Multi-Data Centre Environment

Thus far, the simulations of the model have taken a rather simplistic view, making the assumptions that there is only one compute resource provider and one broker in the universe. Of course in the real world, this is unlikely to be the case. With the move towards standards in the cloud computing space, theoretically one day provider lock-in should be nullified, allowing consumers to run their applications ubiquitously between providers and as such letting them choose the cheapest deal at the time. This would then lead to a free flowing price market, where demand could dynamically alter the price of the instances offered by the different providers. Simulating this dynamic environment in CReST is likely to turn up some interesting results. Furthermore, tracking the behaviour of a market with multiple brokers could also yield further insight into the likely feasibility of such a model in the real world.

Appendix A

Additional Result Graphs

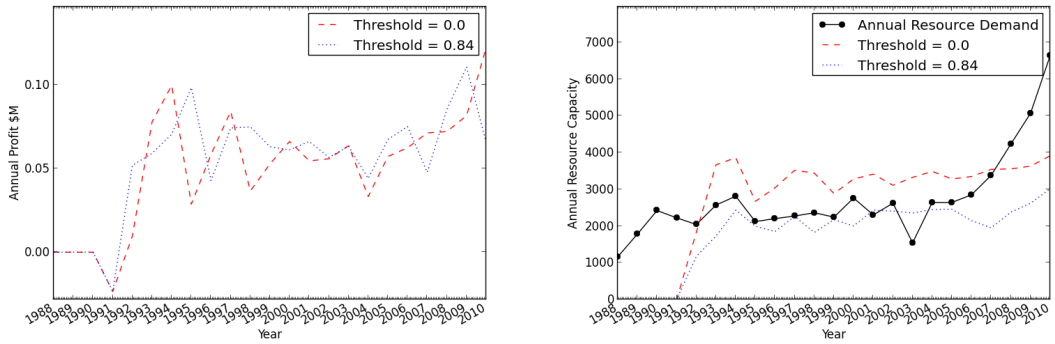

Figure A.1: Rapid Growth Profits and Resource Utilisation

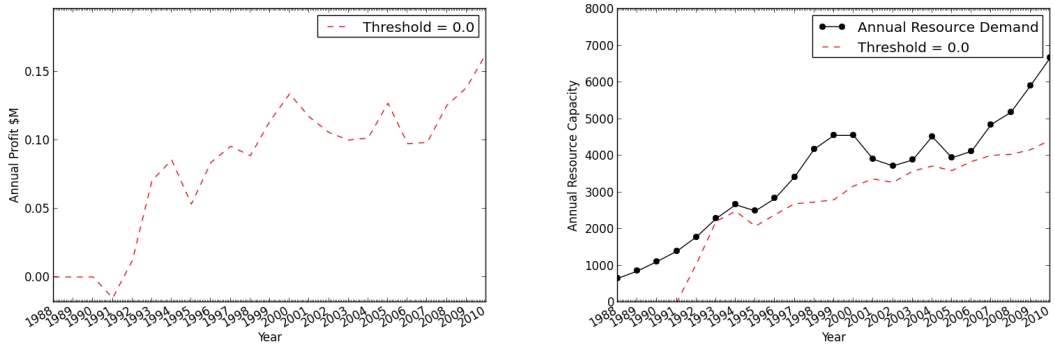

Figure A.2: Steady Growths Profits and Resource Utilisation

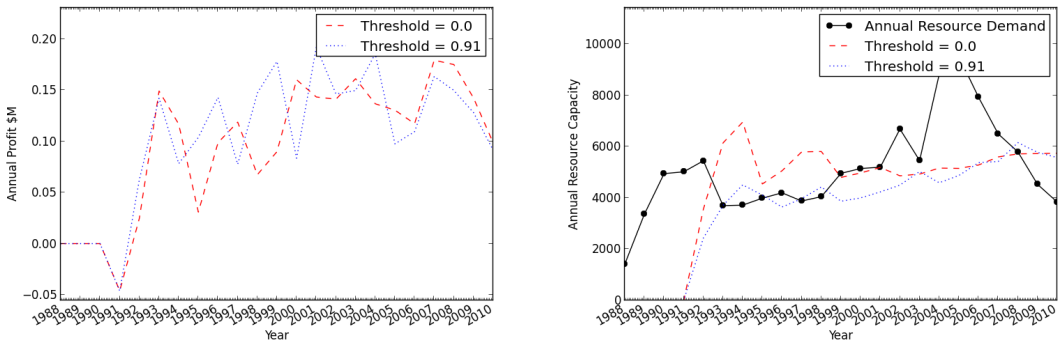

Figure A.3: Steady Profits and Resource Utilisation

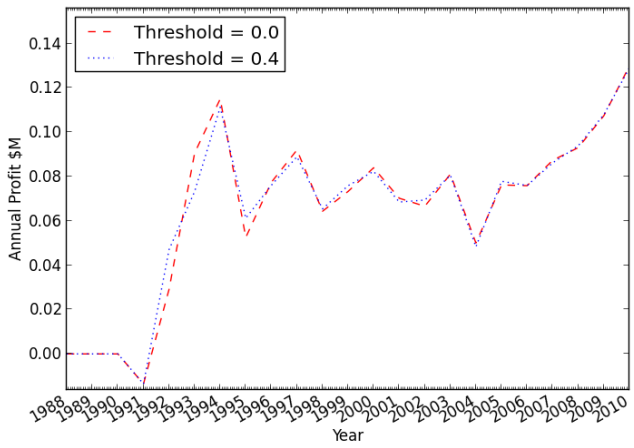

Figure A.4: Rapid Growth Profits with New Prices

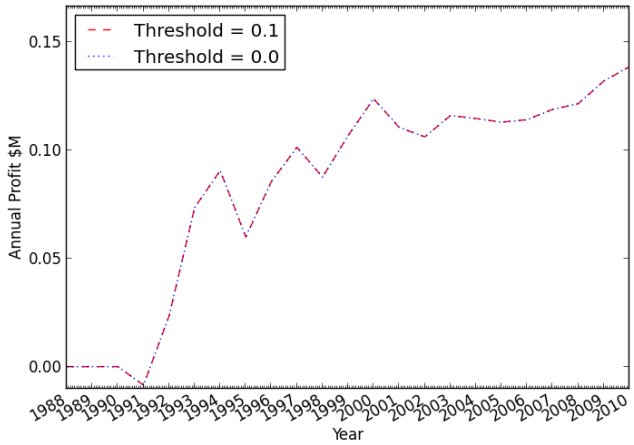

Figure A.5: Steady Growth Profits with New Prices

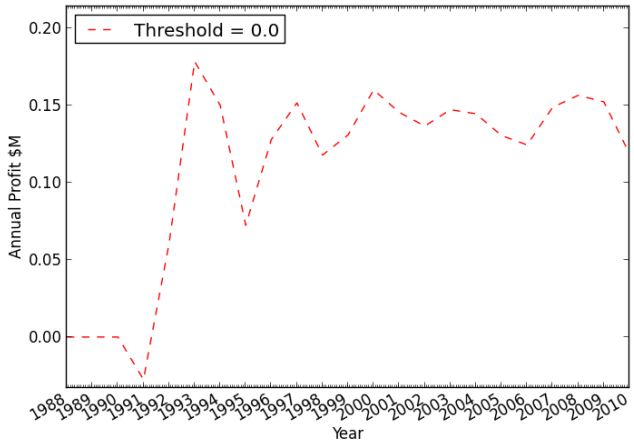

Figure A.6: Steady Profits with New Prices

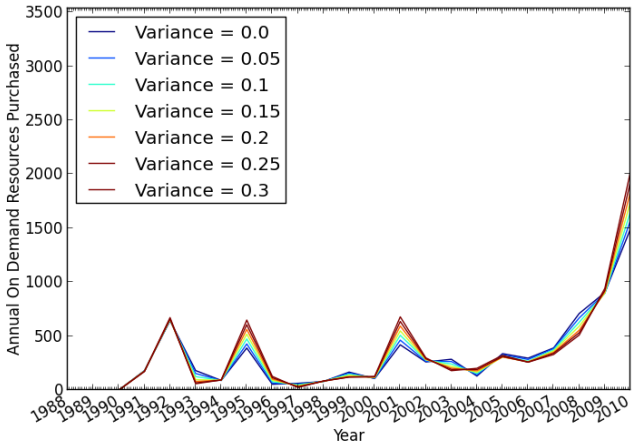

Figure A.7: On-Demand instances purchased under different levels of variance.

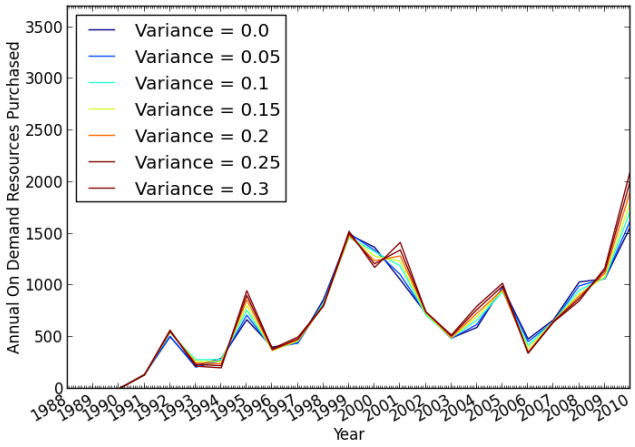

Figure A.8: On-Demand instances purchased under different levels of variance.

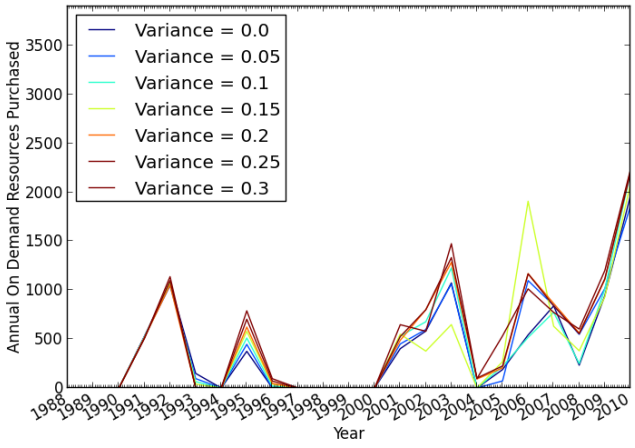

Figure A.9: On-Demand instances purchased under different levels of variance.

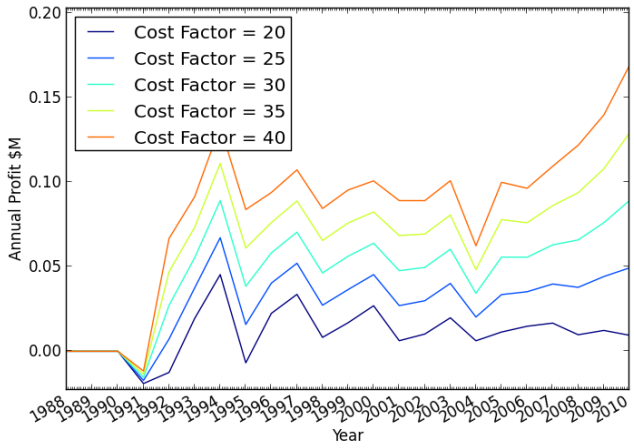

Figure A.10: Annual Broker Profit in Rapid Growth Market at $\theta = \theta_{opt}$ for differing Cost Factors.

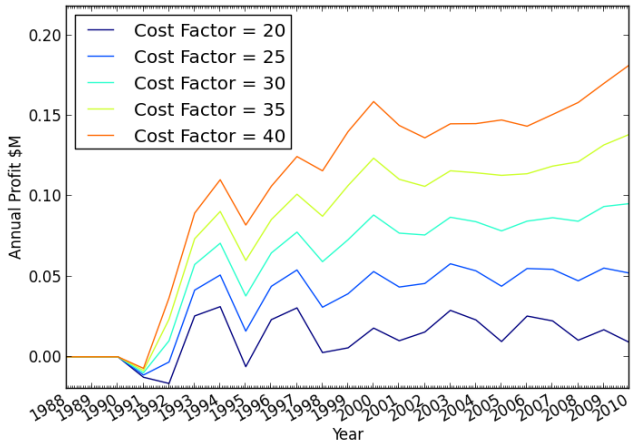

Figure A.11: Annual Broker Profit in Steady Growth Market at $\theta = \theta_{opt}$ for differing Cost Factors.

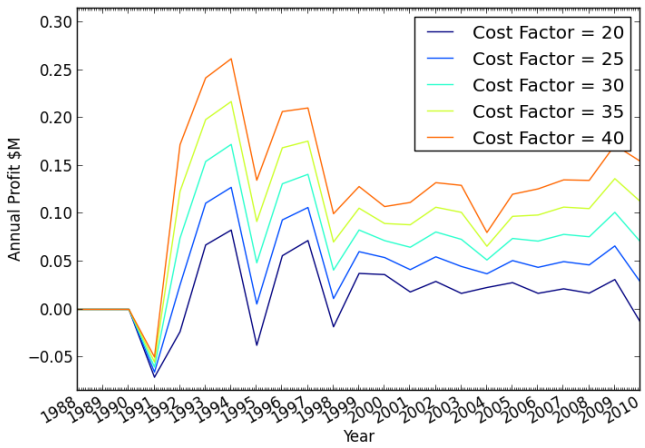

Figure A.12: Annual Broker Profit in Recession Recovery Market at $\theta = \theta_{opt}$ for differing Cost Factors.

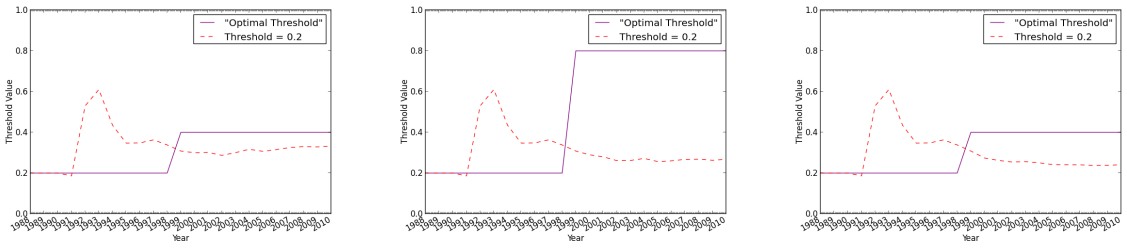

Figure A.13: Market Shock from Steady Growth to Rapid Growth, Recession Recovery and Steady Markets.

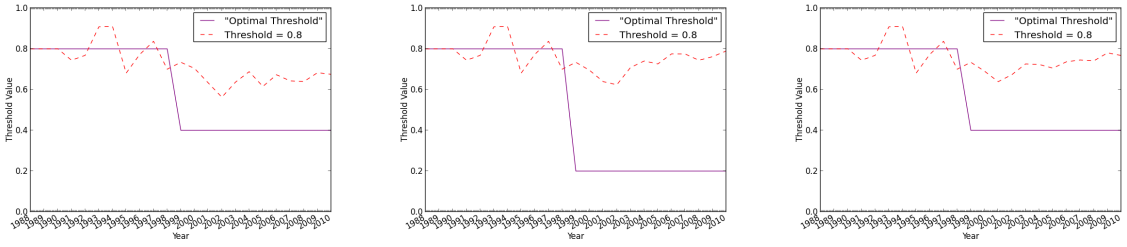

Figure A.14: Market Shock from Recession Recovery to Rapid Growth, Steady Growth and Steady Markets.

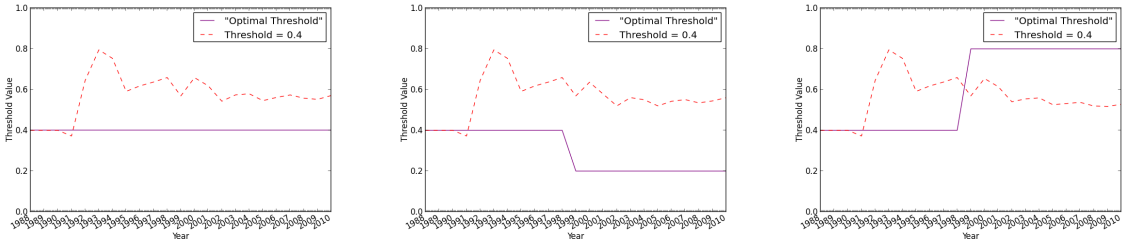

Figure A.15: Market Shock from Steady to Rapid Growth, Steady Growth and Recession Recovery Markets.

Bibliography

[1] Michael Armbrust, Armando Fox, Rean Griffith, Anthony D. Joseph, Randy Katz, Andy Konwinski, Gunho Lee, David Patterson, Ariel Rabkin, Ion Stoica, and Matei Zaharia. Above the clouds: A berkeley view of cloud computing. Technical report, UC Berkeley, 2009.

[2] W. Brian Arthur. Competing technologies, increasing returns, and lock-in by historical events. *The Economic Journal*, 1989.

[3] Tim Berners-Lee. Information management: A proposal. `http://info.cern.ch/Proposal.html`, 1989.

[4] Jeff Bezos. Keynote. y combinator startup school at stanford university. 2008.

[5] Rajkumar Buyya. *High Performance Cluster Computing: Architectures and Systems*, volume 1. Prentice Hall, 1999.

[6] Nicholas Carr. *The Big Switch: Rewiring the world, from Edison to Google*. W. W. Norton, 2009.

[7] J. Cartlidge and D. Ait-Boudaoud. Autonomous virulence adaptation improves coevolutionary optimisation. *IEEE Transactions on Evolutionary Computation*, 15, 2011.

[8] John Cartlidge and Dave Cliff. Comparison of cloud middleware protocols and subscription network topologies using CReST, the cloud research simulation toolkit. In F. Desprez, D. Ferguson, E. Hadar, F. Leymann, M. Jarke, and M. Helfert, editors, *3rd Int. Conf. on Cloud Computing & Services Science (CLOSER-2013)*, pages 58–68, Aachen, Germany, May 2013. SciTePress.

[9] Scott H. Clearwater and Bernardo A. Huberman. Swing options: a mechanism for pricing it peak demand. `http://www.hpl.hp.com/research/idl/papers/swings`, 2005.

[10] D. Cliff and J. Bruten. Minimal-intelligence agents for bargaining behaviours in market-based environments. Technical report, HP Labs, Bristol, 1997.

[11] Dave Cliff. Remotely hosted services and "cloud computing". Technical report, Becta, 2010.

[12] Rachel A. Dines. Build or buy? the economics of data center facilities. Technical report, Forrester Research, Inc, 2011.

[13] Manek Dubash. Moore's law is dead, says gordon moore. *TechWorld*, 2005.

[14] Apache Software Foundation. Apache log4j. `http://logging.apache.org/log4j/1.2/`.

[15] Simson Garfinkel. *Architects of the Information Society, Thirty-Five Years of the Laboratory for Computer Science at MIT.* MIT Press, 1999.

[16] Jeremy Geelan. Twenty-one experts define cloud computing. *Cloud Computing Journal*, 2009.

[17] Google. Data centers. `http://www.google.com/about/datacenters/efficiency/internal`.

[18] Derek Gottfrid. Self-service, prorated supercomputing fun! *New York Times Blog*, 2007.

[19] Gartner Group. Gartner's hype cycle report. Technical report, Gartner Group, 2008.

[20] Brian Hayes. Cloud computing. *Communications of the ACM*, 51, 2008.

[21] Jonathan Heiliger. Building efficient data centers with the open compute project. *Facebook Engineering Notes*, April 2011.

[22] Kai Hwang. Keynote. massively distributed systems: From grids and p2p to clouds. 2008.

[23] LSCITS. Crest - cloud research simulation toolkit. `https://sourceforge.net/projects/cloudresearch/`.

[24] LSCITS. Crest developer's guide, 2012.

[25] Joe McKendrick. Cloud computing's vendor lock-in problem: Why the industry is taking a step backward. *Forbes*, November 2011.

[26] Nine Lives Media. Top 100 cloud services providers: 2012 edition. Technical report, Nine Lives Media, 2012.

[27] Peter Mell and Timothy Grance. The nist definition of cloud computing. Technical report, National Institute of Standards and Technology, U.S. Department of Commerce, 2011.

[28] Dejan Milojicic. Cloud computing: Interview with russ daniels and franco travostino. *IEEE Internet Computing*, 5, 2008.

[29] Gordon E. Moore. Cramming more components onto integrated circuits. *Electronics Magazine*, 38, 1965.

[30] Erica Naone. Technology overview: Conjuring clouds. `http://www.technologyreview.com/briefings/cloud/`, 2009.

[31] Mike P. Papazoglou and Willem-Jan van den Heuvel. Service oriented architectures: Approaches, technologies and research issues. *The VLDB Journal*, 16, 2005.

[32] Carlota Perez. Respecialisation and the deployment of the ict paradigm. Technical report, Universities of Cambridge and Sussex, 2005.

[33] Gregory F. Pfister. *In Search of Clusters*. Prentice Hall, 1998.

[34] Srikumar Venugopal James Broberg Ivona Brandic Rajkumar Buyya, Chee Shin Yeo. Cloud computing and emerging it platforms: Vision, hype, and reality for delivering computing as the 5th utility. *Future Generation Computer Systems*, 25, 2009.

[35] Owen Rogers and Dave Cliff. The effects of truthfulness on a computing resource options market. 2010.

[36] Owen Rogers and Dave Cliff. The effects of market demand on truthfulness in a computing resource options market. 2011.

[37] Owen Rogers and Dave Cliff. A financial brokerage model for cloud computing. *Journal of Cloud Computing: Advances, Systems and Applications*, 1, 2012.

[38] Owen Rogers and Dave Cliff. Forecasting demand for cloud computing resources: An agent-based simulation of a two tiered approach. 2012.

[39] D. E. Rumelhart, G. E. Hinton, and R. J. Williams. *Learning Internal Representations by Error Propagation*, volume 1. 1986.

[40] Salesforce. Force.com platform as a service. `http://www.force.com`.

[41] Eric Shmidt. Atmosphere: Fireside chat with eric schmidt. `http://www.youtube.com/watch?v=qBaVyCcw47M`, 2010.

[42] A. Stage and T Setzer. Network-aware migration control and scheduling of differentiated virtual machine workloads. *ICSE Workshop on Software Engineering Challenges of Cloud Computing*, 2009.

[43] Luis Vaquero, Luis Rodero-Merino, Juan Caceres, and Maik Lindner. A break in the clouds: Towards a cloud definition. Technical report, Telefonica Investigacion y Desarrollo and SAP Research, 2009.

[44] Werner Vogels. A head in the clouds - the power of infrastructure as a service. `http://www.youtube.com/watch?v=9AS8zzUaO3Y`, 2008.

[45] Kevin Wallsten. "yes we can": How online viewership, blog discussion, campaign statements, and mainstream media coverage produced a viral video phenomenon. *Journal of Information Technology & Politics*, 7, 2010.

[46] B. Widrow and Jr. M. E. Hoff. Adaptive switching circuits. *IRE WESCON Convention Rec.*, 4, 1960.

[47] S. W. Wilson. Zcs: A zeroth level classifier system. *Evolutionary Computation*, 2, 1994.

[48] S. W. Wilson. Classifier fitness based on accuracy. *Evolutionary Computation*, 3, 1995.

[49] Fang Wu, Li Zhang, and Bernardo A. Huberman. Truth-telling reservations. *Algorithmica*, 52, 2008.