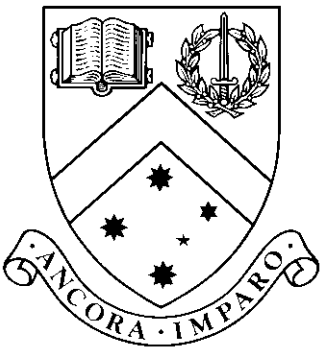
An Evaluation of Configuration Management for High Performance Computing on Clouds

by

Tian Yu Goh, BITS



Thesis

Submitted by Tian Yu Goh

in partial fulfillment of the Requirements for the Degree of

Honours degree of Bachelor of Information Technology and Systems (3336)

Supervisor: Dr. Jefferson Tan

Caulfield School of Information Technology
Monash University

July, 2013

To my family, for supporting me through my research endeavours.

Contents

List of Tables

List of Figures

Listings

An Evaluation of Configuration Management for High Performance Computing on Clouds

Tian Yu Goh, BITS
[redacted]
Monash University, 2013

Supervisor: Dr. Jefferson Tan
[redacted]

Abstract

This thesis has developed a Dynamic Configuration Management Framework and implemented a prototype using Puppet Enterprise to support evaluation of the suitability of said framework. The field of research in Configuration Management, specifically in the High Performance Computing Application area for Cloud infrastructures is still under developed and this research contributes greatly to that field of research, serving as a stepping stone for future work to be done. The results from the implementation of the prototype demonstrates an overwhelming increase in efficiency and reliability of Configuration Management of 77% to 81%, and 89% to 100% respectively. This assures the framework of two facts. Firstly, the proof that the use of the framework proposed and the Configuration Management tool has a positive effect on efficiency, and secondly, that the reliability of the system Configuration Management increases when compared to a traditional style of Configuration Management, that is – the manual or semi-scripted process. This thesis demonstrates the feasibility on the investment for a Configuration Management tool in use with distributed infrastructure such as the Cloud. The framework proposed also serves as a guideline and process chart for Configuration Management of dynamic packages and High Performance Computing application requirements.

An Evaluation of Configuration Management for High Performance Computing on Clouds

Declaration

I declare that this thesis is my own work and has not been submitted in any form for another degree or diploma at any university or other institute of tertiary education. Information derived from the published and unpublished work of others has been acknowledged in the text and a list of references is given.

Tian Yu Goh
July 18, 2013

Acknowledgments

I would like to extend my thanks to my supervisor, Dr. Jefferson Tan, for the encouragement, interest and constant guidance he has given me over the course of this research work. I would also like to thank Professor Frada Burstein and Morgan Priestnall for providing me with inspiration and insight, and Rick Wu for his help throughout this research work. Last but not least, I would like to acknowledge my fellow Honours students, who have helped make the past year fun and enjoyable. It has been a great journey thus far, and I look forward to where my research will take me.

Tian Yu Goh

Monash University
July 2013

Chapter 1

Introduction

1.1 Preamble

With the recent advances in computing power in High Performance Computing (HPC), researchers have been exploring various methods and 'frameworks' available for HPC applications, especially in eResearch and eScience. The need for more computing power grows daily as an increasing amount of research turn to HPC as a means to model, or simulate various experiments before carrying the actual experiments out in order to reduce the number of actual samples and experiments required, which results in time and money saved. In the traditional context, HPC would have referred to expensive supercomputers, or dedicated clusters of processing power. With the advent of Cloud Computing over half a decade ago, a large amount of attention has been focused on the Cloud paradigm from researchers and organisations alike – looking for newer ways to exploit and utilise the features that the Cloud brings about, which are basically the flexibility or scalability (elasticity) of on-demand processing power (Fujimoto et al., 2010; Brandt et al., 2009). Cloud Computing has become a large cost-saving option for organisations with rapidly growing data and for organisations that perform scientific research which involve "in silico" experiments (Oliveira et al., 2010, 2009).

In recent years, there has been a multitude of studies conducted, exploring the possibility of HPC applications on Cloud Computing platforms. The promise of HPC power available on-demand', as a cheaper alternative to the traditional concept of HPC – supercomputers and dedicated clusters, as a much more reliable platform than grid computing, while combining the scalability of clusters and resource pools (Ostermann et al., 2010; Jackson et al., 2010). As the amount of computing power available increases, the general problem with Configuration Management becomes a greater priority over time – the more computer resources there are, the more difficult, and time-consuming it is to manage the configuration of such resources.

The important question remains – How do we manage these resources in an orderly manner? Although prior research in attempts to compare and illustrate various CM tools has been carried out (Jahidur, 2012; Önnberg, 2012), research into the suitability of tools, and the process of Configuration Management for High Performance Computing on the Cloud platform remains mostly unexplored.

1.2 Problem Statement

High Performance Computing is an area of great interest, especially to organisations that perform large-scale research involving simulations, and is a growing area of interest for organisations that must process a lot of data every day. Institutions such as Monash University utilise HPC in e-Research, which is the utilisation of Advanced Computing and Information Communication Technologies (ICT) to assist in research projects. Some of these projects require the use of large computational resources to run simulations in parallel, without which it would take a much longer time to complete. These simulations are able to assist in and eliminating a degree of actual experimentation that needs to be done in the process of research. The general problem with this is that the more computer resources (computing power) there are, the higher the difficulty in, and the more time-consuming it is to manage the configuration of such resources. Within large scale projects, the process of installing a number of software packages multiple times with specific configurations for as many computational nodes or instances as there are involved, can easily scale from one to many dozens of such instances; dramatically increasing the workload involved. In summary, the main question would be to find out how and what processes and components would be required in order for efficient and reliable Configuration Management activities for High Performance Computing on the Cloud platform.

1.3 Aims and Outcomes

This thesis aims to design a framework for Configuration Management activities for High Performance Computing components on the Cloud, and to evaluate the effect on efficiency and reliability of a Configuration Management tool as a prototype that fits the framework proposed, in configuring multiple Virtual Machines which can be extended to Cloud Nodes. This research project will produce a dynamic Configuration Management framework that is viable, and a working prototype will be developed and deployed in order to demonstrate the validity of the proposed dynamic Configuration Management framework for HPC/HTC applications on Clouds or other distributed infrastructure. An evaluation will be conducted on the framework and the prototype to determine if it meets the requirements and improves

the ease at which Configuration Management will be carried out. The criteria for evaluation of the Configuration Management tool are as follows:

- Efficiency in Configuration Management
- Reliability with the configuration deployed

1.4 Benefits and Contributions

Some of the expected benefits and contributions are listed as follows:

- Proposal of a framework for dynamic Configuration Management needs
- Evaluation of Puppet Enterprise as a Configuration Management Tool for Clouds
- Evaluation on how well Puppet Enterprise performs as a Configuration Management tool when applied to HPC applications.
- Importance of Configuration Management in HPC

1.5 Thesis Outline

This thesis consists of five main chapters as outlined below.

- **Chapter 2** reviews the current existing literature, and introduces the key concepts of High Performance Computing, Cloud Computing, Configuration Management and addresses the issue of filling the gap identified between traditional static Configuration Management and Dynamic Configuration Management, in the perspective for High Performance Computing Applications on the Cloud platform.
- **Chapter 3** introduces the methods and components identified that is of interest to this research, specifically in terms of High Performance Computing applications on the Cloud, or distributed infrastructure. A framework is proposed and described, and some details on the implementation and evaluation are outlined.
- **Chapter 4** discusses the implementation of the prototype by introducing the environment, and discussing the Configuration Management tool being deployed, following which a discussion of the implementation process is provided.

- **Chapter 5** describes the evaluation process of the proposed framework and the prototype, as well as the results obtained from the evaluation process. The evaluation process is outlined and a short analysis of the results is carried out.

- **Chapter 6** brings the research work into its conclusion, and provides a summary of this thesis, as well as the achievements and contributions made, the limitations on the research is acknowledged, and some difficulties and problems encountered are presented. Finally, some identified directions for future works are provided.

Chapter 2

Literature Review

2.1 High Performance Computing

HPC is defined by iVEC as the "use of computers for high throughput of computation, solving large problems, or getting results faster" (iVEC, 2012). One context that this research will examine is e-Science, defined as the "large scale science that will increasingly be carried out through distributed global collaborations enabled by the Internet" (NeSC, 2001), and in a typical scenario or setting, such work requires access to large data collections, computing resources at scale, and high performance computer visualisation tools at the individual scientist level.

HPC can be considered an umbrella of a number of different architectures or environments such as clusters and supercomputers. As described in iVEC's definition, HPC is considered the usage of computer resources for high throughput of computation, etc. It is important to note that when considering HPC, the environment or architecture to be used greatly depends on the task at hand; for example, is a large high performance, multiprocessor system such as a supercomputer required, or would it be more suitable to run the job at hand on a cluster-like system, which concentrates more on the throughput over a large number of tasks that is spread over a longer period of time (Strohmaier et al., 2005; Mahmoud et al., 2012).

The Monash e-Science and Grid Engineering Laboratory (MeSsAGE Lab) plays a role in e-Science in many fields of research, including but not limited to, Chemistry and Physics, Medical and Life Sciences, Engineering and Design, Mathematics and Computer Science, Economics and Finance, Environmental Science, Earth Sciences and Astronomy (MeSsAGELab, 2012). In that context, HPC plays a large role in e-Research, defined as "a set of activities that harness the power of advanced information and communication technologies (ICTs) for research" (eResearchSA, 2012), in providing the computational power required to simulate and model a vast variety of data for e-Research, that without HPC, would have been computationally

infeasible. An example of research that was carried out using such methods is Abramson et al. (2006)'s research into Drug Design where some of the processes in simulating and modelling the effects of a drug were done using HPC.

With the availability of HPC computing power, scientists executing parallel or distributed simulation and modelling applications are able to try and scale the amount of simulations / models done in order to achieve a higher rate of accuracy or more results. With that scaling, more resources are required to compute the same simulation or model (Gallard et al., 2012, p. 136). As with before, the limitations on the research conducted would be typically determined by the computing power that the researchers home institution or national initiatives could provide. In that sense, resources available were typically clusters that were being managed by batch queuing applications or something similar. This restriction forced researchers to either find the capacity for computing power from an external entity, abort or compromise on the result sample set, or reduce their scope in hope of reducing the required capacity of computational power (Bethwaite et al., 2010, p.221).

2.2 Cloud Computing

Cloud computing is defined by NIST as "a model for enabling ubiquitous, convenient, on-demand network access to a shared pool of configurable computing resources (e.g., networks, servers, storage, applications, and services) that can be rapidly provisioned and released with minimal management effort or service provider interaction" (Mell and Grance, 2009), with five essential characteristics which are listed as follows:

- on-demand self-service
- broad network access
- resource pooling
- rapid elasticity
- measured service

Throughout the years, many researchers have attempted to classify the service models that are covered under the Cloud computing paradigm. For example, Wang et al. (2008) describes it as being three service models, primarily Hardware as a Service (HaaS), Software as a Service (SaaS), and Data as a Service (DaaS). This is contrary to the definition provided by Mell and Grance (2009) as the three service models of SaaS, Platform as a Service (PaaS), and Infrastructure as a Service (IaaS). Some other researchers have developed their own classifications that either extend

or provide an alternative view into these service models, which include but are not limited to: Cluster as a Service (CaaS) (Dodai, 2011; Yokoyama and Yoshioka, 2012; Yokoyama et al., 2012), High Performance Computing as a Service (HPCaaS) (Jamjoom et al., 2012).

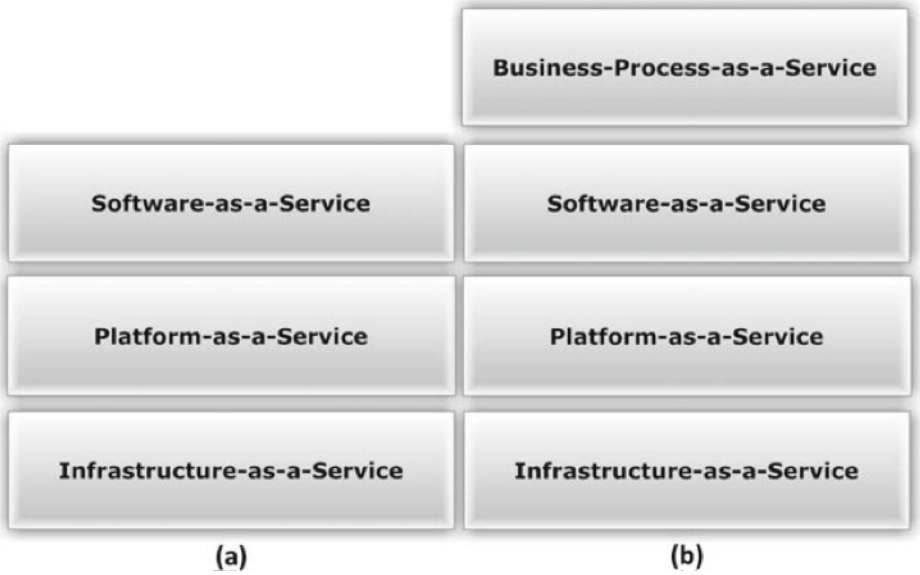


Figure 2.1: (a) SPI service model vs. (b) IBM service model (Hamdaqa and Tahvildari, 2012, p. 50)

The commonly accepted models are the NIST Software Platform Infrastructure (SPI) model, and the IBM model as shown in a comparison at Figure 2.1. It is seen that, the two service models are complementary to each other, and not contradictory (Ahson and Ilyas, 2010; Hamdaqa and Tahvildari, 2012). For the purposes of this research, the NIST SPI model will be adopted for use to comply with widely accepted standards. A layered model architecture following the NIST SPI model was introduced by (Zhang et al., 2010) and is shown in Figure 2.2.

As seen in Figure 2.2, there are three types of services on the leftmost column, and these are as follows:

- **IaaS**: This layer includes the hardware and infrastructure, and is commonly provided by data centres and infrastructure providers such as Amazon EC2, Microsoft Azure, Rackspace, IBM SmartCloud, GoGrid, and FlexiScale. The provider typically provides and is responsible for the physical resources in the cloud, as well as the network and cooling equipment that goes with it. The infrastructure layer is more commonly known as the virtualization layer, and generally can be referred to as the pool of resources (computing and storage) that is a key component of Cloud computing, especially since “many

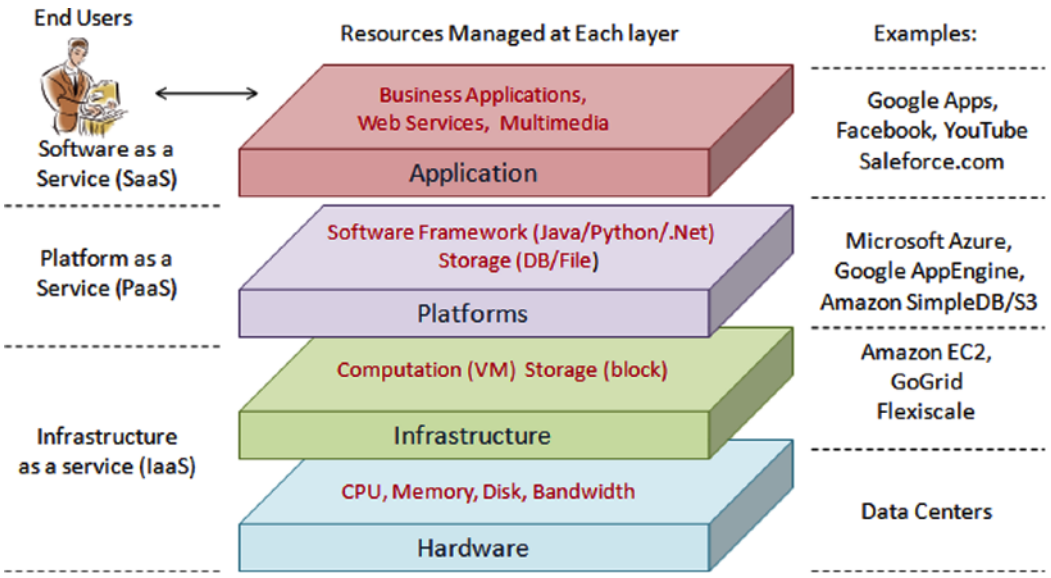


Figure 2.2: Cloud Computing Architecture (Zhang et al., 2010, p. 9)

key features, such as dynamic resource assignment, are only made available through virtualization technologies" (Zhang et al., 2010, p. 9).

- **PaaS**: The platform layer concerns itself with the operating systems and frameworks, such as Application Programming Interfaces (API) (Google App Engine) to provide API support for application implementations.
- **SaaS**: The application layer contains the cloud applications such as Facebook, Twitter, YouTube, and Google Apps. The main difference between traditional applications and cloud applications is that cloud applications are able to scale automatically on resources in order to achieve higher performance and availability at lower operating costs (Zhang et al., 2010, p. 9).

Some of the popularly used IaaS frameworks available are:

- Eucalyptus
- Nimbus
- OpenStack
- OpenNebula

In von Laszewski et al. (2012)'s work, a comparison of the IaaS frameworks was conducted and Figure 2.3 as shown was derived from that data, comparing the features of the various IaaS frameworks.

The authors also evaluated the popularity of the frameworks and noted that Nimbus and Eucalyptus were the most popular, but might have been so as they had

✓ indicates a positive evaluation. The more checkmarks the better.

	OpenStack	Eucalyptus 2.0	Nimbus	OpenNebula
Interfaces	EC2 and S3, Rest Interface. Working on OCCI ✓✓	EC2 and S3, Rest Interface. Working on OCCI ✓✓	EC2 and S3, Rest Interface ✓	Native XML/RPC, EC2 and S3, OCCI, Rest Interface ✓✓✓
Hypervisor	KVM, XEN, VMware Vsphere, LXC, UML and MS HyperV ✓✓✓	KVM and XEN. VMWare in the enterprise edition. ✓✓	KVM and XEN ✓	KVM, XEN and VMWare ✓✓
Networking	- Two modes: (a) Flat networking (b) VLAN networking -Creates Bridges automatically -Uses IP forwarding for public IP -VMs only have private IPs ✓✓✓	- Four modes: (a) managed; (b) managed-novLAN; (c) system; and (d) static - In (a) & (b) bridges are created automatically - IP forwarding for public IP -VMs only have private IPs ✓✓✓	- IP assigned using a DHCP server that can be configured in two ways. - Bridges must exists in the compute nodes ✓✓	- Networks can be defined to support Ebtable, Open vSwitch and 802.1Q tagging -Bridges must exists in the compute nodes -IP are setup inside VM ✓✓✓
Software deployment	- Software is composed by component that can be placed in different machines. - Compute nodes need to install OpenStack software ✓	- Software is composed by component that can be placed in different machines. - Compute nodes need to install OpenStack software ✓	Software is installed in frontend and compute nodes ✓✓	Software is installed in frontend ✓✓✓
DevOps deployment	Chef, Crowbar, Puppet ✓✓✓	Chef*, Puppet* ✓ (*according to vendor)	no	Chef, Puppet ✓✓
Storage (Image Transference)	- Swift (http/s) - Unix filesystem (ssh) ✓	Walrus (http/s) ✓	Cumulus (http/https) ✓	Unix Filesystem (ssh, shared filesystem or LVM with CoW) ✓
Authentication	X509 credentials, LDAP ✓✓✓	X509 credentials ✓	X509 credentials, Grids ✓✓	X509 credential, ssh rsa keypair, password, LDAP ✓✓✓
Avg. Release Frequency	<4month	>4 month ✓	<4 month	>6 month ✓
License	OpenSource - Apache ✓	OpenSource ≠ Commercial	OpenSource Apache ✓	OpenSource Apache ✓

Figure 2.3: Comparison of Multiple IaaS Cloud Frameworks (von Laszewski et al., 2012, p. 738)

strongly advertised and recommended the two systems (von Laszewski et al., 2012, p. 735).

The NIST definition of Cloud computing also provides for four deployment models (Mell and Grance, 2009) as follows:

- **Private Cloud**. Cloud infrastructure used exclusively by a single organisation. For example, a Private Cloud Storage solution provided by Tonido (2012).

- **Community Cloud**. Cloud infrastructure used exclusively by a single community of organisations (consumers) with a single shared concern (e.g. mission, security requirements, etc.). An example would be a shared Google Document Folder.

- **Public Cloud**. Public Cloud is used by the general public, and is hosted by a cloud provider. An example would be Rackspaces Public Cloud (Rackspace, 2012).

- **Hybrid Cloud**. A Hybrid Cloud is a "composition of two or more distinct cloud infrastructures (private, community, or public) that remain unique entities, but are bound together by standardised or proprietary technology that

enables data and application portability" (Mell and Grance, 2009). One example of this is Intel's Hybrid Cloud Platform (Intel, 2012).

It is noted that access to cloud environments has been classified into sub-taxonomies by research. Oliveira et al. (2010, p. 56) stated "API is a fundamental artifact for access through programming languages such as Java, Python, or C. By using an API, more complex applications may use cloud infrastructure in a native form", and defines access into four types: web browsers, thin clients, mobile clients and API. A limitation exists on the API in cloud environments. For example, Google App Engine (Google, 2012) "provide not only a cloud computing infrastructure, but also a complete software stack with a restricted API so that software developers are forced to write programs that can run in a shared-nothing environment and thus facilitate elastic scaling" (Abadi, 2009, pp. 2-3).

Being the emergent trend for delivering Information Technology (IT) services to end-users and enterprises (Vecchiola et al., 2009, p. 5), Cloud computing has been gaining a lot of interest, especially from scientists and researchers alike with the ability to scale computing resources according to the requirements and cost budgets of the project or the end user (Vecchiola et al., 2009, p. 5). In the field of e-Research, there have been a large number of studies done, quantifying the performance of such platforms in its use as HPC for scientific applications, and "in silico" experiments. There have been studies that investigated specific science (e-Science) applications on clouds (Evangelinos and Hill, 2008), but it is noted that said studies were sensitive to network latency – which would be a major negative factor on performance as cloud resources are not located in a single data centre, and can be distributed around the world in multiple geographic locations.

As mentioned previously, e-Science and e-Research assist researchers by providing computing power to simulate and model a vast variety of data for research. This, in turn reduces the amount of physical real-world experiments required for most projects, which reduces the cost of the entire research as a whole. Cloud computing introduces the on-demand, pay-per-use cost structure. The idea of being able to rent scalable computational power on demand is of interest to scientists and researchers alike. Being able to perform more experiments on the same budget will go a long way in helping research objectives. As mentioned in the previous section, HPC can be broken down into various architectures, such as the cluster or supercomputer. The Cloud paradigm is better described in those architectures as a cluster-like environment, with benefits such as the rapid expandability or scalability to virtually an unlimited amount of processors to better support a massive amount of jobs that are running simultaneously, an 'instant cluster' that can be deployed on-demand on the cloud, so to say. This type of computational environment is more

commonly called the High-Throughput Computing (HTC), or Many Task Computing (MTC) environment, which was described by Raicu et al. (2008) as the bridge between the HTC and HPC paradigms.

Some current ongoing issues with Cloud computing as HPC include performance issues stemming from communications and interference from external traffic over a network(s). It was observed in some studies that parallel codes executed slower when compared to execution on dedicated nodes (Fujimoto et al., 2010, p. 3), and a strong correlation was identified between the percentage of time that a HPC application spends communicating between computational nodes, and the overall performance of the application itself, which basically meant that when the HPC application had more inter-nodal communication, the overall performance of the HPC application decreases (Jackson et al., 2010, p. 167). Another study states that unlike the cluster computing architecture, the Cloud computing paradigm attempts to focus on increasing the overall system performance of the cloud as a whole (Dillon et al., 2010, p. 30).

It can be said that Cloud computing will work for parallel problems with little to no inter-job communications. Support for "handling large data sets, the concept of moving computation to data, and the better quality of services provided such as fault tolerance and monitoring, simplify the implementation details of such problems over the traditional systems" (Ekanayake and Fox, 2010, p. 36). As such, the type of architecture that suits the Cloud paradigm applied to HPC applications, as mentioned previously, is more of a 'instant cluster-like' environment. There exists a need to research and enhance the cloud to produce better performance, scalability and stability (Ramakrishnan et al., 2011, p. 57). Considering that HPC environments such as supercomputers may contain just one shared filesystem for all the computational cores available, so packages are only installed once, as compared to a cluster-like environment, where there is a need to deploy the same package to a large amount of filesystems, with an identical configuration. The question then lies in the ability to ensure that the installations for the packages required are done in a fast, efficient and reliable manner.

2.3 Configuration Management

The definition of Software Configuration Management (SCM) by IEEE is widely accepted (IEEE, 1988, 1990). CM is a relatively large function that covers not only the management of configurations of hardware and software, but also includes inventory control and management, configuration of hardware and software, storage, backup, and comparison, implementation of configuration changes, and detection of changes to configuration, hardware and software (Cisco, 2007). Configuration Management

also allows a default configuration to be specified for specific devices, modify and to load new configurations on them (Oppenheimer, 2011). Wood and Pereira (2011, p. 19) state "Configuration is the most critical process of any heterogeneous network. This is because it impacts the network in terms of security, performance, resilience, predictability. Distributed computing paradigms have changed system configuration management more than could have been predicted as the dynamism of such systems requires more complex set up and maintenance". CM has become a necessity in order to reduce the error rate and difficulty in which it has become to maintain and deploy components in large infrastructures. Magherusan-Stanciu et al. (2011, p. 25) has stated that the complexity and the workload of administrative staff increase with the amount of computational power that is added into the system. This proves especially true when the amount of administration staff do not increase, but the ratio of workload to staff members increases with each machine being added to the pool of resources available.

CM tools that assist in automation with regard to configuration and deployment of resources are available. Önnberg (2012) illustrates in Figure 2.4, a table of such Configuration Management tools and their supported platforms.

	AIX	HP-UX	Linux	BSD	Mac OS X	Solaris	Windows
Bcfg2	/		X	X	/	X	
cdist			X	X	X		
CFEngine	X	X	X	X	X	X	X
Chef	X	X	X	X	X	X	X
DACS			X	X		X	
etch			X	X		X	
LCFG			/		/	/	
Opsi							X
Pallet			X		X		
Puppet	X	X	X	X	X	X	X
Quattor			X		X		
Salt			X	X			/
Spacewalk			X			X	
X = Supported, / = Partial Support							

Figure 2.4: List of System Configuration Management Software and their supported platforms (Önnberg, 2012, p. 5)

A study conducted by Delaet, Anderson, and Joosen (2008) show that most of the existing 'modern' systems already require sufficiently complex system configurations that are normally beyond the ability of human administrators to configure manually in a reliable manner, "configuration errors are the biggest contributors to service failures (between 40% and 51%), and these errors take the longest time to repair" (Delaet et al., 2008; Oppenheimer, 2003; Patterson, 2002, p. 594). A set

of case studies conducted by Oppenheimer (2003) has demonstrated CMs importance, and the importance of understanding sufficiently, the system configuration in a distributed system configuration. He states that incorrect or insufficient understanding of such distributed system configuration can influence service availability when either performing some kind of action or when diagnosing a problem with the service (Oppenheimer, 2003, p. 1). This is especially important when there isnt any proper CM tool or process in effect that will allow the administrator to understand the current configuration before proceeding to make changes that will affect the dependencies, both down and upstream.

Historical research in the area has identified the two major streams of Configuration Management, namely the static traditional process, and the dynamic process (Kramer and Magee, 1985). Dynamic Configuration Management isn't a new concept, but it has been slowly developing and there is a disturbing lack of research in the area pertaining to components for High Performance Computing applications on the Cloud. As mentioned in the previous section, the Cloud paradigm has been growing and attracting great interest for the benefits of rapid scalability and on-demand deployment in the High Performance Computing field. It then makes sense to utilise a framework of dynamic Configuration Management in order to smoothen and provide a logical order to the complexity of Configuration Management.

2.4 CM for HPC Applications on the Cloud

Oren Michels, CEO of Mashery said that "Although the root cause of this particular issue was a resource contention issue between instances, things like that are going to happen. There may now be a fix for this particular edge case, but there are undoubtedly others that will crop up over time. The real failure here was a failure of monitoring, and a failure of transparency" (Gillett et al., 2008) in describing the downtime experienced by Amazons Elastic Compute Cloud (EC2) service, pointing out the need for Fault, Configuration, Accounting, Performance and Security (FCAPS) measurement, management and optimisation (Mikkilineni and Sarathy, 2009, p. 60). CM fits into the second point of the FCAPS framework, which consists of five points; Fault Management, Configuration Management, Accounting Management, Performance Management, and Security Management (Allan and Nadeau, 2006). In Cloud computing, the need for CM is even more urgent, given the large amount of computational resources that tend to reside within it.

In order to increase productivity and efficiency of deployments and of the cloud, it is necessary to reduce the complexity and/or the administrative workload that the cloud brings about by automating configuration and maintenance, and providing a

method to handle CM (Magherusan-Stanciu et al., 2011; Rimal et al., 2010). The best way to handle such environments, especially in data centres is to utilise CM tools and frameworks such as Puppet, cfengine, KickStart and Chef. Such tools assist in configuration management by providing a framework for which to write program/configuration code by, as compared to writing code by hand (Rimal et al., 2010, p. 29).

In larger distributed systems, for example on a cloud, it is impossible to completely shut down the servers for reconfiguration. Hence there is a need to be able to dynamically reconfigure the servers while they are in operation. Research has identified that there is "a further need to access and reconfigure resources and services controlled by different organisations. These systems are too large and complex to be managed by a single human manager" (Crane et al., 1995, p. 1). This holds especially true when we discuss CM on the Cloud. The basic idea of a cloud means that resources are not always geographically aligned, and can be spread all over the globe. Hence, as discussed in a previous section, the ability to deploy multiple packages of identical configurations to a large number of nodes within the distributed system on demand is critical.

Cloud (IaaS) providers can be typically divided into two types. They will either provide support for CM via their in-house tools, or allow a user to utilise a third party tool (typically open source), such as Puppet. In recent research conducted by Jahidur (2012), he identified some benefits of having proper CM. For example, the "opportunity of better control over all the nodes through visibility and tracking", and "enhanced understanding, supervision and managing of complicated system and infrastructure" (Jahidur, 2012, p. 8).

An example of the installation process of the overhead on manual setup of instances (nodes) on Amazon EC2 Cloud is shown in Figure 2.5. As shown in Figure 2.5, the boxed area has to be repeated for every instance to be setup (Ostermann et al., 2010, p. 117). In a benchmarking study by (Ostermann et al., 2010, p. 120), they have conducted a repetition of 20 times for the five instance types, for a total of 100 instances (nodes), by hand – which is rather time consuming. One point to note is that since the image used to instantiate the nodes are, or can be unique to the software / applications required by the research project, it is required to re-instantiate all the nodes for each research project (Gallard et al., 2012, p. 138). This is very time consuming, hence there are CM tools available to assist in this matter. Some of these tools are shown in the following sections.

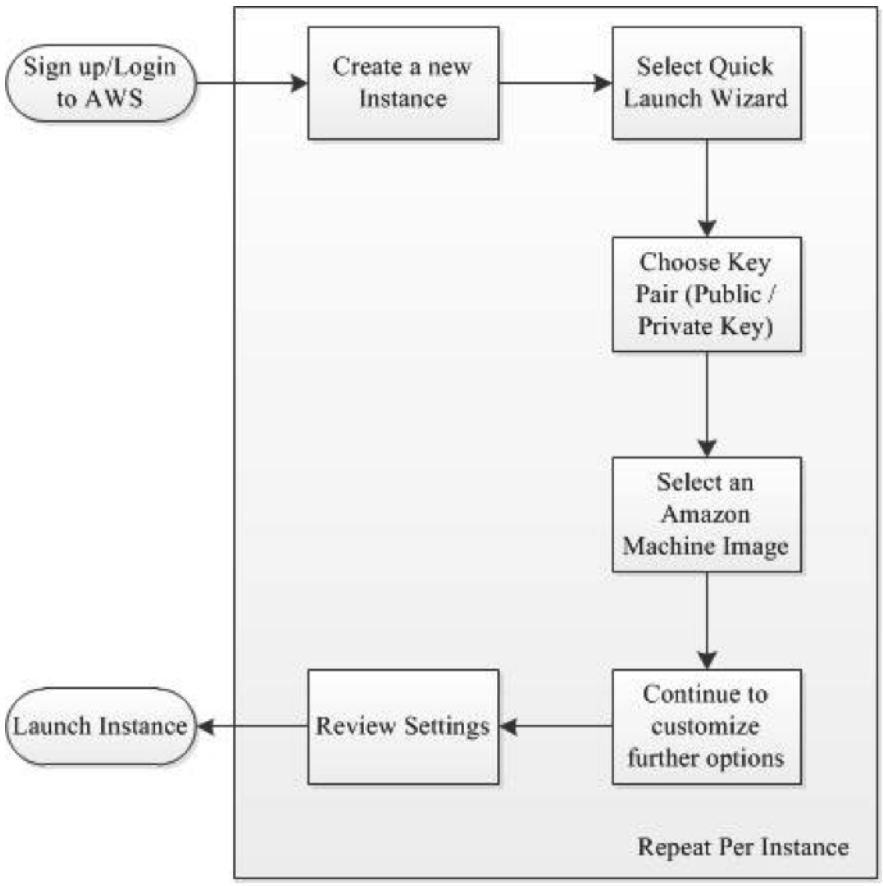


Figure 2.5: Amazon EC2 Instance Manual Setup adapted from (Amazon, 2012)

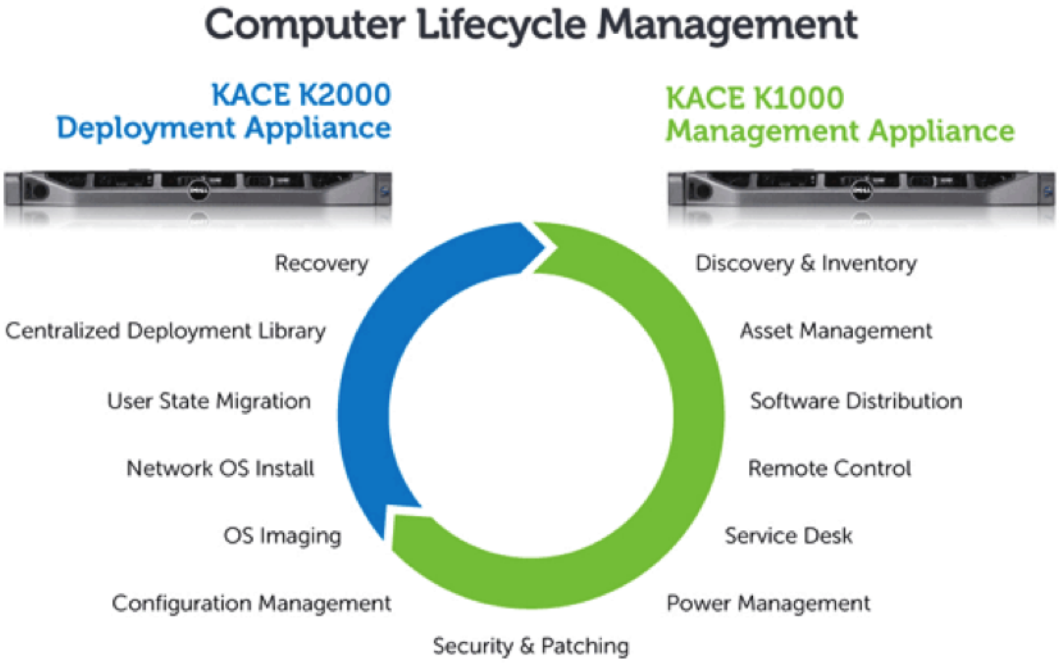


Figure 2.6: Dell KACE tool (Dell, 2012)

2.4.1 Vendor provided tools

Dell KACE

Dell provides what it calls a multi-role appliance that is primarily divided into two categories as shown in Figure 2.6. The K2000 focuses on deployment, and the K1000 focuses on management. The KACE tool also only supports certain configurations, such as Windows, Mac, RedHat Unix, Ubuntu, and SUSE, with the Unix family only being supported by the K1000. It supports only the VMware ESX and ESXi Server 3.5 and 4.0, and the Open Virtualization Format (OVF).

IBM Tivoli

IBM provides a software solution it calls Tivoli. Tivoli is capable of providing support from Assessment, Design, and Deployment to Maintenance (IBM, 2010). In terms of support, Tivoli seems to support a larger range of platforms, but does not have a definitive list for all its product offerings (IBM, 2012).

VMware Tools

VMware have their own VMware Tools which supports a multitude of platforms as well, but work specifically with VMware virtualization appliances (VMware, 2012).

2.4.2 Third-party tools

Puppetlabs Puppet / Puppet Enterprise

Puppetlabs offers two alternatives, an open-source alternative (Puppet) that works only with Amazon EC2 and offers support for CM in Operating Systems (OS) and Applications, or Puppet Enterprise which works with VMware as well, and offers a more complete range of support for CM (PuppetLabs, 2012a). Puppet works by providing a declarative language to program in; definitions of the required instances are then created by the user and instantiated by the tool. The tool provides some degree of automation, and users are able to put together instances from a library of configuration scripts (definition scripts), which means that users are able to customise the entire instance easily and be able to create multiple instances using the automation provided. Puppet has some limitations, being that it was designed for Unix OS; however it can run on Windows Server 2003, Server 2008 R2 and Windows 7 as well.

Ubuntu CloudInit

Ubuntu provides its own tool called CloudInit. It provides CM for the early initialisation of instances (Ubuntu, 2012). It is available by default on the Ubuntu Cloud Image, and on Amazon EC2 Amazon Machine Image (AMI) for Ubuntu.

2.5 Past Related studies

Other methods of CM attempts include image management by Cloud and HPC vendor hosts. Generally, these hosts do not provide custom image storage for end-users who may want to swap between various sets of operating environments and application packages. In this instance, a proper CM tool would allow the user to dynamically adjust for the applications installed, instead of having to store an entire set of images of the system, and the packages available. This saves storage space (which costs the end-user money), and the worry about having to keep all the packages up to date. An alternative to having a sufficiently dynamic CM tool that enables such easy swapping and modification of configuration was suggested by Diaz, von Laszewski, Wang, and Fox (2012). They suggested creating a Universal Image Registrar for Cloud and HPC Infrastructures and offered the FutureGrid (2012) Image Management Architecture to instantiate and store new, previously un-created images to the IaaS Cloud Framework and HPC Cluster (Bare-Metal) Infrastructure. As mentioned previously, research done by Kramer and Magee (1985) demonstrated the differences between static and dynamic configuration processes, and discussed the dynamic configuration process for distributed systems. The importance of CM on distributed or Cloud infrastructure has also been discussed in studies done (Wood and Pereira, 2011; Oppenheimer, 2003; Crane et al., 1995).

In the area of e-Science applications and HPC applications on the Cloud, some research has been done, some of these researchers have discussed their experiences and results obtained as a process of actually running the applications on the cloud (Ramakrishnan et al., 2011; Klinginsmith et al., 2011). Other researchers have benchmarked their studies and described the performance of HPC on the Cloud (Vöckler et al., 2011; Strijkers et al., 2010; Simmhan et al., 2010; Fujimoto et al., 2010). Ramakrishnan et al. (2010) described how the existing Cloud environments can be improved for e-Science usage, and research in analysing and benchmarking the performance of HPC applications on existing Cloud infrastructure (Hill and Humphrey, 2009). Wang et al. (2009) described how MTC and HTC service providers benefitted from the economies of scale in using the Cloud platform, and in more recent work, a thesis was published investigating the use of CM tools in large infrastructures (Rahman, 2012).

Chapter 3

Research Design

As mentioned previously, this research project aims to design a framework for dynamic CM of virtual HPC nodes in the Cloud, as well as to develop a prototype to demonstrate the fitness of the framework proposed, as well as to demonstrate the effect of a CM tool on the efficiency and reliability criterions. In order to do so, it is necessary to first identify what components are critical to CM and of interest to the main process of dynamic CM for this research.

3.1 Open-Source CM Tools

The analysis of the components required was done by looking at the currently available commercial and open source CM tools available. There are a number of such tools on the market currently, and two of the more popular open-source tools, Puppet and Chef, are described below with their core functionalities.
The two tools analysed are Puppet and Chef. The model of operation on both tools are very similar to each other, and generally follow an agent-server mode of communication.

3.1.1 Puppetlabs Puppet

Puppet is described as an IT automation tool that provides the capability for system administrators to conduct CM activities, by establishing a reference configuration state for which to monitor configuration drift from. It can then accept or reject further changes, or automatically force a state of configuration on a group of resources. (PuppetLabs, 2012b) Puppet utilises a four step process revolving around a declarative model-based approach (PuppetLabs, 2012c) which is shown in Figure 3.1. The model starts off by defining the desired configuration state, then simulating and testing the state to ensure correctness and validity, applying the state defined to the nodes selected and enforcing it, then reporting back about the monitoring status,

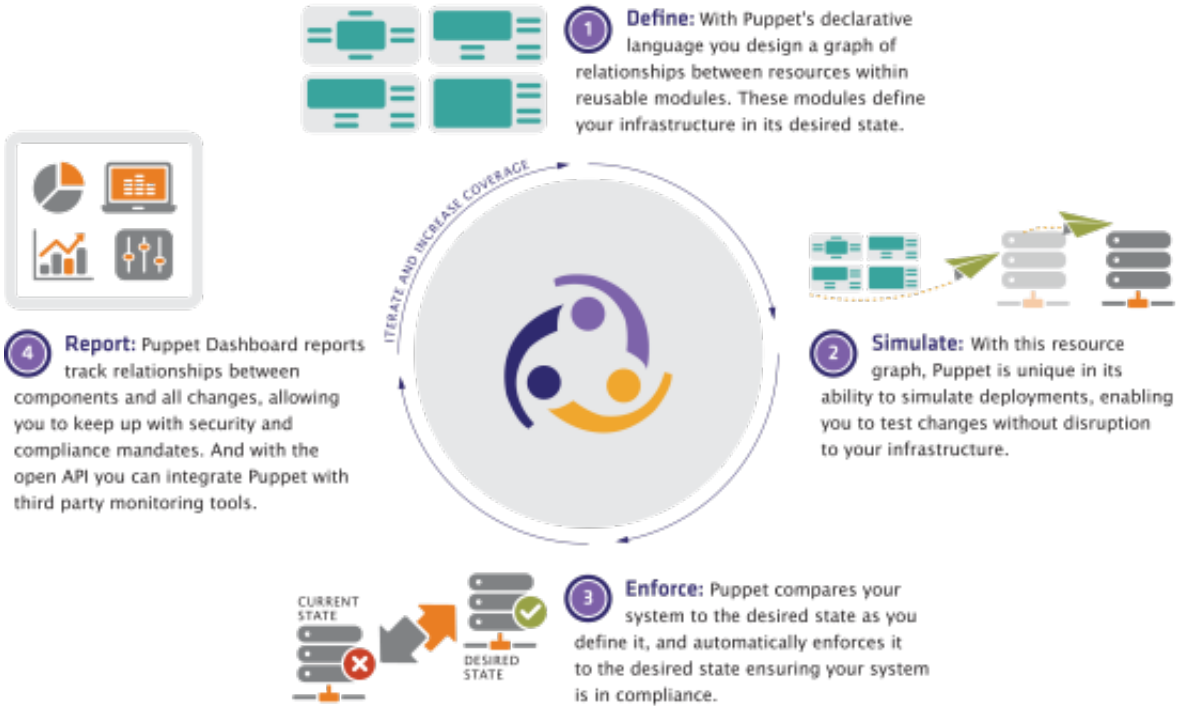


Figure 3.1: The Puppet Model (PuppetLabs, 2012b)

or tracking the ongoing configuration state of the resources to ensure compliance with the predefined state. In Puppet, an administrator would write a manifest of configuration states of a specific package, or a group of packages and test compile it, push it to a repository or sub-versioning server, then pull it into the puppet master node before deploying it to specific groups of nodes. Puppet then monitors the configuration state of the resources on each node using an agent. The Puppet agent talks to the Puppet master to see if there were any configuration changes available or any configuration drift, and apply the correct configuration as required.
In summary, Puppet provides the following critical features of interest to this research:

- Deploying configuration states to specific groups of nodes
- Specifying default configuration states
- Monitoring and detecting configuration state changes

3.1.2 Opscode Chef

Similar to Puppet, Chef is an automation tool, or framework for CM, and is billed as 'Infrastructure for Code' (OpsCode, 2013). Chef uses a code-then-deploy model as shown in Figure 3.2. The administrator would write or 'code' the settings and

turn them into collections called cookbooks, then save to a repository and deploy it to the chef server via a tool called knife. The chef server would then 'compile' the configuration states and deploy them onto the nodes as required. The chef nodes, or chef-client, pulls the configuration state and applies it to the nodes at a fixed interval of time. If there were no changes to the current configuration then this process would do nothing but ensure that the state remains as previously defined, otherwise the configuration would be re-applied to ensure compliance with the preset values.

In summary, Chef provides the following critical features of interest to this research:

- Deploying configuration states to specific groups of nodes
- Specifying default configuration states
- Monitoring and re-deploying configuration state at regular intervals

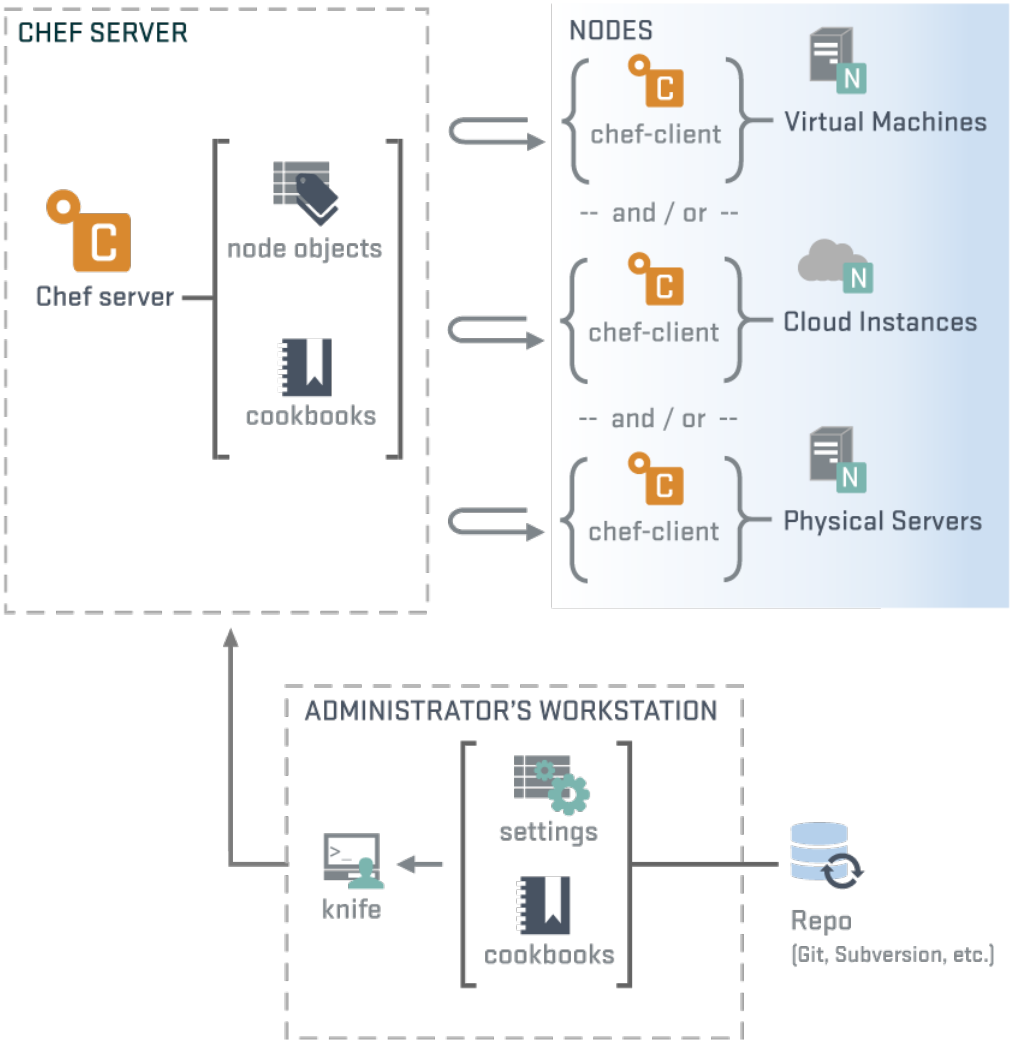


Figure 3.2: The Chef Model (OpsCode, 2013)

3.1.3 Analysis Summary

After analysing the two open-source CM tools available, there are three of critical components of interest that have been identified as follows.

- Deploying configuration states to specific groups of nodes
- Specifying default configuration states
- Monitoring and detecting configuration state changes

3.2 Identifying Components of CM

As discussed in the previous section, the analysis of two open-source CM tools have resulted in three critical components of interest to this research.

3.2.1 Deploying configuration states to specific groups of nodes

A configuration state is a set of configuration defined in code that specifies a specific 'state' that is desired. The ability to deploy single or multiple configuration states (providing that they do not conflict with each other), to specific nodes or groups of nodes as defined is important. For example, a configuration state for the apache web server, along with a configuration state for the mysql database server can be deployed to a group of five nodes in a farm of twenty (turning five of twenty nodes into a web/database server).

3.2.2 Specifying default configuration states

A default configuration state is the overall configuration state that is to be applied by default to a new node that is being added into the pool of resources available or to be managed. For example, when commissioning a new rack of 30 servers and adding them into the pool of resources available to be managed, a default configuration state can be pre-set to be applied to these servers upon being added. This would result in the servers having an identical 'default' configuration state upon being added, without further administrator interaction.

3.2.3 Monitoring and detecting configuration state changes

Configuration state change is defined as when the configuration state on a node has drifted from the defined configuration state last deployed on said node. There should be some monitoring process that will enable detection of such configuration

drifts, and be able to automatically re-apply the last defined configuration state to maintain the compliance of that node to the predefined state.

3.3 Framework Design

Dynamic Configuration Management is not a new field of research, as evidenced in studies done by Kramer and Magee (1985). In their work, they illustrated the difference between a static and dynamic configuration process; which basically resolved itself in differences with iterating through configuration specification versions. After considering the three identified components of CM in the previous section, a proposed framework was designed around a generic model for distributed, cluster-like, dynamic configuration computational environments, where the need to be able to make multiple installations, or deployments of various packages to a large number of computational nodes in an efficient, reliable way is critical. The framework is shown in Figure 3.3 as a process flow chart, and is accompanied by the component interaction diagram in Figure 3.4.

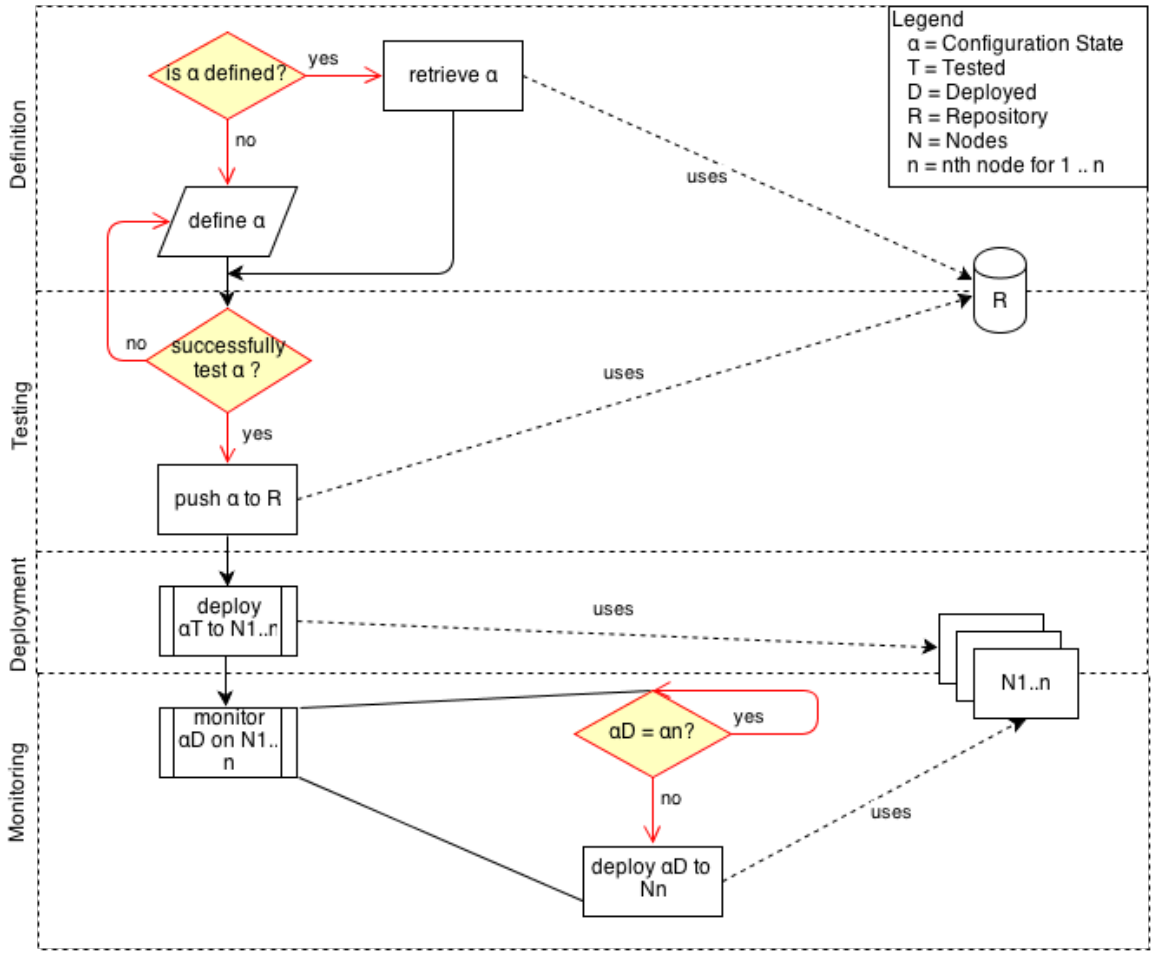


Figure 3.3: Proposed Dynamic Configuration Management Framework

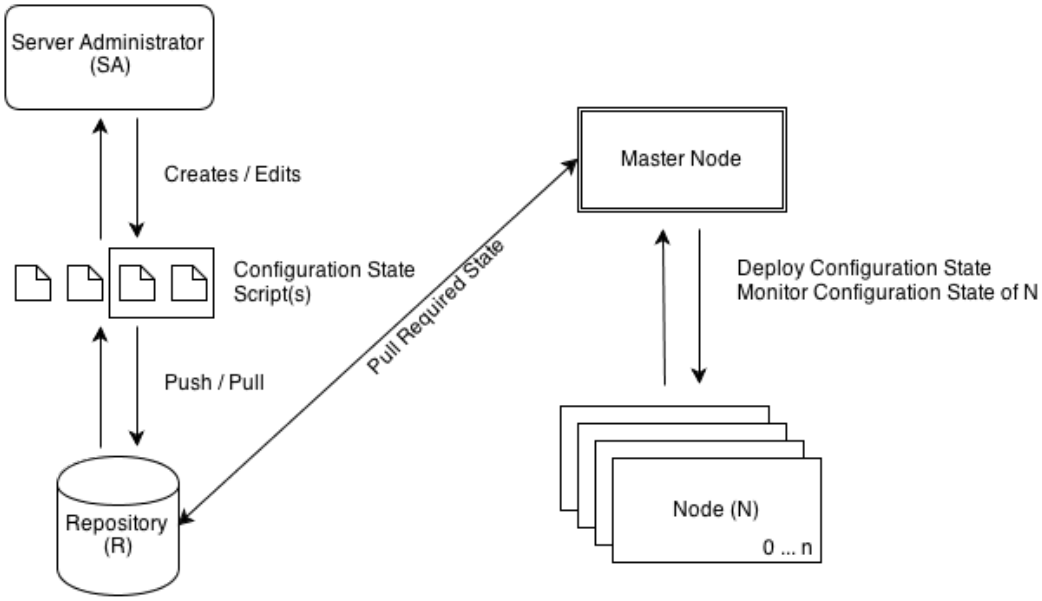


Figure 3.4: Component Interaction Diagram for Proposed Framework

The component diagram shown in Figure 3.4 consists of five main entities, the Server Administrator (SA), the Configuration State (α) scripts, the Repository (R), the Master node, and the numbered (n) Nodes (N). In a standard flow of events, the SA would create or edit Configuration State scripts which can be grouped logically into larger scripts, as indicated by the box on the diagram. The scripts are then pushed or pulled to/from R. The master node would communicate with R to pull the required state for deployment and deploy that state to a preset group of nodes (N_n).

Figure 3.3 shows the proposed framework for dynamic Configuration Management. It's important to note that one of the main differences between static and dynamic Configuration Management as highlighted by Kramer and Magee (1985) is that the fact that the configuration specification, or configuration state iterates and must support incremental changes. It is also important to validate the changes, or test the changes and configuration state before deploying it to the nodes in question.

The proposed framework is divided into four logical processes, Definition, Testing, Deployment and Monitoring, and are illustrated as swim-lanes on the model. The series of processes start by checking if the required state, α is defined. If it is, then it would be retrieved from the repository, R. If not, the SA is required to define α, usually in some form of high level code or language, and then α would have to be tested on a single 'test' node to ensure that no conflicts are present before pushing it to R, and deploying it to the nodes required, N_n. Following which, the process must monitor the configuration states on the nodes N_n, and if the deployed configuration state α_D changes, i.e. α_n != α_D, then the process has to redeploy α_D to the node

N_n with the configuration drift to ensure compliance with the deployed state α_D. The monitoring process is illustrated as a module as it has to run independently of the initial definition process to the deployment process.
For the purposes of this research project, a CM tool mentioned in the previous section, Puppet, will be adopted for the prototype implementation. Puppet is produced by Puppetlabs, and has two versions, an open-source licensed version (Puppet) and a commercially licensed version (Puppet Enterprise).

3.4 Implementation and Evaluation

When considering the CM tool for the prototype implementation, there were two candidates as described in the analysis section previously, which were Chef and Puppet Enterprise. Puppet Enterprise was chosen primarily as it supported up to ten nodes on its free-tier, whereas Chef only provided five nodes on its free-tier license. The open-source version of Puppet was not considered as the commercially licensed version, Puppet Enterprise seemed to provide a larger feature set and support, which is important in an enterprise environment. In the next few sections, the implementation of the prototype using Puppet Enterprise(PE) will be discussed. PE provides a high level descriptive programming language for use in defining customisable configuration 'profiles'. The next chapter will deal heavily in specifics to PE CM with regard to the research implementation for the prototype.

Chapter 4

Research Implementation/ Prototype

A proposed framework was presented in the previous chapter. The framework was designed to be generic enough to be applied to other dynamic distributed environments utilising various other CM tools. This research project will focus on utilising PE as a prototype of the proposed framework. This chapter describes the implementation environment of the prototype using PE, and details the core features of PE being used, followed by a description of the implementation process.

4.1 Environment

This research project utilised the NecTAR Research Cloud located at Monash University, and a server racked at Monash University as the environment. The NecTAR Research Cloud is a national research cloud, consisting of a partnership of various research universities in Australia to develop and operate cloud infrastructure and services running OpenStack and KVM cloud middleware. It consists of seven institutions forming up node 'farms'; University of Melbourne, Australian National University, Monash University, Queensland Cyber Infrastructure, University of Tasmania, iVEC and eResearch South Australia.

4.1.1 Repository

The Repository is deployed on the master node using the standard unix svn (subversion) package. To reduce latency and increase transfer speeds, especially with larger files, it is of benefit to locate the Repository on the same master node so that read/write will be local to the master node. As shown in Figure 3.3, the Server Administrator (SA) creates configuration scripts (α) locally or remotely on a workstation, then tests and pushes it to the Repository. When the SA wants to deploy α

to the nodes via the master node, he or she would need to pull it to the master node, and in order to speed up this process and to reduce as much transfer time overhead as possible, it is of benefit for the Repository to be either on the local network to the master node, or on the same server. In the case of this prototype, I have elected to locate it on the same server due to resource and budget constraints. As mentioned previously, the Repository in this prototype is a standard svn package that allows version control. The package tracks changes across files and folders, providing an easy way to roll-back changes and to manage multiple versions and copies of the same folder or file set. In the case of configuration scripts in the context of dynamic Configuration Management, it is important to be able to support multiple versions of the same script as software packages may contain different configuration states depending on the nature of the job to be run on the computational nodes.

4.1.2 Master Node

The Master Node is the main node that controls and provides the monitoring and management of the distributed compute nodes. It should be able to deploy configuration state scripts to a preset group of compute nodes, and be able to monitor them for changes. In a nutshell, it is the core of the system. Since it is the core of the system, there should exist some redundancy in order to ensure optimal uptime. The master node should also be a powerful server that is capable of fast complex database operations with a good network connection to improve transfer throughput to the compute nodes. The specifications of the master node is listed in Table 4.1. In the following section, the possibility of redundant master nodes will be discussed in the context of Puppet Enterprise.

Type	Description
CPU	Dual Intel Xeon E5506 2.13GHz Quad-Core Processor
RAM	12GB
OS	CentOS 6.4 (Red Hat Enterprise Linux) Kernel 2.6.32
HDD	500GB
Network	1000baseT

Table 4.1: Master Node Specification

4.1.3 Compute Nodes

The Compute Nodes are the nodes that are used for computation. In the case of this research project, I am utilising nodes provisioned from the NecTAR Research Cloud for testing purposes. Due to budget and allocation restrictions, I could only

provision nine m1.small configured nodes. The specifications of the compute nodes provisioned are listed in Table 4.2.

Type	Description
CPU	AMD Opteron 6234 2.4GHz Single-Core Processor
RAM	4GB
OS	CentOS 6.4 (Red Hat Enterprise Linux) Kernel 2.6.32
HDD	40GB
Network	100baseT

Table 4.2: Compute Node Specification

4.2 Puppet Enterprise

Puppet Enterprise (PE) is the commercial version of the free-license open-source Puppet provided by Puppetlabs. It provides more functionality as compared to the open-source Puppet version, targeted at enterprise environments. These additional functionality include task automation, simplified upgrading and maintenance, a graphical user interface, role-based access control, and greater provisioning support as well as configuration management discovery tools (PuppetLabs, 2012a). A deployment diagram of the various roles as installed in the prototype is shown in Figure 4.1.

4.2.1 Roles

PE consists of four roles; Puppet Master, Console, Cloud Provisioner and Puppet Agent.

Puppet Master

The Puppet Master is installed on the master node, and provides the compilation and distribution of configuration states, or catalogs to the puppet agents installed on all the nodes being managed. The configuration catalogs describe the state of resources that have been configured explicitly to exist or not exist on the nodes and consist of manifests defined by the administrator that are identified as classes. Each class corresponds to a manifest, and each manifest can consist of multiple classes. Each class can define a specific set or group of resources, or packages to be installed with specific configuration files. As mentioned in the previous section, PE supports puppet master redundancy by allowing multiple redundant Puppet Masters in the deployment, however this requires further configuration and will not be explored

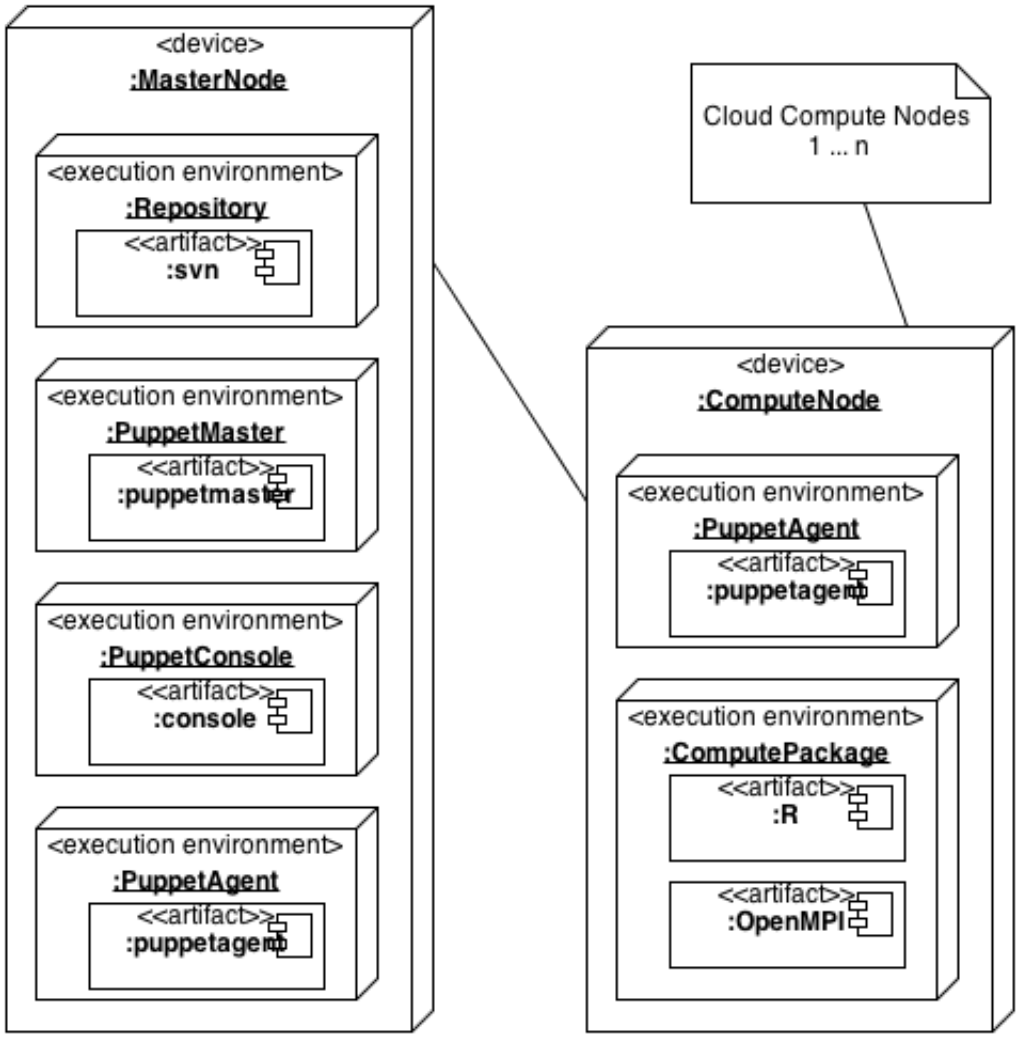


Figure 4.1: Prototype Deployment Diagram

in this prototype. Similarly, there exists the capability to tweak the security being used in the panel login to the Puppet Master / PE Dashboard on the master node, although PE supports other types of security such as LDAP and Kerberos, for the purposes of this prototype, I am using the default HTTPS/SSL configuration. The communication between the Puppet Master and the associated Puppet Agents utilises HTTPS with certificates over a host-verified SSL channel in a Representational State Transfer-like manner (REST). Again, due to scope restrictions, this area of the prototype will not be explored in this research.

Console

The Console is to be installed on a single secure node, and can be installed on the master node. It provides the web console interface that allows resource editing on the nodes, advanced tasks such as triggering Puppet Agent runs, grouping and assigning classes to nodes and viewing reports, charts and inventory changes. It also

provides for approval and rejecting of audited changes. The console collects reports and serves node information to the Puppet Master. In this prototype, the Console will be installed on the master node.

Cloud Provisioner

The Cloud Provisioner is an optional role that can be installed if VMware or Amazon EC2 cloud instances are being used. It can be used to provision new instances and install PE on them, as well as adding the nodes to a group on the console. It has to be installed on a single secure node as it contains confidential information about the accounts. This prototype does not include the use of the Cloud Provisioner role.

Puppet Agent

The Puppet Agent is installed on every single node in the deployment environment. It runs a daemon that pulls and applies configurations provided by the Puppet Master and reports on changes to configurations and resources. It communicates with the Puppet Master via HTTPS using certificates over a host-verified SSL channel.

4.2.2 Language

PE provides its own high level descriptive language to create manifests and classes. A manifest consists of a class definition which contains one or more resource declarations. A resource declaration is shown in Listing 4.1, and contains a resource type *file*, the name of the file *ssh_config*, a path */etc/ssh/ssh_config*, and the source or template for which to copy the file from.

```
file { 'sshd_config':
  path => '/etc/ssh/sshd_config',
  ensure => file,
  mode   => 600,
  owner => root,
  source => 'puppet:///modules/sshd/sshd_config',
}
```

Listing 4.1: ssh_config

Classes also support usage of regular expressions, conditional statements such as if statements, case statements, and programming methods such as overriding and defaults.

Each manifest has a fixed file structure to use, and each manifest has a folder named after the manifest class name. Within that folder, would have at least one main folder, the manifest folder. In most cases, there can be a files folder as well. The manifest folder would contain at minimum the *init.pp* manifest file and the latter

files folder serves as the root file storage for each manifest. To utilise the files contained within, the path *puppet:///modules/* is appended to the folder path from the manifest root folder, for example *puppet:///modules/sshd/sshd_config* would refer to the file *sshd_config* in the files folder. An example of a manifest for *openssh-server* is shown in Listing 4.2.

PE compiles the various manifests defined for a node group, into a catalog and applies it to the nodes in a node group to form a defined system configuration state, as shown in Figure 4.2.

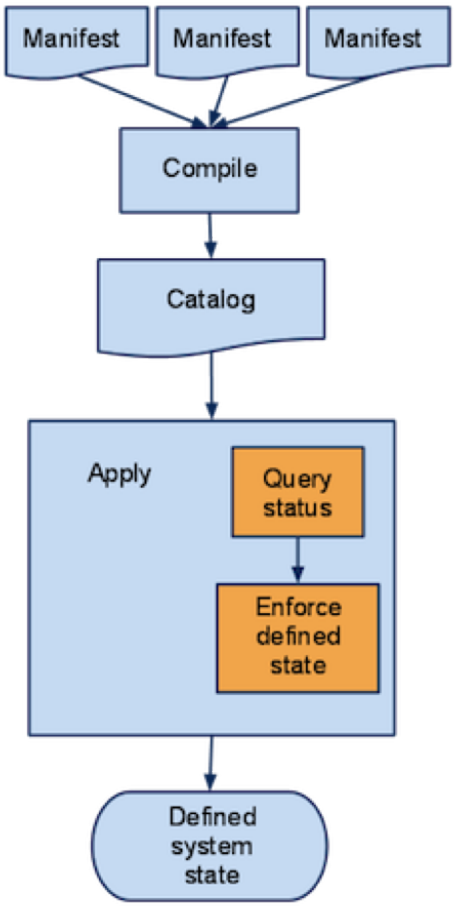


Figure 4.2: PE Compilation State Diagram

4.2.3 Developing Components

Application packages used in HPC applications can be divided into two main types; the monolithic and the modular type. In this research project, we have identified two major packages that are in use for HPC applications across eResearch, one of each type of package. To briefly describe an example of the two types of packages, the first type of package, the monolithic type which usually consists of a fixed set of

components, can be represented by OpenMPI. The second type, the modular type are packages that contain varying components and is represented by R. Both are packages commonly used in HPC applications, and further elaboration of the two is provided in the following subsections. Traditional Configuration Management covers static configuration states, where packages generally have a single base configuration state per version. The main striking difference with modular packages when compared to monolithic, is that modular have varying configuration states that change depending on the required components of the package to be installed in question.

```
class openssh-enable {
    package { 'openssh-server':
      ensure => present,
      before => [ File['/etc/ssh/sshd_config'], File['/home/ec2-user/.
          ssh/authorized_keys'] ],
    }
    file { '/etc/ssh/sshd_config':
      ensure => file,
      mode   => 600,
      source => 'puppet:///modules/sshd/sshd_config',
    }
    file { '/home/ec2-user/.ssh/authorized_keys':
      ensure => file,
      owner => ec2-user,
      group => ec2-user,
      mode   => 600,
      source => 'puppet:///modules/sshd/authorized_keys',
    }
    service { 'sshd':
      ensure     => running,
      enable     => true,
      hasrestart => true,
      hasstatus  => true,
      subscribe  => File['/etc/ssh/sshd_config'],
    }
}
```

Listing 4.2: openssh-enable

```
class openssh-disable {
    package { 'openssh-server':
      ensure => absent,
    }
}
```

Listing 4.3: openssh-disable

In PE, to address the issue of multiple preconfigured configuration states, it is possible to create a manifest or group of manifests with classes specifying each component within the main package. Since in PE the naming of the manifests is not fixed, meaning you can create varying sets of manifests with version names in the manifest class name. For example, Listing 4.2 and Listing 4.3 shows two short manifests with varying configurations for the same package, openssh-server. Listing 4.2 shows a standard openssh-server manifest with a predefined *authorized_keys* and *sshd_config* configuration file. Listing 4.3 shows a openssh-server manifest to ensure that it isn't installed.

OpenMPI

OpenMPI is an open source Message Passing Interface Library commonly used in High Performance Computing applications. It is available to most platforms, and is most commonly used on the Unix platform. It is monolithic in nature and is also available on most general release Unix public repositories. Being monolithic, there generally is one singular configuration state for each release version, which simplifies the management of the OpenMPI package.

R

R is an open-source environment for statistical computing and graphics, and is available on multiple platforms, including Unix, Windows and Mac OS. It is a popular package that is considered an alternative for MATLAB, a commercially licensed environment. It is relatively modular in nature, and can be installed in multiple configurations with varying options and components enabled or disabled. R allows for a large variety of statistical and graphical techniques and is highly extendable. R consists of eight basic packages supplied with the basic distribution and there are many more available via CRAN sites. As mentioned previously, since modular packages can contain varying configured components and not all would be common to every job run on a HPC environment, there exists a need to have multiple preconfigured configuration states, and the ability to manage and maintain said states, as described in the Dynamic Configuration Management model in the previous chapter.

4.3 Implementation

Finally, to briefly describe the implementation process, the following four sections will outline the processes taken while installing and preparing the prototype for evaluation.

4.3.1 Deploying Puppet Enterprise

```
q_install=y
q_puppet_cloud_install=n
q_puppet_enterpriseconsole_auth_database_name=console_auth
q_puppet_enterpriseconsole_auth_database_password=tdZcugNSEg8CRkHrqBnJ
q_puppet_enterpriseconsole_auth_database_user=console_auth
q_puppet_enterpriseconsole_auth_password='CDOmnde3342kjnsdf9'
q_puppet_enterpriseconsole_auth_user_email=t[redacted]
q_puppet_enterpriseconsole_database_install=n
q_puppet_enterpriseconsole_database_name=console
q_puppet_enterpriseconsole_database_password=AahdSDVasaa5ZR9rQiRepQjV
q_puppet_enterpriseconsole_database_remote=n
q_puppet_enterpriseconsole_database_root_password='dkjdgaisd8ga9dsfW&3
    sd'
q_puppet_enterpriseconsole_database_user=console
q_puppet_enterpriseconsole_httpd_port=443
q_puppet_enterpriseconsole_install=y
q_puppet_enterpriseconsole_inventory_hostname=s-jeff-h627.infotech.
    monash.edu.au
q_puppet_enterpriseconsole_inventory_port=8140
q_puppet_enterpriseconsole_master_hostname=s-jeff-h627.infotech.monash.
    edu.au
q_puppet_enterpriseconsole_setup_db=y
q_puppet_enterpriseconsole_smtp_host=smtp.gmail.com
q_puppet_enterpriseconsole_smtp_password=
q_puppet_enterpriseconsole_smtp_port=25
q_puppet_enterpriseconsole_smtp_use_tls=n
q_puppet_enterpriseconsole_smtp_user_auth=n
q_puppet_enterpriseconsole_smtp_username=
q_puppet_symlinks_install=y
q_puppetagent_certname=s-jeff-h627.infotech.monash.edu.au
q_puppetagent_install=y
q_puppetagent_server=s-jeff-h627.infotech.monash.edu.au
q_puppetca_install=y
q_puppetmaster_certname=s-jeff-h627.infotech.monash.edu.au
q_puppetmaster_dnsaltnames=s-jeff-h627.infotech.monash.edu.au
    ,130.194.70.59
q_puppetmaster_enterpriseconsole_hostname=localhost
q_puppetmaster_enterpriseconsole_port=443
q_puppetmaster_install=y
q_vendor_packages_install=y
q_verify_packages=y
```

Listing 4.4: PE Answer file

Puppet Enterprise (PE) is provided as a compressed tar folder by Puppetlabs, and its installation process can be automated by supplying an answer file as shown

in Listing 4.4, and calling the installation program with parameters, as seen in Listing 4.5. The answer file is shown in Listing 4.4 installs the Puppet Master, Puppet Agent, Puppet Console roles onto the master node.

The installation of PE was straightforward, with strict forward and reverse DNS lookup requirements on all the nodes. PE requires both apache web server and the mysql database server to be preinstalled and configured.

```
/root/puppet-enterprise-2.8.1-el-6-x86_64/puppet-enterprise-installer -
    a agent.answer
```

Listing 4.5: PE Installation

4.3.2 Deploying Nodes and Puppet Agent

The nodes were deployed using the NecTAR Research Cloud's Amazon EC2 API, and a post installation script shown in Listing 4.6 was utilised to automate the installation of the packages required. As the nodes were deployed with a base image of CentOS 6.4 with no additional group packages configured, it was necessary to install a few package groups to ensure that there were no missing required components. The script also created the answer file for PE and called the installation automatically. Once the script had finished execution, the administrator is required to login to the PE dashboard to authorise the certificate for the newly generated agent, as seen in Figure 4.3.

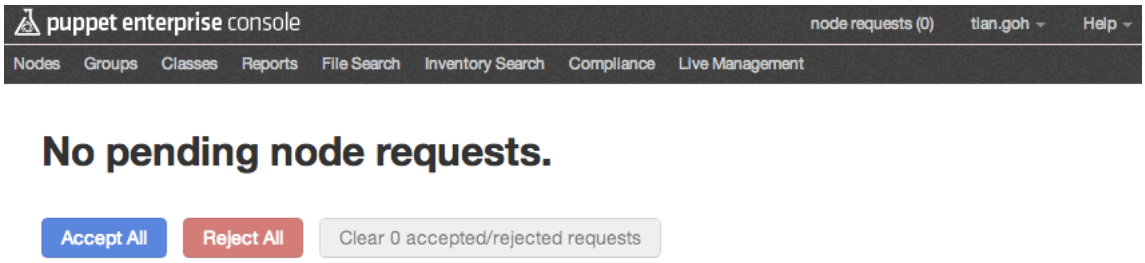


Figure 4.3: PE Authorise Agent

```
#!/bin/bash
cd /root
wget https://s3.amazonaws.com/pe-builds/released/2.8.1/puppet-
    enterprise-2.8.1-el-6-x86_64.tar.gz
tar -zxf puppet-enterprise-2.8.1-el-6-x86_64.tar.gz
yum update -y
yum groupinstall "Development Tools" -y
yum groupinstall "Networking Tools" -y
yum groupinstall "Server Platform" -y
yum install bind-utils -y
ip=`ifconfig eth0 | grep "inet addr" | cut -d\t -f 2 | cut -d" " -f 2 |
     sed s/addr://`
host=`nslookup $ip | grep "name =" | cut -f 2 | cut -d" " -f 3 | sed s
    /.$// `
hostname $host
node=node6
touch agent.answer
echo q_fail_on_unsuccessful_master_lookup=y >> agent.answer
echo q_install=y >> agent.answer
echo q_puppet_cloud_install=n >> agent.answer
echo q_puppet_enterpriseconsole_install=n >> agent.answer
echo q_puppet_symlinks_install=y >> agent.answer
echo q_puppetagent_certname=$(echo $node) >> agent.answer
echo q_puppetagent_install=y >> agent.answer
echo q_puppetagent_server=s-jeff-h627.infotech.monash.edu.au >> agent.
    answer
echo q_puppetca_install=n >> agent.answer
echo q_puppetmaster_install=n >> agent.answer
echo q_vendor_packages_install=y >> agent.answer
echo q_verify_packages=y >> agent.answer
/root/puppet-enterprise-2.8.1-el-6-x86_64/puppet-enterprise-installer -
    a agent.answer
/opt/puppet/bin/puppet agent --test
```

Listing 4.6: Automated Installation

4.3.3 Configuring Node Groups

Groups of nodes can be configured on the PE dashboard by adding a group, and adding nodes to them, as seen in Figure 4.4.

4.3.4 Deploying Packages

Packages to be deployed would have to conform to the folder structure and be located in */etc/puppetlabs/puppet/modules/*. Once it is defined, the class, or manifest then can be added to PE via the dashboard as seen in Figure 4.5. After which the class

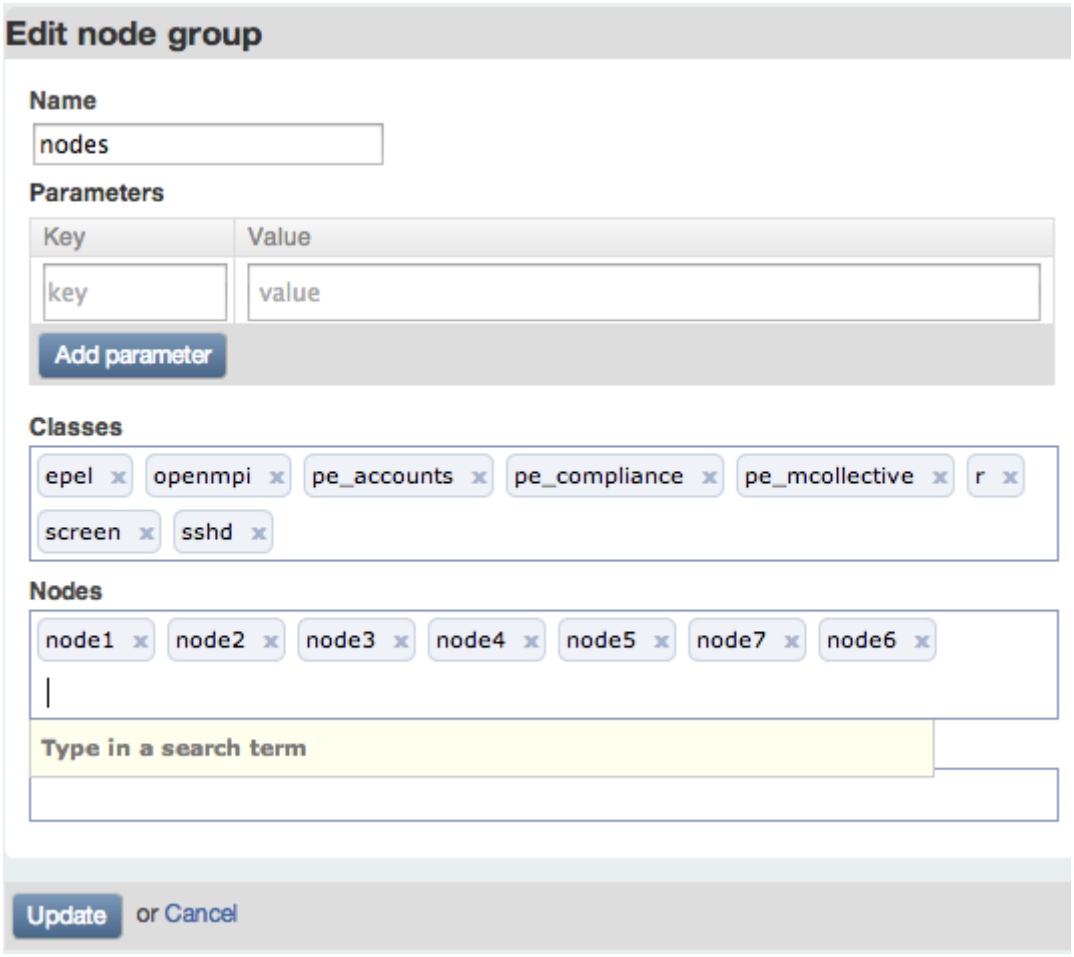


Figure 4.4: PE Configure Node Group

can be added to node groups, which basically tells PE to use the manifest when compiling the configuration state for the nodes in the node group, which is shown in Figure 4.4. The node group can be considered a configuration state that contains specific sets of manifests or classes, and in either case it is straightforward to add and remove both classes and nodes from the node groups.

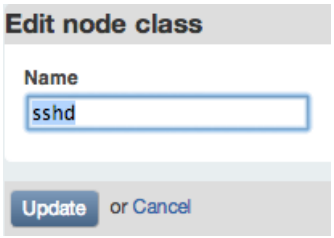


Figure 4.5: PE Add Class

Chapter 5

Research Evaluation

This chapter will discuss the evaluation of the framework and prototype, along with its associated achievements, limitations and difficulties encountered during the research prototyping.
This research has produced a framework for dynamic configuration management, and the prototype built was used to validate that the framework was correct. At the same time, the prototype using the Puppet Enterprise CM Tool is also evaluated to demonstrate the improvement in the criterions of Efficiency and Reliability. In order to evaluate the prototype, it is required to first specify the setting and scenarios to be run for the prototype evaluation experiments, which will be addressed in the following section.

5.1 Settings and Scenarios

In evaluating the prototype, three scenarios across two main deployment settings have been drawn up for use. The two deployment settings are as follows:

- Configuration of newly deployed nodes (New Nodes)
- Re-configuration of existing nodes (Existing Nodes)

These two settings are two of the most common settings that would happen within a deployment environment. And similarly, the three scenarios are as follows:

- Manual Configuration by administrator (Manual Configuration)
- Semi-automated Configuration by administrator using scripts (Scripted Configuration)
- Automated Configuration using the prototype (Automated Configuration)

The three scenarios listed above are the three probable scenarios to happen in a deployment environment. To determine the two criterions, it is first needed to define the exact measure for how they would be weighed, and this is shown in Table 5.1.

Criteria	Measure
Efficiency	Time Taken in seconds
Reliability	Percentage similarity of nodes determined by PE's resource inspection tool

Table 5.1: Evaluation Criteria

The reliability criteria is determined by PE's resource inspection tool, which determines the similarity of nodes inspected on a whole, and for specific packages. The graphical interface of the resource inspection tool, and the results from the evaluation of the reliability criteria are dependant of the package resource value, which is third on the summary list as shown in Figure 5.1.

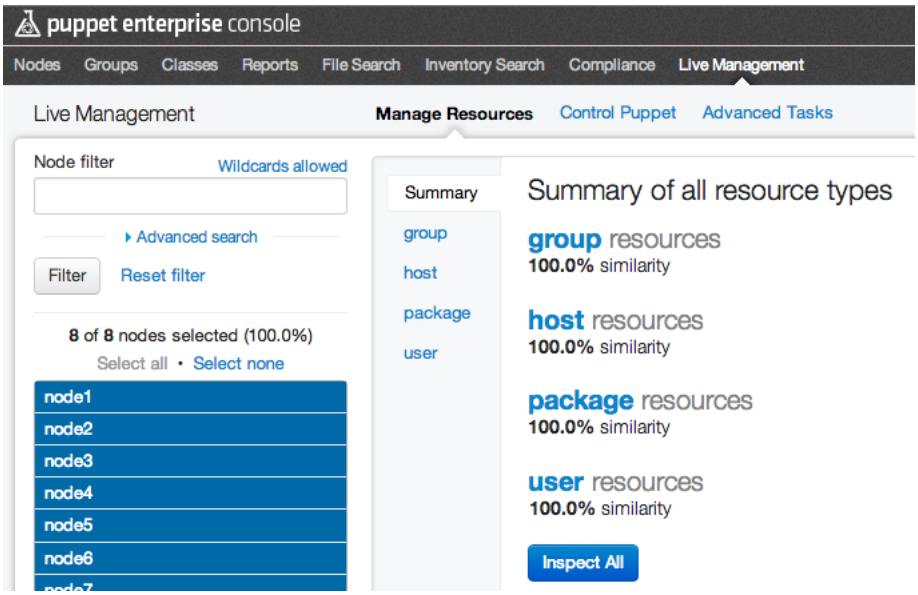


Figure 5.1: PE Resource Inspection Tool

Deployment Setting	Package Name	Version	Source
New Node	OpenMPI	1.5.4-1	CentOS Repository
New Node	R	3.0.1-2	EPEL Repository
Existing Node	OpenMPI	1.5.4-1	CentOS Repository
Existing Node	R	2.13.0-2	RepoForge Repository

Table 5.2: Evaluation Package Versions

The evaluation process basically entails installing the openmpi and R packages from the official repositories, and in the case of the existing nodes, the un-installation of the previously installed versions would be conducted before the re-installation process. The software versions to be installed are listed in Table 5.2.
The efficiency criteria evaluation will be determined by time using the unix timestamp in seconds and nanoseconds, and the difference subtracted to determine the time taken, calculated using the code as shown in Listing 5.1. The accuracy of the timestamp will be to two nanoseconds.
The Scripted Configuration process scripts are available in Appendix B as Listings B.1 and B.2. The Manual Configuration process follows the scripts, with the exception that it is manually entered. The timings will be then collected in a predefined file on each of the nodes, and then compiled.

```
#!/bin/bash
# Efficiency Evaluation code
# Note: time captured is in format of seconds and nanoseconds as a
    single string and is processed into a 12 digit timestamp in unix
    time with an accuracy of two nanoseconds
# capture start time into variable start
start=‘date +%s%N | cut -b1-12‘
# capture end time into variable end
end=‘date +%s%N | cut -b1-12‘
# calculate time taken by deducting end time from start time
timetaken=‘expr $end - $start‘
# record the calculated time taken to file
echo $timetaken > /tmp/tdata
```

Listing 5.1: Efficiency Evaluation Source Code

In order to achieve a good statistical sample size, and to reduce the bias of speed of the network on the evaluation process, the evaluation tests were run a total of ten times for each scenario and setting, resulting in a sample size of ten sets of two settings for three scenarios, which is a sample of 80 for each row of finalised results. As mentioned previously, the Cloud computing paradigm is geographically diverse and as such, the Cloud nodes are not located on the same server and could be located at different data centres. Over the space of a regular day, the speed of the network relies on the a few criteria, which are basically the original intended throughput, as well as the current traffic utilisation. To reduce the bias of the network speed, the evaluation process is run at intervals of 4.8 hours across 48 hours, or two days, on Sunday and Monday, as shown in Table 5.3. The two days were specifically chosen to determine if there were differences across a normal work-day and a weekend, or rest-day. Finally, due to resource limitations, and to restrict the scope of configuration

and the complexity of the task, the evaluation is limited to eight nodes and the two packages as described in a previous chapter, OpenMPI and R.

Day	Time	Evaluation Sequence No
Sunday	00:00 Midnight	01
Sunday	04:48 Morning	02
Sunday	09:36 Morning	03
Sunday	14:24 Afternoon	04
Sunday	19:12 Evening	05
Monday	00:00 Midnight	06
Monday	04:48 Morning	07
Monday	09:36 Morning	08
Monday	14:24 Afternoon	09
Monday	19:12 Evening	10

Table 5.3: Prototype Evaluation Schedule

5.2 Prototype Evaluation Results

This section will briefly describe the data obtained after the evaluation of the prototype according to the three scenarios and two settings, for a total of two main sets of data. The results discussion will be broken into two sections, the prototype evaluation, followed by a short discussion of accuracy of the framework against the Puppet Enterprise model used in the prototype.

The evaluation of the prototype has produced the results as shown in Table 5.4. The data collected was collated into a spreadsheet in the Microsoft Excel program. The results analysis was then done in the Apple iWork Numbers program to generate the data graphs shown in Figure 5.2 and Figure 5.3. The finalised efficiency results were calculated by first averaging the each set of sequence run for the eight nodes for each of the ten sequence runs, then averaging the calculated result of the ten sequence runs, for each scenario and setting (a total of six). Similarly, the finalised reliability result was calculated by averaging the set of ten sequence run reliability raw data for each scenario and setting to obtain the finalised reliability average. The raw data as seen in Appendix C as Tables C.3, C.4, C.5, C.7, C.8 and C.9 are the data collected from the evaluation process. Each data row of the Tables C.2 and C.6 represent the averaged results from one evaluation test iteration for a specific scenario and setting. A summary table was then generated, as seen in Table 5.4, which describes the averaged result for the six scenario and setting combinations.

Observing the results in Table 5.4, it is interesting to note that in terms of reliability, as seen in Figure 5.3, the similarity of the nodes configured manually on

Setting	Scenario	Efficiency (Avg)	Reliability (Avg)
New Nodes	Manual Configuration	942.18s	92%
New Nodes	Scripted Configuration	801.57s	97%
New Nodes	Automated Configuration	186.28s	100%
Existing Nodes	Manual Configuration	1055.73s	89%
Existing Nodes	Scripted Configuration	872.15s	98%
Existing Nodes	Automated Configuration	192.31s	100%

Table 5.4: Prototype Evaluation Results

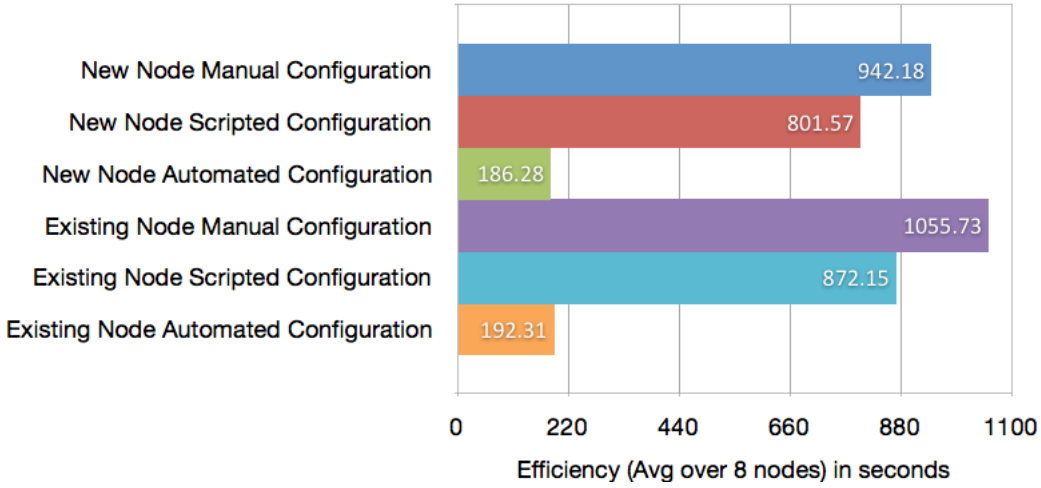


Figure 5.2: Efficiency Result Graph

existing nodes, when compared to the new nodes actually decreased from 92% to 89%. After investigation, I discovered that this was due to a two part reason: firstly, as the nodes are in the cloud, the repositories that the system was retrieving packages from had different minor versions, which lead to the difference in configuration, and secondly, when re-configuring the existing nodes and removing packages, it was easy to miss removing some files as the process was done manually multiple times. It is noted that when the process was repeated with scripting, the reliability increased in comparison to the manual process, which can be attributed to running the same script across all the nodes; hence reducing the chance of mistakes or errors since the script would have been tested before deployment. The reliability results also demonstrate the difference with using Clouds and other fixed infrastructure in the sense that the Cloud paradigm revolves around geographically diverse computational nodes and instances. The instances are not always located in the same data center, or even in the same country, as with the repositories which usually consist of geolocated servers, meaning that depending on your geographical location, you might be using a different repository. The differences will at best contain minor version differences, and in the case where the user requires custom configuration files,

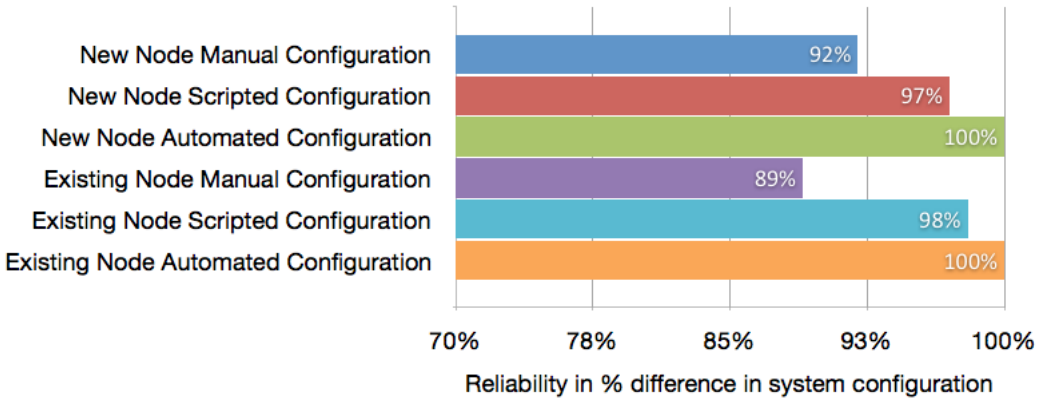


Figure 5.3: Reliability Result Graph

PE has the ability to deploy the same configuration file(s) that the user has created, hence ensuring that the nodes deployed contain the same configuration. Finally, when looking at the final automated process for the new and existing nodes, the similarity result returned 100% on both cases, proving that the CM Tool does the best job of ensuring compliance and similarity across multiple nodes.

The efficiency measure also solidly demonstrates the time savings that using a CM Tool such as PE provides. When comparing the results for manual and scripted configuration versus automated configuration process, there is a huge increase of 77% and 81% in efficiency for all the cases when using an automated process. This is also attributed to the difference in runtime, that is to say that in the scripted process, the nodes were configured in serial and consecutively instead of in parallel which is what the automated process did. This resulted in a large time saving or increase in efficiency.

Finally, as discussed briefly in the previous section, the evaluation process was run ten times, partially to reduce the possible bias that the current usage and load of the network would have on the evaluation results. Based on the raw data processed for standard deviation figures in Appendix C over Tables C.10 and C.11, the sample sets shown in Table 5.5 demonstrate a standard deviation of between 0.002 and 0.691 for the new nodes, and 0.013 and 0.309 for the existing nodes using the manual configuration process. Similarly, the automated configuration process for the new and existing nodes resulted in a standard deviation between 0.084 and 0.524, and 0.013 and 0.309 respectively. The values for the automated configuration process clock in at between 0.009 and 0.386 for the new nodes, and 0.002 and 0.068 for the existing nodes. While the results demonstrate that there is a lesser region of deviation using the automated process, it can be seen that the network performance, or current load and speed actually affects the configuration and package deployment process to a

small extent. With the first two types, the manual and scripted configuration processes, part of the deviation values can be explained by the fact that the tests were conducted manually to some extent, whereas the automated configuration was done solely by machine. It can be then concluded that the evaluation for efficiency did, to a small extent be affected by the network load or speed at the point of execution.

Setting	Scenario	Min Efficiency Std. Dev.	Max Efficiency Std. Dev.
New Node	Manual Configuration	0.002	0.691
New Node	Scripted Configuration	0.084	0.524
New Node	Automated Configuration	0.009	0.386
Existing Node	Manual Configuration	0.013	0.309
Existing Node	Scripted Configuration	0.006	0.118
Existing Node	Automated Configuration	0.002	0.068

Table 5.5: Standard Deviation of Efficiency between Evaluation Sequences

5.3 Framework Evaluation Results

Based on the prototype, the activities carried out using PE against the framework components is provided in Table 5.6. The framework describes a list of activities that should be carried out in as part of a logical process slow, as seen previously in Figure 3.3. Hence, in evaluating the framework, the seven main activities being carried out as part of the four processes identified in the framework are listed, and compared against the activities that can be carried out in the prototype deployed. The evaluation results in Table 5.6 demonstrate that the prototype using PE fits the framework very well. PE allows naming of manifests and classes in a version-like manner, which includes examples such as *openssh-server_1.2.1-enable*, *openssh-server_1.2.1-disable* and *openssh-server_1.5.0-enable*. In this way, the sub-versioning server can tie in with the manifests as it allows users to fork a current configuration to a new folder and name it differently, which also would also be reflected in the manifests directory. Furthermore, with PE there are two ways to deploy the configuration state to nodes, which are namely by command line and by using the graphical interface provided. The availability to deploy the configuration states by command line allow the development of complementary tools that would further automate the process of preparing the computational environment for a queued job. That being said, it depends on the job queue management software, and the process that the entire environment revolves around. The environment could be set up dynamically by some queue management software, or could be configured by an administrator

prior to the deployment of nodes for a new job. In either case, the benefit of having a predefined process flow for dynamic configuration management is present.

Process	Framework Activity	PE Activity	Fit (Yes/No)
Definition	is α defined?	check if α is defined in R	Yes
Definition	if α defined, retrieve from R	found α in R, pull α from R	Yes
Definition	if α not defined, define α	SA defines α	Yes
Testing	test of α successful?	SA tests α using test run of puppet agent	Yes
Testing	push α to R	SA commits new version in svn	Yes
Deployment	deploy α_T to $N_1..n$	SA deploy α_T to node group	Yes
Monitoring	monitor α_D on $N_1..n$	PE monitors node group for configuration drift and reapplies α_D as required	Yes
Overall Fit			Yes
Legend	α = configuration state R = Repository N = Node $_1..n$ = for 1 to n $_T$ = Tested $_D$ = Deployed		

Table 5.6: Puppet Enterprise and Framework Comparison

Chapter 6

Conclusion

This chapter summarises the research work carried out and its corresponding significance. The achievements and contributions, along with the limitations, difficulties and problems encountered are discussed, following which some directions for future works are listed.

6.1 Summary

This thesis has identified a gap in current research literature with regard to Configuration Management for High Performance Computing on the Cloud platform. Furthermore, there has been little to no research actively being done within the last decade on the issue of Dynamic Configuration Management for the current distributed Cloud platforms. Based on that, this thesis has contributed a short analysis of two existing Configuration Management tools in identifying three components of Configuration Management of interest to this field, which are in effect, the deployment, specification, monitoring of configuration states to specific groups of computational nodes, and detecting changes or configuration drift on these configuration states. A framework was designed, building on past research (Kramer and Magee, 1985) and the three components identified, as a Dynamic Configuration Management Framework. Following which, the framework was evaluated using a prototype built using Puppet Enterprise, and deployed using a server at Monash University, and the NecTAR Research Cloud.

In evaluating of the prototype in terms of the efficiency and reliability criterions, a set of three scenarios across two identified deployment settings was used during the evaluation process that spanned two days and run at intervals of 4.8 hours for a total of ten sequence runs, and the results demonstrate the dramatic increase in efficiency of 77% and 81% as compared to manual and scripted configuration processes. The network latency during evaluation was determined to have a small impact on the evaluation process. Furthermore, the results show that the reliability of the entire

deployment environment in terms of compliance to the configuration state has increased from 89% and 92% for manual configuration processes to 100% when using an automated configuration process on the prototype. A short breakdown of the activity processes of the prototype was matched against the proposed framework, and found to be a direct fit.
In conclusion, the results of the evaluations conducted validate the framework proposed, and have demonstrated a direct improvement when compared to traditional Configuration Management processes. The prototype was evaluated as being successful, although further work grounding the framework and data by deploying other Configuration Management tools would be beneficial, as in demonstrating the scalability of the prototype by increasing the amount of computational nodes managed.

6.2 Achievements and Contributions

In completion of this research work, some contributions and achievements were attained, and are as follows:

- A framework of Dynamic Configuration Management was proposed, and evaluated using a prototype

- A prototype using Puppet Enterprise was created and deployed to demonstrate the improvement in efficiency and reliability in using such tools for Configuration Management

- An efficiency increase of 77% to 81% was achieved when using an automated Configuration Management process.

- A reliability increase to 100% compliance / similarity was achieved when using an automated Configuration Management process.

- Similarly, as this thesis is applied research, it will assist decision makers in the taking up and deployment of similar Configuration Management tools, as the results from the evaluation of the prototype is conclusive.

- Finally, as a relatively *novel* work in this specific field of Configuration Management for High Performance Computing on Clouds, this thesis has contributed towards future research in the field, and should be used as a basis and improvement of previous attempts to describe Dynamic Configuration Management.

6.3 Limitations

There have been some challenges that were out of scope and could not be addressed in this thesis. Similarly, there were some constraints placed on the research work in order to limit the scope of the project. These limitations are listed as follows:

- In evaluating the framework, a decision was made to only concentrate on deploying Puppet Enterprise as a prototype as it would double the time required if a dual deployment was made using both Puppet Enterprise and Chef.

- Chef was not used in this research evaluation due to the lower amount of node restrictions as compared to Puppet Enterprise, which was a free-tier level of five nodes versus ten nodes.

- Due to budget constraints and allocation constraints, the project was allowed access to seven NecTAR node instances and a server at Monash University. An exploratory attempt was made to utilise Amazon Web Services for an increased node count but the decision was made to not increase the difficulty, which laid in the installation and use of the un-evaluated Puppet Enterprise component, the Cloud Provisioner.

6.4 Difficulties and Problems Encountered

During the course of this research work, some setbacks and difficulties were encountered that prevented work from being done, some of which required a workaround, the others meant a limitation on scope which have been mentioned in the previous section. These difficulties and problems encountered are briefly described as follows:

- When the research was in the literature review phase, it was discovered that little to no research was done in the area of Configuration Management for High Performance Computing on Clouds. There were benchmarking research attempts and suitability research done, but not specific to the area that this research was looking at, which increased the difficulty in trying something *novel*, so to say.

- During the selection of a CM Tool, due to a non-existent budget, and while trying to maintain the largest set of features and capabilities available at no cost, I decided to select Puppet Enterprise with ten nodes at the free-tier, as compared to Chef with only five nodes. Further to that, Puppet Enterprise was chosen over the open-source Puppet version as it provided a larger feature set and greater support.

- On building the prototype, some difficulties were encountered regarding the server in use at Monash University. There were networking restrictions on the network port, along with the un-availability of additional network switch ports that prevented the initial plan of utilising the server as a virtual machine host for the prototype as there had to be correct DNS configurations with interfaces exposed to the Internet. The deployment architecture was changed at this point in time to reflect that so as to be able to use external nodes at the NecTAR Research Cloud for computational nodes, with the server mentioned previously as the master node, which delayed the entire evaluation and prototyping phase for a month while discussion was going on with eSolutions in attempts to resolve the issues.

- Some other issues such as power outages in the server room environment, along with data corruption issues were at hand, but quickly dealt with and in some cases the evaluation data that was corrupted had to be deleted and the evaluation rerun.

- Finally, it is also acknowledged that the learning process duration for Puppet Enterprise took longer then originally intended due to unforeseen difficulties, which delayed the evaluation phase by two weeks.

6.5 Future Works

The conclusion of this research work has identified some areas for future research, which are as follows:

- Extension of this thesis work by grounding the framework further in evaluating other CM Tools, such as Chef, or other commercial tools available.

- The area of research in Configuration Management within the High Performance Computing over Cloud platforms is lacking research and should be looked at in an attempt to properly classify the differences between Dynamic and Static Configuration Management, of which some attempts have been made in the past.

- In an extension of the prototype, a deployment environment could be made using the newer version of Puppet Enterprise which is 3.0, and is supposed to achieve efficiency and performance increases of 200%.

- As mentioned previously, a test of scalability could be done using an increased set of computational nodes, for example in the ranges of 500 to 10000.

Appendix A

PE Manifest Source Code

Appendix A contains the manifest code that was developed for the prototype evaluation. The configuration files referenced in the source code in Listing A.3 is not included for security reasons.

```
class r2 {
  package {'R':
    ensure => present,
    version => '2.13.0-2',
  }
}
```

Listing A.1: R2 Manifest Source Code

```
class r3 {
  package {'R':
    ensure => present,
    version => '3.0.1-2',
  }
}
```

Listing A.2: R3 Manifest Source Code

```
class sshd-en {
    package { 'openssh-server':
      ensure => present,
      before => [ File['/etc/ssh/sshd_config'], File['/home/ec2-user/.
          ssh/authorized_keys'] ],
    }
    file { '/etc/ssh/sshd_config':
      ensure => file,
      mode   => 600,
      source => 'puppet:///modules/sshd/sshd_config',
    }
    file { '/home/ec2-user/.ssh/authorized_keys':
      ensure => file,
      owner => ec2-user,
      group => ec2-user,
      mode   => 600,
      source => 'puppet:///modules/sshd/authorized_keys',
    }
    service { 'sshd':
      ensure     => running,
      enable     => true,
      hasrestart => true,
      hasstatus  => true,
      subscribe  => File['/etc/ssh/sshd_config'],
    }
}
```

Listing A.3: sshd-en Manifest Source Code

```
class sshd-dis {
    package { 'openssh-server':
      ensure => absent,
    }
}
```

Listing A.4: sshd-dis Manifest Source Code

```
class openmpi {
  package {'openmpi':
    ensure => present,
    version => 'latest',
  }
}
```

Listing A.5: openmpi Manifest Source Code

Appendix B

Evaluation Scripted Process Source Code

Appendix B lists the two bash scripts used in evaluating the prototype in the Scripted Configuration scenario discussed previously. Listing B.1 shows the script used in the first setting using new nodes, and Listing B.2 shows the script used in the second setting using existing nodes.

```
#!/bin/bash
# Scripted configuration eval
# installs openmpi-1.5.4-1 and R-3.0.1-2
# calculates the time taken
# time is in seconds with two digits of nanoseconds
start=`date +%s%N | cut -b1-12`
yum -y install R openmpi
end=`date +%s%N | cut -b1-12`
timetaken=`expr $end - $start`
echo $timetaken > /tmp/tdata
```

Listing B.1: New Node (Setting 1) Evaluation Scripted Process Source Code

```
#!/bin/bash
# Scripted configuration eval 2
# installs openmpi-1.5.4-1 and R-2.13.0-2
# calculates the time taken
# time is in seconds with two digits of nanoseconds
start=`date +%s%N | cut -b1-12`
yum -y remove R openmpi
wget http://pkgs.repoforge.org/R/R-2.13.0-2.el6.rf.x86_64.rpm .
rpm -ivh R-2.13.0-2.el6.rf.x86_64.rpm
wget http://mirror.centos.org/centos/6/os/x86_64/Packages/openmpi
    -1.5.4-1.el6.x86_64.rpm .
rpm -ivh openmpi-1.5.4-1.el6.x86_64.rpm
end=`date +%s%N | cut -b1-12`
timetaken=`expr $end - $start`
echo $timetaken > /tmp/tdata2
```

Listing B.2: Existing Node (Setting 2) Evaluation Scripted Process Source Code

Appendix C

Prototype Evaluation Raw Data

Appendix C contains the evaluation raw data for the prototype. Table C.1 provides the legend for the scenario numbers used in the raw data, Table C.2 and Table C.6 contains the semi-processed data obtained, and Tables C.3, C.4, C.5, C.7, C.8, C.9 contain the raw data collected as a result of the evaluation process.

ScenarioNo	Description	Tool/s
1	Manual Configuration	Manual/ By Hand
2	Scripted Configuration	Bash Scripting
3	Automated Configuration	Puppet Enterprise

Table C.1: Raw data legend for the Scenario Numbers referenced

SeqNo	Scenario	Source	Time	Processed Time(s)	Reliability Raw	Efficiency Avg
1	1	Node1	94613	946.13	92.00%	943.02
1	1	Node2	94525	945.25		
1	1	Node3	94115	941.15		
1	1	Node4	94359	943.59		
1	1	Node5	94173	941.73		
1	1	Node6	94109	941.09		
1	1	Node7	94120	941.20		
1	1	Node8	94398	943.98		
2	1	Node1	94256	942.56	93.00%	942.15
2	1	Node2	94231	942.31		
2	1	Node3	94342	943.42		
2	1	Node4	94148	941.48		
2	1	Node5	94235	942.35		

Continued. . .

SeqNo	Scenario	Source	Time	Processed Time(s)	Reliability Raw	Efficiency Avg
2	1	Node6	94187	941.87		
2	1	Node7	94105	941.05		
2	1	Node8	94218	942.18		
3	1	Node1	94196	941.96	92.50%	942.53
3	1	Node2	94267	942.67		
3	1	Node3	94169	941.69		
3	1	Node4	94490	944.90		
3	1	Node5	94260	942.60		
3	1	Node6	94327	943.27		
3	1	Node7	94245	942.45		
3	1	Node8	94067	940.67		
4	1	Node1	94070	940.70	91.00%	942.31
4	1	Node2	94079	940.79		
4	1	Node3	94009	940.09		
4	1	Node4	94179	941.79		
4	1	Node5	94409	944.09		
4	1	Node6	94337	943.37		
4	1	Node7	94276	942.76		
4	1	Node8	94487	944.87		
5	1	Node1	94239	942.39	93.00%	941.91
5	1	Node2	94377	943.77		
5	1	Node3	94013	940.13		
5	1	Node4	94168	941.68		
5	1	Node5	94004	940.04		
5	1	Node6	94364	943.64		
5	1	Node7	94265	942.65		
5	1	Node8	94097	940.97		
6	1	Node1	94317	943.17	92.00%	942.95
6	1	Node2	94377	943.77		
6	1	Node3	94437	944.37		
6	1	Node4	94293	942.93		
6	1	Node5	94195	941.95		
6	1	Node6	94045	940.45		
6	1	Node7	94481	944.81		
6	1	Node8	94214	942.14		
7	1	Node1	94212	942.12	90.00%	940.77

Continued...

SeqNo	Scenario	Source	Time	Processed Time(s)	Reliability Raw	Efficiency Avg
7	1	Node2	93861	938.61		
7	1	Node3	93807	938.07		
7	1	Node4	94246	942.46		
7	1	Node5	94117	941.17		
7	1	Node6	94231	942.31		
7	1	Node7	94261	942.61		
7	1	Node8	93878	938.78		
8	1	Node1	94132	941.32	92.00%	941.81
8	1	Node2	94046	940.46		
8	1	Node3	94340	943.40		
8	1	Node4	94257	942.57		
8	1	Node5	94361	943.61		
8	1	Node6	93983	939.83		
8	1	Node7	94166	941.66		
8	1	Node8	94160	941.60		
9	1	Node1	94031	940.31	93.00%	941.82
9	1	Node2	94208	942.08		
9	1	Node3	94317	943.17		
9	1	Node4	94205	942.05		
9	1	Node5	94233	942.33		
9	1	Node6	94274	942.74		
9	1	Node7	93877	938.77		
9	1	Node8	94307	943.07		
10	1	Node1	93872	938.72	91.00%	942.24
10	1	Node2	94461	944.61		
10	1	Node3	94213	942.13		
10	1	Node4	94127	941.27		
10	1	Node5	93958	939.58		
10	1	Node6	94524	945.24		
10	1	Node7	94211	942.11		
10	1	Node8	94426	944.26		

Table C.3: Raw Result Data - Results for Setting 1 - New Nodes, Scenario 1 - Manual Configuration

SeqNo	Scenario	Source	Time	Processed Time(s)	Reliability Raw	Efficiency Avg
1	2	Node1	79994	799.94	97.00%	801.73
1	2	Node2	80101	801.01		
1	2	Node3	80303	803.03		
1	2	Node4	80077	800.77		
1	2	Node5	80423	804.23		
1	2	Node6	80171	801.71		
1	2	Node7	80024	800.24		
1	2	Node8	80293	802.93		
2	2	Node1	80090	800.90	97.00%	801.83
2	2	Node2	80232	802.32		
2	2	Node3	80361	803.61		
2	2	Node4	80199	801.99		
2	2	Node5	80211	802.11		
2	2	Node6	80192	801.92		
2	2	Node7	80223	802.23		
2	2	Node8	79957	799.57		
3	2	Node1	80300	803.00	97.00%	800.81
3	2	Node2	80101	801.01		
3	2	Node3	80302	803.02		
3	2	Node4	80325	803.25		
3	2	Node5	79986	799.86		
3	2	Node6	79841	798.41		
3	2	Node7	79885	798.85		
3	2	Node8	79909	799.09		
4	2	Node1	80212	802.12	96.00%	801.88
4	2	Node2	80133	801.33		
4	2	Node3	80394	803.94		
4	2	Node4	80108	801.08		
4	2	Node5	79973	799.73		
4	2	Node6	80019	800.19		
4	2	Node7	80271	802.71		
4	2	Node8	80396	803.96		
5	2	Node1	80136	801.36	97.00%	802.00
5	2	Node2	80071	800.71		
5	2	Node3	80143	801.43		
5	2	Node4	80304	803.04		

Continued. . .

SeqNo	Scenario	Source	Time	Processed Time(s)	Reliability Raw	Efficiency Avg
5	2	Node5	80234	802.34		
5	2	Node6	80356	803.56		
5	2	Node7	80049	800.49		
5	2	Node8	80303	803.03		
6	2	Node1	80102	801.02	97.00%	802.61
6	2	Node2	80063	800.63		
6	2	Node3	80219	802.19		
6	2	Node4	80344	803.44		
6	2	Node5	80412	804.12		
6	2	Node6	80187	801.87		
6	2	Node7	80342	803.42		
6	2	Node8	80422	804.22		
7	2	Node1	80091	800.91	97.00%	801.86
7	2	Node2	80164	801.64		
7	2	Node3	80388	803.88		
7	2	Node4	80079	800.79		
7	2	Node5	80238	802.38		
7	2	Node6	80115	801.15		
7	2	Node7	80031	800.31		
7	2	Node8	80384	803.84		
8	2	Node1	80011	800.11	98.00%	800.91
8	2	Node2	80088	800.88		
8	2	Node3	80226	802.26		
8	2	Node4	80193	801.93		
8	2	Node5	80158	801.58		
8	2	Node6	79965	799.65		
8	2	Node7	79989	799.89		
8	2	Node8	80095	800.95		
9	2	Node1	79965	799.65	97.00%	801.38
9	2	Node2	80094	800.94		
9	2	Node3	79999	799.99		
9	2	Node4	80234	802.34		
9	2	Node5	80049	800.49		
9	2	Node6	80429	804.29		
9	2	Node7	80034	800.34		
9	2	Node8	80302	803.02		

Continued. . .

SeqNo	Scenario	Source	Time	Processed Time(s)	Reliability Raw	Efficiency Avg
10	2	Node1	79985	799.85	97.00%	800.63
10	2	Node2	80115	801.15		
10	2	Node3	80061	800.61		
10	2	Node4	80043	800.43		
10	2	Node5	80104	801.04		
10	2	Node6	80122	801.22		
10	2	Node7	79950	799.50		
10	2	Node8	80126	801.26		

Table C.4: Raw Result Data - Results for Setting 1 - New Nodes, Scenario 2 - Scripted Configuration

SeqNo	Scenario	Source	Time	Processed Time(s)	Reliability Raw	Efficiency Avg
1	3	Node1	18505	185.05	100.00%	185.51
1	3	Node2	18688	186.88		
1	3	Node3	18435	184.35		
1	3	Node4	18631	186.31		
1	3	Node5	18502	185.02		
1	3	Node6	18594	185.94		
1	3	Node7	18610	186.10		
1	3	Node8	18442	184.42		
2	3	Node1	18511	185.11	100.00%	186.35
2	3	Node2	18587	185.87		
2	3	Node3	18543	185.43		
2	3	Node4	18587	185.87		
2	3	Node5	18678	186.78		
2	3	Node6	18734	187.34		
2	3	Node7	18678	186.78		
2	3	Node8	18759	187.59		
3	3	Node1	18715	187.15	100.00%	186.73
3	3	Node2	18691	186.91		
3	3	Node3	18719	187.19		
3	3	Node4	18660	186.60		
3	3	Node5	18687	186.87		

Continued...

SeqNo	Scenario	Source	Time	Processed Time(s)	Reliability Raw	Efficiency Avg
3	3	Node6	18579	185.79		
3	3	Node7	18655	186.55		
3	3	Node8	18677	186.77		
4	3	Node1	18672	186.72	100.00%	186.06
4	3	Node2	18605	186.05		
4	3	Node3	18471	184.71		
4	3	Node4	18748	187.48		
4	3	Node5	18653	186.53		
4	3	Node6	18741	187.41		
4	3	Node7	18513	185.13		
4	3	Node8	18446	184.46		
5	3	Node1	18609	186.09	100.00%	186.56
5	3	Node2	18561	185.61		
5	3	Node3	18752	187.52		
5	3	Node4	18522	185.22		
5	3	Node5	18794	187.94		
5	3	Node6	18623	186.23		
5	3	Node7	18752	187.52		
5	3	Node8	18631	186.31		
6	3	Node1	18614	186.14	100.00%	186.12
6	3	Node2	18662	186.62		
6	3	Node3	18693	186.93		
6	3	Node4	18585	185.85		
6	3	Node5	18616	186.16		
6	3	Node6	18532	185.32		
6	3	Node7	18624	186.24		
6	3	Node8	18570	185.70		
7	3	Node1	18733	187.33	100.00%	186.30
7	3	Node2	18610	186.10		
7	3	Node3	18476	184.76		
7	3	Node4	18653	186.53		
7	3	Node5	18701	187.01		
7	3	Node6	18620	186.20		
7	3	Node7	18692	186.92		
7	3	Node8	18554	185.54		
8	3	Node1	18624	186.24	100.00%	186.43

Continued. . .

SeqNo	Scenario	Source	Time	Processed Time(s)	Reliability Raw	Efficiency Avg
8	3	Node2	18729	187.29		
8	3	Node3	18571	185.71		
8	3	Node4	18545	185.45		
8	3	Node5	18713	187.13		
8	3	Node6	18610	186.10		
8	3	Node7	18615	186.15		
8	3	Node8	18739	187.39		
9	3	Node1	18664	186.64	100.00%	186.61
9	3	Node2	18696	186.96		
9	3	Node3	18728	187.28		
9	3	Node4	18625	186.25		
9	3	Node5	18624	186.24		
9	3	Node6	18611	186.11		
9	3	Node7	18712	187.12		
9	3	Node8	18627	186.27		
10	3	Node1	18573	185.73	100.00%	186.16
10	3	Node2	18490	184.90		
10	3	Node3	18699	186.99		
10	3	Node4	18613	186.13		
10	3	Node5	18647	186.47		
10	3	Node6	18611	186.11		
10	3	Node7	18632	186.32		
10	3	Node8	18660	186.60		

Table C.5: Raw Result Data - Results for Setting 1 - New Nodes, Scenario 3 - Automated Configuration

SeqNo	Scenario	Source	Time	Processed Time(s)	Reliability Raw	Efficiency Avg
1	1	Node1	105605	1056.05	90.00%	1055.61
1	1	Node2	105544	1055.44		
1	1	Node3	105646	1056.46		
1	1	Node4	105665	1056.65		
1	1	Node5	105268	1052.68		

Continued...

SeqNo	Scenario	Source	Time	Processed Time(s)	Reliability Raw	Efficiency Avg
1	1	Node6	105628	1056.28		
1	1	Node7	105577	1055.77		
1	1	Node8	105556	1055.56		
2	1	Node1	105331	1053.31	88.00%	1055.38
2	1	Node2	105696	1056.96		
2	1	Node3	105494	1054.94		
2	1	Node4	105478	1054.78		
2	1	Node5	105674	1056.74		
2	1	Node6	105531	1055.31		
2	1	Node7	105599	1055.99		
2	1	Node8	105497	1054.97		
3	1	Node1	105549	1055.49	88.00%	1055.52
3	1	Node2	105501	1055.01		
3	1	Node3	105592	1055.92		
3	1	Node4	105690	1056.90		
3	1	Node5	105568	1055.68		
3	1	Node6	105541	1055.41		
3	1	Node7	105426	1054.26		
3	1	Node8	105545	1055.45		
4	1	Node1	105548	1055.48	90.00%	1055.93
4	1	Node2	105719	1057.19		
4	1	Node3	105575	1055.75		
4	1	Node4	105718	1057.18		
4	1	Node5	105498	1054.98		
4	1	Node6	105555	1055.55		
4	1	Node7	105519	1055.19		
4	1	Node8	105614	1056.14		
5	1	Node1	105583	1055.83	88.50%	1056.35
5	1	Node2	105752	1057.52		
5	1	Node3	105532	1055.32		
5	1	Node4	105716	1057.16		
5	1	Node5	105613	1056.13		
5	1	Node6	105541	1055.41		
5	1	Node7	105752	1057.52		
5	1	Node8	105587	1055.87		
6	1	Node1	105719	1057.19	90.00%	1056.29

Continued...

SeqNo	Scenario	Source	Time	Processed Time(s)	Reliability Raw	Efficiency Avg
6	1	Node2	105573	1055.73		
6	1	Node3	105621	1056.21		
6	1	Node4	105597	1055.97		
6	1	Node5	105666	1056.66		
6	1	Node6	105586	1055.86		
6	1	Node7	105673	1056.73		
6	1	Node8	105595	1055.95		
7	1	Node1	105465	1054.65	89.00%	1055.75
7	1	Node2	105576	1055.76		
7	1	Node3	105575	1055.75		
7	1	Node4	105666	1056.66		
7	1	Node5	105549	1055.49		
7	1	Node6	105591	1055.91		
7	1	Node7	105553	1055.53		
7	1	Node8	105627	1056.27		
8	1	Node1	105573	1055.73	88.00%	1055.27
8	1	Node2	105500	1055.00		
8	1	Node3	105532	1055.32		
8	1	Node4	105631	1056.31		
8	1	Node5	105515	1055.15		
8	1	Node6	105489	1054.89		
8	1	Node7	105536	1055.36		
8	1	Node8	105436	1054.36		
9	1	Node1	105461	1054.61	87.00%	1055.21
9	1	Node2	105435	1054.35		
9	1	Node3	105475	1054.75		
9	1	Node4	105449	1054.49		
9	1	Node5	105639	1056.39		
9	1	Node6	105622	1056.22		
9	1	Node7	105491	1054.91		
9	1	Node8	105594	1055.94		
10	1	Node1	105678	1056.78	91.00%	1055.97
10	1	Node2	105593	1055.93		
10	1	Node3	105675	1056.75		
10	1	Node4	105411	1054.11		
10	1	Node5	105612	1056.12		

Continued...

SeqNo	Scenario	Source	Time	Processed Time(s)	Reliability Raw	Efficiency Avg
10	1	Node6	105837	1058.37		
10	1	Node7	105508	1055.08		
10	1	Node8	105464	1054.64		

Table C.7: Raw Result Data - Results for Setting 2 - New Nodes, Scenario 1 - Manual Configuration

SeqNo	Scenario	Source	Time	Processed Time(s)	Reliability Raw	Efficiency Avg
1	2	Node1	87182	871.82	98.00%	871.94
1	2	Node2	87182	871.82		
1	2	Node3	87173	871.73		
1	2	Node4	87221	872.21		
1	2	Node5	87207	872.07		
1	2	Node6	87172	871.72		
1	2	Node7	87212	872.12		
1	2	Node8	87200	872.00		
2	2	Node1	87207	872.07	98.00%	872.08
2	2	Node2	87260	872.60		
2	2	Node3	87117	871.17		
2	2	Node4	87174	871.74		
2	2	Node5	87274	872.74		
2	2	Node6	87252	872.52		
2	2	Node7	87172	871.72		
2	2	Node8	87204	872.04		
3	2	Node1	87135	871.35	98.00%	872.13
3	2	Node2	87193	871.93		
3	2	Node3	87249	872.49		
3	2	Node4	87260	872.60		
3	2	Node5	87208	872.08		
3	2	Node6	87247	872.47		
3	2	Node7	87181	871.81		
3	2	Node8	87232	872.32		
4	2	Node1	87210	872.10	98.00%	872.27
4	2	Node2	87208	872.08		

Continued...

SeqNo	Scenario	Source	Time	Processed Time(s)	Reliability Raw	Efficiency Avg
4	2	Node3	87232	872.32		
4	2	Node4	87275	872.75		
4	2	Node5	87203	872.03		
4	2	Node6	87206	872.06		
4	2	Node7	87228	872.28		
4	2	Node8	87257	872.57		
5	2	Node1	87186	871.86	98.00%	872.17
5	2	Node2	87272	872.72		
5	2	Node3	87245	872.45		
5	2	Node4	87268	872.68		
5	2	Node5	87219	872.19		
5	2	Node6	87155	871.55		
5	2	Node7	87174	871.74		
5	2	Node8	87213	872.13		
6	2	Node1	87162	871.62	99.00%	871.92
6	2	Node2	87186	871.86		
6	2	Node3	87195	871.95		
6	2	Node4	87208	872.08		
6	2	Node5	87197	871.97		
6	2	Node6	87180	871.80		
6	2	Node7	87195	871.95		
6	2	Node8	87210	872.10		
7	2	Node1	87228	872.28	98.00%	872.27
7	2	Node2	87229	872.29		
7	2	Node3	87243	872.43		
7	2	Node4	87210	872.10		
7	2	Node5	87218	872.18		
7	2	Node6	87252	872.52		
7	2	Node7	87244	872.44		
7	2	Node8	87194	871.94		
8	2	Node1	87230	872.30	97.00%	872.28
8	2	Node2	87249	872.49		
8	2	Node3	87254	872.54		
8	2	Node4	87206	872.06		
8	2	Node5	87222	872.22		
8	2	Node6	87209	872.09		

Continued...

SeqNo	Scenario	Source	Time	Processed Time(s)	Reliability Raw	Efficiency Avg
8	2	Node7	87260	872.60		
8	2	Node8	87197	871.97		
9	2	Node1	87229	872.29	98.00%	872.22
9	2	Node2	87186	871.86		
9	2	Node3	87225	872.25		
9	2	Node4	87184	871.84		
9	2	Node5	87215	872.15		
9	2	Node6	87267	872.67		
9	2	Node7	87245	872.45		
9	2	Node8	87222	872.22		
10	2	Node1	87207	872.07	98.00%	872.25
10	2	Node2	87209	872.09		
10	2	Node3	87206	872.06		
10	2	Node4	87221	872.21		
10	2	Node5	87275	872.75		
10	2	Node6	87202	872.02		
10	2	Node7	87224	872.24		
10	2	Node8	87259	872.59		

Table C.8: Raw Result Data - Results for Setting 2 - New Nodes, Scenario 2 - Scripted Configuration

SeqNo	Scenario	Source	Time	Processed Time(s)	Reliability Raw	Efficiency Avg
1	3	Node1	19241	192.41	100.00%	192.41
1	3	Node2	19221	192.21		
1	3	Node3	19218	192.18		
1	3	Node4	19221	192.21		
1	3	Node5	19221	192.21		
1	3	Node6	19250	192.50		
1	3	Node7	19258	192.58		
1	3	Node8	19297	192.97		
2	3	Node1	19227	192.27	100.00%	192.44
2	3	Node2	19259	192.59		
2	3	Node3	19224	192.24		

Continued...

SeqNo	Scenario	Source	Time	Processed Time(s)	Reliability Raw	Efficiency Avg
2	3	Node4	19192	191.92		
2	3	Node5	19223	192.23		
2	3	Node6	19380	193.80		
2	3	Node7	19234	192.34		
2	3	Node8	19214	192.14		
3	3	Node1	19213	192.13	100.00%	192.25
3	3	Node2	19225	192.25		
3	3	Node3	19211	192.11		
3	3	Node4	19229	192.29		
3	3	Node5	19247	192.47		
3	3	Node6	19207	192.07		
3	3	Node7	19182	191.82		
3	3	Node8	19285	192.85		
4	3	Node1	19263	192.63	100.00%	192.43
4	3	Node2	19275	192.75		
4	3	Node3	19207	192.07		
4	3	Node4	19167	191.67		
4	3	Node5	19205	192.05		
4	3	Node6	19338	193.38		
4	3	Node7	19221	192.21		
4	3	Node8	19266	192.66		
5	3	Node1	19234	192.34	100.00%	192.30
5	3	Node2	19219	192.19		
5	3	Node3	19203	192.03		
5	3	Node4	19220	192.20		
5	3	Node5	19266	192.66		
5	3	Node6	19248	192.48		
5	3	Node7	19214	192.14		
5	3	Node8	19238	192.38		
6	3	Node1	19266	192.66	100.00%	192.28
6	3	Node2	19202	192.02		
6	3	Node3	19233	192.33		
6	3	Node4	19212	192.12		
6	3	Node5	19231	192.31		
6	3	Node6	19221	192.21		
6	3	Node7	19221	192.21		

Continued...

SeqNo	Scenario	Source	Time	Processed Time(s)	Reliability Raw	Efficiency Avg
6	3	Node8	19239	192.39		
7	3	Node1	19252	192.52	100.00%	192.23
7	3	Node2	19219	192.19		
7	3	Node3	19215	192.15		
7	3	Node4	19222	192.22		
7	3	Node5	19219	192.19		
7	3	Node6	19200	192.00		
7	3	Node7	19218	192.18		
7	3	Node8	19237	192.37		
8	3	Node1	19245	192.45	100.00%	192.29
8	3	Node2	19226	192.26		
8	3	Node3	19293	192.93		
8	3	Node4	19228	192.28		
8	3	Node5	19162	191.62		
8	3	Node6	19213	192.13		
8	3	Node7	19242	192.42		
8	3	Node8	19225	192.25		
9	3	Node1	19244	192.44	100.00%	192.21
9	3	Node2	19246	192.46		
9	3	Node3	19212	192.12		
9	3	Node4	19260	192.60		
9	3	Node5	19221	192.21		
9	3	Node6	19227	192.27		
9	3	Node7	19164	191.64		
9	3	Node8	19195	191.95		
10	3	Node1	19206	192.06	100.00%	192.22
10	3	Node2	19266	192.66		
10	3	Node3	19228	192.28		
10	3	Node4	19294	192.94		
10	3	Node5	19212	192.12		
10	3	Node6	19193	191.93		
10	3	Node7	19208	192.08		
10	3	Node8	19167	191.67		

Table C.9: Raw Result Data - Results for Setting 2 - New Nodes, Scenario 3 - Automated Configuration

Setting No: 1	New Nodes			
SequenceNo	Scenario	Time	ProcessedTime(s)	ReliabilityAverage
1	1	94302	943.02	92.0%
2	1	94215	942.15	
3	1	94253	942.53	
4	1	94231	942.31	
5	1	94191	941.91	
6	1	94295	942.95	
7	1	94077	940.77	
8	1	94181	941.81	
9	1	94182	941.82	
10	1	94224	942.24	
1	2	80173	801.73	97.0%
2	2	80183	801.83	
3	2	80081	800.81	
4	2	80188	801.88	
5	2	80200	802.00	
6	2	80261	802.61	
7	2	80186	801.86	
8	2	80091	800.91	
9	2	80138	801.38	
10	2	80063	800.63	
1	3	18551	185.51	100.0%
2	3	18635	186.35	
3	3	18673	186.73	
4	3	18606	186.06	
5	3	18656	186.56	
6	3	18612	186.12	
7	3	18630	186.30	
8	3	18643	186.43	
9	3	18661	186.61	
10	3	18616	186.16	

Table C.2: Semi-Raw Result Data - Average Summary Results for Setting 1 - New Nodes

Setting No: 2	Existing Nodes			
SequenceNo	Scenario	Time	ProcessedTime(s)	ReliabilityAvg
1	1	105561	1055.61	89.0%
2	1	105538	1055.38	
3	1	105552	1055.52	
4	1	105593	1055.93	
5	1	105635	1056.35	
6	1	105629	1056.29	
7	1	105575	1055.75	
8	1	105527	1055.27	
9	1	105521	1055.21	
10	1	105597	1055.97	
1	2	87194	871.94	98.0%
2	2	87208	872.08	
3	2	87213	872.13	
4	2	87227	872.27	
5	2	87217	872.17	
6	2	87192	871.92	
7	2	87227	872.27	
8	2	87228	872.28	
9	2	87222	872.22	
10	2	87225	872.25	
1	3	19241	192.41	100.0%
2	3	19244	192.44	
3	3	19225	192.25	
4	3	19243	192.43	
5	3	19230	192.30	
6	3	19228	192.28	
7	3	19223	192.23	
8	3	19229	192.29	
9	3	19221	192.21	
10	3	19222	192.22	

Table C.6: Semi-Raw Result Data - Average Summary Results for Setting 2 - Existing Nodes

SequenceNo	Scenario	Processed Time(s)	Standard Deviation	Efficiency Avg
1	1	943.02	0.433	942.15
2	1	942.15	0.002	
3	1	942.53	0.189	
4	1	942.31	0.079	
5	1	941.91	0.120	
6	1	942.95	0.400	
7	1	940.77	0.691	
8	1	941.81	0.171	
9	1	941.82	0.167	
10	1	942.24	0.046	
1	2	801.73	0.084	801.57
2	2	801.83	0.133	
3	2	800.81	0.377	
4	2	801.88	0.159	
5	2	802.00	0.215	
6	2	802.61	0.524	
7	2	801.86	0.149	
8	2	800.91	0.329	
9	2	801.38	0.091	
10	2	800.63	0.466	
1	3	185.51	0.386	186.28
2	3	186.35	0.032	
3	3	186.73	0.224	
4	3	186.06	0.110	
5	3	186.56	0.137	
6	3	186.12	0.081	
7	3	186.30	0.009	
8	3	186.43	0.075	
9	3	186.61	0.164	
10	3	186.16	0.063	

Table C.10: Standard Deviation between Evaluation Sequences - Setting 1 - New Nodes

SequenceNo	Scenario	Processed Time(s)	Standard Deviation	Efficiency Avg
1	1	1055.61	0.058	1055.73
2	1	1055.38	0.176	
3	1	1055.52	0.106	
4	1	1055.93	0.103	
5	1	1056.35	0.309	
6	1	1056.29	0.281	
7	1	1055.75	0.013	
8	1	1055.27	0.231	
9	1	1055.21	0.259	
10	1	1055.97	0.123	
1	2	871.94	0.108	872.15
2	2	872.08	0.039	
3	2	872.13	0.011	
4	2	872.27	0.061	
5	2	872.17	0.006	
6	2	871.92	0.118	
7	2	872.27	0.060	
8	2	872.28	0.066	
9	2	872.22	0.032	
10	2	872.25	0.051	
1	3	192.41	0.051	192.31
2	3	192.44	0.068	
3	3	192.25	0.029	
4	3	192.43	0.061	
5	3	192.30	0.002	
6	3	192.28	0.012	
7	3	192.23	0.039	
8	3	192.29	0.007	
9	3	192.21	0.047	
10	3	192.22	0.044	

Table C.11: Standard Deviation between Evaluation Sequences - Setting 2 - Existing Nodes

References

Abramson, D., Amoreira, C., Baldridge, K., Berstis, L., Kondrick, C., and Peachey, T. (2006). A flexible grid framework for automatic protein-ligand docking. In *e-Science and Grid Computing, 2006. e-Science'06. Second IEEE International Conference on*, pages 47–47. IEEE.

Ahson, S. A. and Ilyas, M. (2010). *Cloud Computing and Software Services: Theory and Techniques*. CRC Press, Inc.

Allan, D. and Nadeau, T. (2006). A framework for multi-protocol label switching (mpls) operations and management (oam).

Amazon (2012). Getting started with amazon ec2 linux instances - amazon elastic compute cloud.

Bethwaite, B., Abramson, D., Bohnert, F., Garic, S., Enticott, C., and Peachey, T. (2010). Mixing grids and clouds: High-throughput science using the nimrod tool family. *Cloud Computing*, pages 219–237.

Brandt, J., Gentile, A., Mayo, J., Pebay, P., Roe, D., Thompson, D., and Wong, M. (2009). Resource monitoring and management with ovis to enable hpc in cloud computing environments. In *Parallel & Distributed Processing, 2009. IPDPS 2009. IEEE International Symposium on*, pages 1–8. IEEE.

Cisco (2007). Network configuration management.

Crane, S., Dulay, N., Fosså, H., Kramer, J., Magee, J., Sloman, M., and Twidle, K. (1995). Configuration management for distributed software services. *Integrated Network Management IV*, 4.

Delaet, T., Anderson, P., and Joosen, W. (2008). Managing real-world system configurations with constraints. In *Networking, 2008. ICN 2008. Seventh International Conference on*, pages 594–601. IEEE.

Dell (2012). Dell kace appliances and dell configuration services.

Diaz, J., von Laszewski, G., Wang, F., and Fox, G. (2012). Abstract image management and universal image registration for cloud and hpc infrastructures. In *Cloud Computing (CLOUD), 2012 IEEE 5th International Conference on*, pages 463–470. IEEE.

Dillon, T., Wu, C., and Chang, E. (2010). Cloud computing: Issues and challenges. In *Advanced Information Networking and Applications (AINA), 2010 24th IEEE International Conference on*, pages 27–33. IEEE.

Dodai, P. (2011). Project dodai.

Ekanayake, J. and Fox, G. (2010). High performance parallel computing with clouds and cloud technologies. *Cloud Computing*, pages 20–38.

eResearchSA (2012). What is eresearch?

Evangelinos, C. and Hill, C. (2008). Cloud computing for parallel scientific hpc applications: Feasibility of running coupled atmosphere-ocean climate models on amazon's ec2. *ratio*, 2(2.40):2.34.

Fujimoto, R., Malik, A., and Park, A. (2010). Parallel and distributed simulation in the cloud. *SCS M&S Magazine*, 3:1–10.

FutureGrid (2012). Futuregrid.

Gallard, J., Lèbre, A., Morin, C., Naughton, T., Scott, S., and Vallée, G. (2012). Architecture for the next generation system management tools. *Future Generation Computer Systems*, 28(1):136–146.

Gillett, F., Brown, E., Staten, J., and Lee, C. (2008). Future view: the new tech ecosystems of cloud, cloud services, and cloud computing. *Forrester Research Paper*.

Hamdaqa, M. and Tahvildari, L. (2012). Cloud computing uncovered: A research landscape. *Advances in Computers*, page 41.

Hill, Z. and Humphrey, M. (2009). A quantitative analysis of high performance computing with amazon's ec2 infrastructure: The death of the local cluster? In *Grid Computing, 2009 10th IEEE/ACM International Conference on*, pages 26–33. IEEE.

IBM (2010). Ibm maximo technology for business and it agility. *IBM white paper*, page 12.

IBM (2012). Ibm tivoli software.

IEEE (1988). Ieee guide to software configuration management. *ANSI/IEEE Std 1042-1987*, page 1.

IEEE (1990). Ieee standard for software configuration management plans. *IEEE Std 828-1990*, page 1.

Intel (2012). Intel hybrid cloud program.

iVEC (2012). Glossary.

Jackson, K., Ramakrishnan, L., Muriki, K., Canon, S., Cholia, S., Shalf, J., Wasserman, H., and Wright, N. (2010). Performance analysis of high performance computing applications on the amazon web services cloud. In *Cloud Computing Technology and Science (CloudCom), 2010 IEEE Second International Conference on*, pages 159–168. IEEE.

Jahidur, R. (2012). Investigating configuration management tools usage in large infrastructure.

Jamjoom, H. T., E., P. M., Huiming, Q., Yaoping, R., R., S. D., Zon-yin, S., and Anshul, S. (2012). High performance computing as a service.

Klinginsmith, J., Mahoui, M., and Wu, Y. M. (2011). Towards reproducible escience in the cloud. In *Cloud Computing Technology and Science (CloudCom), 2011 IEEE Third International Conference on*, pages 582–586. IEEE.

Kramer, J. and Magee, J. (1985). Dynamic configuration for distributed systems. *Software Engineering, IEEE Transactions on Software Engineering*, SE-11(4):424–436.

Magherusan-Stanciu, C., Sebestyen-Pal, A., Cebuc, E., Sebestyen-Pal, G., and Dadarlat, V. (2011). Grid system installation, management and monitoring application. In *Parallel and Distributed Computing (ISPDC), 2011 10th International Symposium on*, pages 25–32. IEEE.

Mahmoud, M., Majid, S., Jefferson, T., and Fateme, K. (2012). Scaling up transit priority modelling using high-throughput computing. In *Tenth Australasian Symposium on Parallel and Distributed Computing (AusPDC 2012)*, volume 127, pages 53–62. Australian Computer Society (ACS).

Mell, P. and Grance, T. (2009). The nist definition of cloud computing. *National Institute of Standards and Technology*, 53(6):50.

MeSsAGELab (2012). escience applications.

Mikkilineni, R. and Sarathy, V. (2009). Cloud computing and the lessons from the past. In *Enabling Technologies: Infrastructures for Collaborative Enterprises, 2009. WETICE'09. 18th IEEE International Workshops on*, pages 57–62. IEEE.

NeSC (2001). What is e-science?

Oliveira, D., Baião, F., and Mattoso, M. (2010). Towards a taxonomy for cloud computing from an e-science perspective. *Cloud Computing*, pages 47–62.

Oliveira, D., Cunha, L., Tomaz, L., Pereira, V., and Mattoso, M. (2009). Using ontologies to support deep water oil exploration scientific workflows. In *Services-I, 2009 World Conference on*, pages 364–367. IEEE.

Önnberg, F. (2012). *Software Configuration Management: A comparison of Chef, CFEngine and Puppet*. PhD thesis, University of Skövde.

Oppenheimer, D. (2003). The importance of understanding distributed system configuration. In *Proceedings of the 2003 Conference on Human Factors in Computer Systems workshop*.

Oppenheimer, P. (2011). *Developing Network Management Strategies*, volume 1, chapter 9, page 226. Cisco Press, Indianapolis, IN 46240, USA, 3 edition.

OpsCode (2013). Opscode learn chef.

Ostermann, S., Iosup, A., Yigitbasi, N., Prodan, R., Fahringer, T., and Epema, D. (2010). A performance analysis of ec2 cloud computing services for scientific computing. *Cloud Computing*, pages 115–131.

Patterson, D. (2002). A simple way to estimate the cost of downtime. In *Proc. 16th Systems Administration Conf.— LISA*, pages 185–8.

PuppetLabs (2012a). Compare puppet & puppet enterprise.

PuppetLabs (2012b). Puppet open source.

PuppetLabs (2012c). What is puppet?

Rackspace (2012). Public cloud.

Rahman, J. (2012). Investigating configuration management tools usage in large infrastructure. Master's thesis, Department of Informatics, University of Oslo.

Raicu, I., Foster, I. T., and Zhao, Y. (2008). Many-task computing for grids and supercomputers. In *Many-Task Computing on Grids and Supercomputers, 2008. MTAGS 2008. Workshop on*, pages 1–11. IEEE.

Ramakrishnan, L., Jackson, K., Canon, S., Cholia, S., and Shalf, J. (2010). Defining future platform requirements for e-science clouds. In *Proceedings of the 1st ACM symposium on Cloud computing*, pages 101–106. ACM.

Ramakrishnan, L., Zbiegel, P., Campbell, S., Bradshaw, R., Canon, R., Coghlan, S., Sakrejda, I., Desai, N., Declerck, T., and Liu, A. (2011). Magellan: experiences from a science cloud. In *Proceedings of the 2nd international workshop on Scientific cloud computing*, pages 49–58. ACM.

Rimal, B., Choi, E., and Lumb, I. (2010). A taxonomy, survey, and issues of cloud computing ecosystems. *Cloud Computing*, pages 21–46.

Simmhan, Y., van Ingen, C., Subramanian, G., and Li, J. (2010). Bridging the gap between desktop and the cloud for escience applications. In *Cloud Computing (CLOUD), 2010 IEEE 3rd International Conference on*, pages 474–481. IEEE.

Strijkers, R., Toorop, W., van Hoof, A., Grosso, P., Belloum, A., Vasuining, D., de Laat, C., and Meijer, R. (2010). Amos: Using the cloud for on-demand execution of e-science applications. In *e-Science (e-Science), 2010 IEEE Sixth International Conference on*, pages 331–338. IEEE.

Strohmaier, E., Dongarra, J. J., Meuer, H. W., and Simon, H. D. (2005). Recent trends in the marketplace of high performance computing. *Parallel Computing*, 31(3-4):261–273.

Tonido (2012). Tonido launches private cloud storage alternative to google drive for enterprises.

Ubuntu (2012). Cloudinit.

Vecchiola, C., Pandey, S., and Buyya, R. (2009). High-performance cloud computing: A view of scientific applications. In *Pervasive Systems, Algorithms, and Networks (ISPAN), 2009 10th International Symposium on*, pages 4–16. IEEE.

VMware (2012). Overview of vmware tools.

Vöckler, J., Juve, G., Deelman, E., Rynge, M., and Berriman, B. (2011). Experiences using cloud computing for a scientific workflow application. In *Proceedings of the 2nd international workshop on Scientific cloud computing*, pages 15–24. ACM.

von Laszewski, G., Diaz, J., Wang, F., and Fox, G. (2012). Comparison of multiple cloud frameworks. In *Cloud Computing (CLOUD), 2012 IEEE 5th International Conference on*, pages 734–741. IEEE, IEEE.

Wang, L., Tao, J., Kunze, M., Castellanos, A., Kramer, D., and Karl, W. (2008). Scientific cloud computing: Early definition and experience. In *High Performance Computing and Communications, 2008. HPCC'08. 10th IEEE International Conference on*, pages 825–830. Ieee.

Wang, L., Zhan, J., Shi, W., Liang, Y., and Yuan, L. (2009). In cloud, do mtc or htc service providers benefit from the economies of scale? In *Proceedings of the 2nd Workshop on Many-Task Computing on Grids and Supercomputers*, page 7. ACM.

Wood, K. and Pereira, E. (2011). Impact of misconfiguration in cloud–investigation into security challenges.

Yokoyama, S. and Yoshioka, N. (2012). Cluster as a service for self-deployable cloud applications. In *Cluster, Cloud and Grid Computing (CCGrid), 2012 12th IEEE/ACM International Symposium on*, pages 703–704. IEEE.

Yokoyama, S., Yoshioka, N., and Shida, T. (2012). Cloud in a cloud for cloud education. In *Principles of Engineering Service Oriented Systems (PESOS), 2012 ICSE Workshop on*, pages 63–64. IEEE.

Zhang, Q., Cheng, L., and Boutaba, R. (2010). Cloud computing: state-of-the-art and research challenges. *Journal of Internet Services and Applications*, 1(1):7–18.