UNIVERSITY OF OSLO
Department of Informatics

A Self-management Approach to Service Optimization and System Integrity through Multi-agent Systems

Lu Xing

Oslo University College

May 19, 2008

A Self-management Approach to Service Optimization and System Integrity through Multi-agent Systems

Lu Xing

Oslo University College

May 19, 2008

Abstract

The primary goal of this thesis is to make simple approaches to implement self-management functions based on the regular administration tools as a way to simplify tedious tasks for the system administrators. In order to improve the efficiency of system administration, this work mainly focuses on how to combine ideas from the area of self-managing with the more well-established concepts from virtualization technology.

A fully functioning self-management system was developed for two essential administration tasks in this project: service optimization and system integrity. In these scenarios, each virtual machine can make its own decision when and where to live migrate itself between the physical nodes in order to reach the predefined policies. This method minimizes the maintaining tasks for the system administrators, only leaving the high-lever administration policies for the users to decide. From a higher point of view, this project can be used as an example for other system administrators, showing how to optimize the regular administration tools for different self-management purposes.

Acknowledgements

First and foremost, I would like to express my genuine thanks to my supervisor Kyrre M. Begnum for his support through the entire project. His continuous inspiration always brings genius and valuable ideas into this experiment. Thank you for finding resources to support this project, showing great encouraging in my progress, modifying MLN to suit this projects needs and all kinds of mental and technical support. Without your help, this project would never have been the same. I also greatly appreciate the cooperation with Æleen Frisch from Exponential Consulting, Erik Hjelmås and Jon Langseth from Gjøvik University College. Thanks for their help with offering the servers and network, adapting their network for our needs, and maintaining servers for us. All their great efforts help me to finally implement this experiment.

Special thanks to Professor Mark Burgess. Thank you for your effort in improving this master program, cooperating through the entire masters course and seeking all kinds of opportunities for us. I am always proud to be your student and honored in participating this master program. Thanks to all the teachers and fellow classmates I have been working with. Their knowledge and experience help me to open mind and see a different world. Special thanks to Marius B.Gundersen, his cooperation and feedback has been a good motivation for me to try my best in this project.

Last but not least, I want to thank my dearest family. Thanks for supporting me to realize my dream, and letting the only child stay far away from home for such long time. I am also forever grateful to have my dearest boyfriend standing beside me for these months. Thanks for inspiring and encouraging me all the time. Especially, thanks for participating in this project, discussing with me and suffering from my poor English all the time. I really appreciate what you have done, that is more than what I can express.

Once again, thank you all.

Oslo, May 2008

Lu Xing

Contents

1 Introduction **1**
1.1 Motivation 1
1.2 Example Case: A Global Fabric for Service Placement 2
1.2.1 Challenge 2
1.2.2 Approach: A Global Infrastructure with Manual Management . . 2
1.2.3 Analysis 3
1.2.4 Improved Approach: Global Infrastructure with Autonomic Virtual Machines 5
1.3 Problem Statement 5
1.4 Thesis Outline 7

2 Background **9**
2.1 Virtualization Technology 9
2.1.1 Introduction to Virtualization 9
2.1.2 Advantages and Disadvantages of Virtualization 9
2.1.3 Classification of Virtualization 10
2.1.4 Migration of Virtual Machines 14
2.2 Autonomic Computing and Self-management 15
2.2.1 Autonomic Computing 15
2.2.2 Self-management 16
2.3 Multi-agent Systems 17
2.3.1 Introduction to Multi-agent Systems 17
2.3.2 Concepts in Multi-agent Systems 17
2.3.3 Multi-agent Systems Models 18
2.4 Virtual Private Network (VPN) 20
2.4.1 VTun 20
2.4.2 Creating a VTun VPN 21
2.5 Xen 22
2.5.1 Xen Architecture 22
2.5.2 Migration in Xen 22
2.6 Storage Area Network (SAN) 23
2.7 Shared Storage Infrastructure 24
2.8 Manage Large Networks (MLN) 25

2.8.1 Defining Project in MLN . . . 25
2.8.2 Building up Project in MLN . . . 26
2.9 Cfengine . . . 26
2.9.1 Cfengine Components . . . 28
2.9.2 Cfengine Configuration File . . . 28

3 Methodology 31
3.1 Objectives . . . 31
3.2 Environment . . . 32
3.2.1 Physical Servers . . . 32
3.3 Architecture Overview . . . 33
3.4 Infrastructure Design Requirements . . . 34
3.5 Network and Storage Infrastructure . . . 34
3.6 Server and Software Environment . . . 35
3.7 Model of Migration . . . 36
3.7.1 Virtual Machines Migrate Through Tunnels . . . 36
3.8 Virtual Machine Self-management through Migration . . . 37
3.9 Scenarios . . . 39
3.9.1 Scenario 1: Self-management in Service Optimization . . . 39
3.9.2 Scenario 2: Self-management in System Integrity . . . 41

4 Result 1: System and Network Setup 43
4.1 Creating Virtual Machines . . . 43
4.1.1 Installation and Configuration of Xen . . . 43
4.1.2 Using MLN to Set up Virtual Machines . . . 44
4.2 Setting up Network Infrastructure . . . 46
4.2.1 Configuring Shared Storage . . . 46
4.2.2 Creating Virtual Tunnels . . . 46
4.3 Building Experiment Environment . . . 47
4.3.1 Dynamic DNS . . . 47

5 Result 2: Implementation of the Experiment 49
5.1 Self-management Operational Strategy . . . 49
5.1.1 Developed Scripts and Their Functions . . . 51
5.1.2 Defining Parameters for Analysis Script . . . 51
5.1.3 Service Optimization Process . . . 52
5.1.4 System Integrity Process . . . 54
5.2 User Request Generator . . . 55
5.2.1 Formula for Generating Requests . . . 55
5.2.2 Using Generators with Different Magnitudes . . . 55
5.2.3 Adding Noise in Generator Scripts . . . 57
5.2.4 Adding Sleep Time in Generator Scripts . . . 57
5.2.5 Using Generators with Different Sleep Time . . . 58
5.2.6 Using Generators with Different Periods . . . 59

5.3 Self-monitoring Function . . . 61
5.4 Self-analysis Function . . . 61
5.4.1 Self-analysis Function in Service Optimization Scenario . . . 62
5.4.1.1 Analyzing Highest Request Rate Percentage . . . 62
5.4.1.2 Analyze Highest Requests Location . . . 63
5.4.1.3 Analyze Highest Requests Rate Tendency . . . 63
5.4.2 Self-analyzing Function in System Integrity Scenario . . . 64
5.4.2.1 Analyze Client Activity . . . 65
5.4.2.2 Analyze Clients Tendency . . . 65
5.4.2.3 Analyze Clients Location . . . 66
5.4.2.4 Analyze Log File's Growth Rate . . . 67
5.5 Self-planning Function (Using Cfengine for Decision-making) . . . 67
5.5.1 Decision-making in Service Optimization Scenario . . . 68
5.5.2 Decision-making in System Integrity Scenario . . . 69
5.6 Action Scripts . . . 70
5.6.1 Migrating Virtual Machine . . . 70
5.6.2 Back up Files to Local Redundancy Storage . . . 71

6 Result 3: Measurement and Analysis 73
6.1 User Requests Analysis . . . 73
6.1.1 User Request Trends and Their Summary . . . 73
6.2 Self-management Log File Analysis . . . 75
6.2.1 Log File Output . . . 75
6.2.2 Highest User Requests Percentage Trend . . . 77
6.2.3 Highest User Requests Percentage and the Corresponding Policy 78
6.3 Virtual Machine Behavior Analysis . . . 79
6.3.1 Measuring the Round Trip Time . . . 79
6.3.2 Using Round Trip Time to Measure Virtual Machine's Migration 80
6.3.3 Virtual Machine Behavior in Scenarios . . . 80
6.3.3.1 Virtual Machine Migration in Service Optimization . . . 81
6.3.3.2 Virtual Machine Migration in System Integrity . . . 82

7 Discussion 85
7.1 Simulation versus Realistic of Experiment Environment . . . 85
7.1.1 The Reason Why to Implement the Experiment in Realistic . . . 85
7.1.2 Practical Problems . . . 85
7.2 Defining Good Backup . . . 86
7.3 Keeping the Service Available . . . 87
7.4 Optimizing Generator Scripts . . . 87
7.5 Problems Caused by Time Shift in Virtual Machine . . . 87
7.6 Using Cfengine for Decision Making . . . 89
7.7 Problem Caused by Update IP Address . . . 89
7.8 The Probability of Migration . . . 90
7.8.1 Analyze Total Request Trend . . . 90

7.8.2 Highest Request Percentage 91
7.8.3 Highest User Activity Boolean 91
7.8.4 The Probability of Migration 92
7.8.5 Other Parameter for Policy 93
7.9 System Convergency 96
7.10 System Drawback 96

8 Conclusion **99**
8.1 Future Work 100

A Network and Environment Implementation **105**
A.1 MLN Configuration File for Creating Virtual Machine and Networks . . 105
A.2 MLN Configuration File for Virtual Machine Migration 106
A.3 MLN Configuration File for Virtual Machine Backup 106
A.4 VTun Configuration File for VPN Tunnel 107

B Generating User Requests **109**
B.1 Generator for 24 Hours Period Sine Curve 109
B.2 Generator for 8 Hours Period Sine Curve 110

C Self-management Functioning Scripts **113**
C.1 Self-monitoring Function 113
C.2 Self-analyzing Function 114
C.2.1 Self-analyzing in Service Optimization 114
C.2.1.1 Analyze Highest Requests Percentage (user_percentage.pl) 114
C.2.1.2 Analyze Highest Requests Location (user_code.pl) . . . 115
C.2.1.3 Analyze Highest Requests Tendency (user_boolean.pl) . 117
C.2.2 Self-analyzing in System Integrity 118
C.2.2.1 Analyze Clients Connectivity (storage_clients.pl) 118
C.2.2.2 Analyze Clients Tendency (storage_boolean.pl) 118
C.2.2.3 Analyze Clients Location (storage_code.pl) 119
C.2.2.4 Analyze Log File Growth (storage_logfile.pl) 120
C.3 Decision-making Function (Cfengine Configuration File) 121
C.3.1 Decision-making in Service Optimization 121
C.3.2 Self-analyzing in System Integrity 122
C.4 Execute Actions 123
C.5 Virtual Machine Migration 123
C.6 Virtual Machine Backup 124

D Measurement and Analysis **127**
D.1 Filter the Web Service Log File 127
D.2 Using Round Trip Time to Measure Virtual Machine Migration 128
D.3 Synchronize the Time Stamp in Log File 128

List of Figures

1.1 Case Example: Global Server Setup 3
1.2 Case Example: Initial Service Setup 4
1.3 Case Example: Moving Virtual Servers 4

2.1 Full Virtualization Architecture 11
2.2 Operating System Layer Virtualization Architecture 12
2.3 Hardware Virtualization Architecture 13
2.4 Relationship between Flexibility and Performance 14
2.5 Migration Model 14
2.6 Autonomic Model 15
2.7 An Agent Based on Production Systems 19
2.8 Model of a STRIPS-like Operator 19
2.9 Structure of a Component and its Terminals 20
2.10 Virtual Private Network Model 21
2.11 XEN Virtualization Architecture 22
2.12 Migration in Xen 23
2.13 Comparison of DAS, NAS and SAN 24
2.14 Convergent Policy 27

3.1 Architecture Overview 33
3.2 Global Server Network and Storage Infrastructure 35
3.3 Virtual Machine Migration 37
3.4 Virtual Machine Storage Sharing after Migration 37
3.5 Virtual Machine Self-management through Migration 38
3.6 Scenario 1: Self-managemt in Service Optimization 40
3.7 Scenario2: Self-managemt in System Integrity 42

5.1 Virtual Machine Self-management Operational Strategy 50
5.2 Self-management in Service Optimization Process Strategy Written in BRIC 53
5.3 Self-management in System Integrity Process Strategy 54
5.4 Individual Client Generators 56
5.5 Generator without Sleeping Time 58
5.6 Generator with Different Sleeping Time 58

5.7 Generator with the Same Sleeping Time 59
5.8 User Requests Track Created by Generator 60
5.9 Changes in the User Boolean Log File 64
5.10 Changes in the Storage Boolean Log File 66

6.1 User Request Trends and Server Side Summary 74
6.2 Log File Output for the Service Optimization Scenario 75
6.3 Log File Output for the System Integrity Scenario 76
6.4 Highest Request Percentage Trend 77
6.5 Highest Request Percentage and Relative Policy Control Migration . . . 78
6.6 Policy Influence Virtual Machine Behavior 78
6.7 Using Round Trip Time to Measure Virtual Machine's Migration 80
6.8 Virtual Machine Migration in Service Optimization 81
6.9 Virtual Machine Migration in System Integrity 82

7.1 Highest User Activity Boolean Analysis 92
7.2 Probability of Migration 93
7.3 Total User Activity Boolean Analysis 95
7.4 Probability of Migration with Total Request Boolean as Policy 95
7.5 System Convergency 96

List of Tables

3.1 Information of the Servers 33
3.2 Server Task Assignments 36
3.3 Software and OS Versions 36

4.1 MLN Project for Creating the Network and System 44
4.2 MLN Project for Updating Projects 45

5.1 Developed Scripts and Functions 51
5.2 New Location for Storage Backup Policy 67

Chapter 1

Introduction

1.1 Motivation

Today's networks and networked services are becoming increasingly global, a trend stemming from the explosive evolution of the Internet since the 1980s. Not surprisingly, as networks grow, the amount of human power required for system management increases with them. Large-scale networks and rising complexity in topologies make systems difficult to control manually by human beings. Due to this rapidly extending tendency in global systems, administrators will not be able to install, configure, manage, maintain and optimize their systems manually any more.

To resolve this problem, one possible solution is to implement autonomic technologies into system administration. With this type of technology, systems can manage themselves to attain what can be referred to as the expected *ideal state*, set forth in a pre-defined configuration by the system administrators. Through the benefit of substantially decreased management overhead, the system administrators will be able to get rid of the tedious and repetitive maintenance tasks and rather go deeper into the design and high level management of the system.

Virtualization brings more flexibility into system administration and changes the general way that people perceive the components of systems. Different virtual machines can be combined into one physical server, and individual servers can change their own roles and properties on the fly. From the industries' view, virtual systems can reduce a considerable amount off of the hardware costs, since one single server can provide a large amount of low-utilized virtual servers for customers. Virtual machine recovery and recycling tasks are easily managed as well. This means that after finishing its tasks, the useless, under-utilized physical computers do not need to be thrown away. Instead, the old virtual machine can be destroyed and the released resources can be reallocated for another virtual machine later. Thus, reducing hardware costs becomes one of the main reasons why virtualization is popular in the industry.

In order to improve the efficiency of system administration, this work mainly focuses on how to combine ideas from the area of autonomic computing with the more well-established concepts from virtualization technology. In other words, the main

challenge for this project is how to introduce and implement self-management computing techniques into virtual machine-based environments. To illustrate the usefulness and value of this approach, the following section describes a fictional example where self-management combined with virtualization-based systems applies to a popular service.

1.2 Example Case: A Global Fabric for Service Placement

1.2.1 Challenge

Lucas is a big fan of on-line games. After 20 years of experience in on-line gaming, he decides to build his own company to supply servers for game service providers. Based on his own experience and market research, a popular on-line game always covers the whole world with thousands or even millions of users playing it at all times. Especially for real-time action games, the speed of communication between client and server is an important factor to attract the most users. In order to support a high-grade service for all players, response time, network capacity and connection stability are crucial factors for game servers. Lucas hopes that these issues can be resolved by reducing the distance between players and their respective game servers. Normally, client to server proximity is directly related to the perceived overall quality of the offered services. This means the nearer servers are located, the better services users can get. Thus, Lucas decides to set up a global networking system for his game service providers.

The game service providers – Lucas' customers – can not expect the users of one area to keep awake and play the game around the clock. Still, they pay for services that have to be kept running 24 hours a day, 7 days a week. The continuous costs of maintaining servers compared with the alternating income from the users do not sound like a optimal idea to run their business. How to take advantage of time zones instead of wasting resources of the servers becomes a challenge for Lucas.

1.2.2 Approach: A Global Infrastructure with Manual Management

Lucas decides to set up a global network to provide game service companies with virtual servers. In this case, each service is located in a virtual machine, which can be migrated to different locations based on time of day within various time zones. Thus, game service providers can simply pay for the costs of virtual servers, and use the whole global network in order to be close to where the customers are currently active.

As shown in figure 1.1, global servers are located in Canada, USA, Brazil, Norway, Russia, India, China, and Australia.

After setting up network connections between these servers, Lucas builds a virtualization infrastructure on top of it. The virtual machines are more flexible than the physical ones, because they can be migrated to any given server according to the observed usage characteristics. This also means that game service providers can use the same installation and configuration environment at different locations by moving vir-

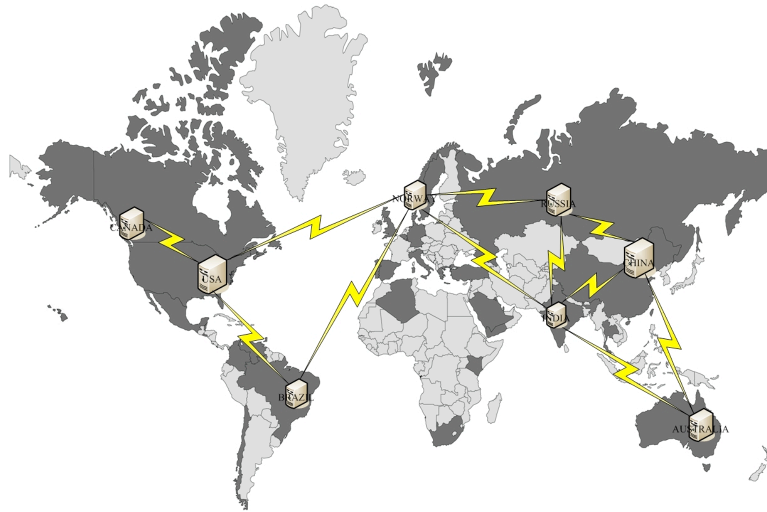


Figure 1.1: Case Example: Global Server Setup

tual machines in a distributed network. In this case, these virtual machines can be sold as "service carriers" to the service providers.

This infrastructure has benefits for both Lucas and his customers. For Lucas, he only needs to maintain the fundamental virtual environment and the network connections. The game service providers need only pay the service carrier fees, and can then use the entire global network to let the virtual machines move about to where they are most suited. Further, the game service providers can also make informed decisions on how many virtual servers they need, and which kind of services they want. For instance, being able to move the virtual servers, copying the virtual servers elsewhere, etc.

1.2.3 Analysis

Take the Asia-Europe continents as an example. First, a service carrier is located in Norway. European players connect to the Norway-based server, while players on the other continent are sleeping. The initial setup is shown in figure 1.2.

Some hours pass by and users in Russia and India wake up and get ready for a fierce battle in the game. However, some players in Norway are still in the middle of their wars. Compared with the number of users at each location, Russia now has two thirds of the total requests of all the players in those three places. In order to provide better services to most of the users, the provider decides to move the servers from Norway to Russia. As a result, the small amount of users from Europe and India have to send their requests to Russia to be able to continue play. Moving the virtual machine from Norway to Russia is shown in figure 1.3

As shown in figure 1.3, the service provider can provide better services for its users by moving the virtual servers. An added benefit is that the provider can also earn more

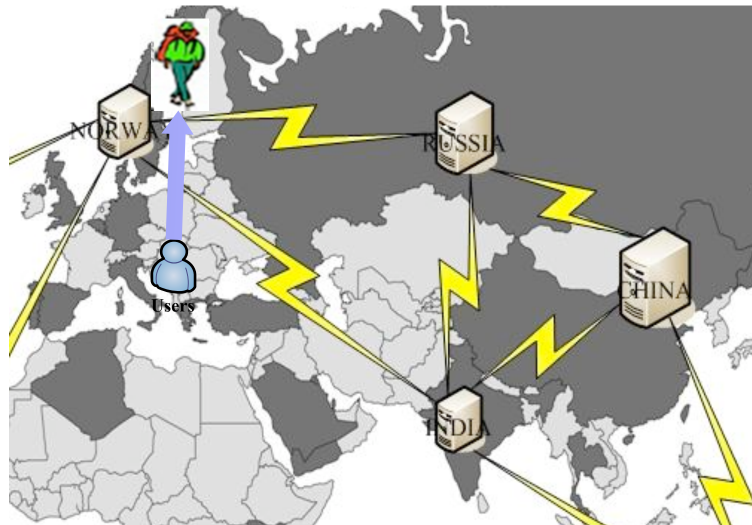


Figure 1.2: Case Example: Initial Service Setup

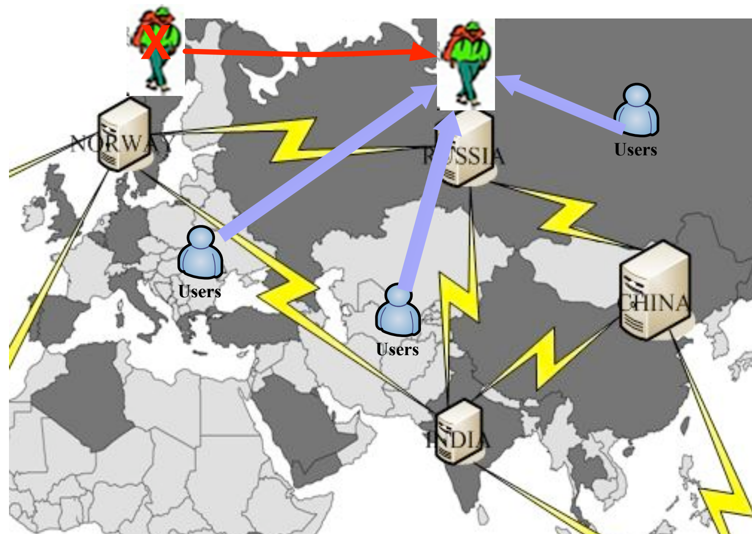


Figure 1.3: Case Example: Moving Virtual Servers

money from the potentially increasing number of users, due to the high quality user experience. Moving a service should be done on a rational decision, typically based on a defined set of parameters deemed important by the service provider: CPU and I/O load, bandwidth quality, latency, storage utilization, user location, etc. In this case, the decision is made by the providers. The decision-making process requires continuous

monitoring which again means that substantial human resources are involved. To improve the performance, this system should be modified to utilize autonomic features, such as self-management.

1.2.4 Improved Approach: Global Infrastructure with Autonomic Virtual Machines

Imagine Lucas having great success running his company, and the market for virtual machine-based services becomes more and more popular all over the world. He will soon end up with hundreds or even thousands of virtual machines in need of configuration and management. If these operational tasks simply rely on manual work, the overhead would potentially become disastrous to Lucas' business; spending millions on new hires and operations can quickly lead to bankruptcy.

However, as another solution, he would think about implementing self-management into this system, modifying the virtual machines to manage themselves to a large degree. With self-managing functions, virtual machines should be able to monitor their environment, analyze available information, and then modify themselves to adapt to varying situations.

1.3 Problem Statement

The case above shows a possible future of how to manage a network and servers, which is to set up a global, self-managed network environment. Two main techniques are included in this infrastructure: virtualization and self-management. Virtualization brings flexibility to system administration, as well as a good platform for deploying autonomic technology. Self-management can potentially give administrators less overhead in managing their networks and servers.

Therefore, the problem statement in this project is:

1. *How to efficiently introduce and implement self-management in two essential virtualization-based system administration tasks:*

   (a) *service optimization*

   (b) *system integrity*

   in order to simplify and minimize the maintenance and operational costs for the system administrators?

"Efficiency" here means how to find a simple approach to implement self-management functions based on the regular administration tools. Therefore, this project can be intorduced as a simple example for other system administrators to optimize their network and system in realistic.

Accordingly, this project is implemented as a realistic system, meaning that instead of simulating the global infrastructure in a local network, the servers included in this project are placed around the world. The purpose of this, as opposed to an ideal lab environment, is to provide a realistic variation in environmental parameters such as network stability and performance. This unpredictability leads to more complex considerations and decision-making. Also, the implementation and output of the system is closer to real-world scenarios, which again can yield immediately applicable results.

Service Optimization

The first task, service optimization, is a typical scenario in system administration. There are several ways to achieve this task. Examples include improving system stability, minimizing costs, reducing service reaction time, etc. Since the definitions of stability and cost vary depending on different requirement and contexts, this paper mainly focuses on reducing the service reaction time or latency, which is commonly understood as the round trip time between the users and the servers.

Other than the obvious step to avoid local bottlenecks, reducing response time can typically be significantly reduced by minimizing the distance between client and server. Specifically, the distance here means the physical distance, which would be a measure primarily of geographical proximity. In general, this would also be related to the routing distance, that is the number of hops that data packets must traverse to reach their destination. However, it is not easy to calculate the hop metrics to optimize the best path between two end-nodes. Therefore, the physical distance is represented as the the round trip time from the users to the servers.

In this scenario, the virtual machine will run a web service daemon and monitor usage as part of its environment. It will then analyze the usage patterns and decide on actions to eventually improve the service performance.

System Integrity

The second task in this project is about maintaining system integrity. Protecting the information contained within the system and its services is part of every operational agreement. Backing up the files to an external storage is a general and common way to ensure basic system integrity.

In the traditional approach, backups are run at a specific time determined by the system administrators. This specific time is normally at night when the system is likely to be idle to prevent the backup operations from interfering with more critical production processes.

In the self-managed systems, the virtual machines can make decisions by themselves when and where to do backup according to the observed characteristics in the monitored environment. To define a good backup schedule, two kinds of methods are discussed here.

1. Do the backup before an operation considered to be risky.

2. Back up the data when it has changed sufficiently since the previous backup. Do this at a time with few clients connected to the services.

This work mainly focuses on the latter approach. In this scenario, the virtual machine will monitor the state of the data for the service it is running. If the data has changed beyond a threshold level, the virtual machine will make a decision on when and where to back up its data before continuing to run the service.

As a conclusion, the main goal of this project is to design and implement a self-management system based on a global virtual infrastructure, which eventually would mean that each single virtual server has the ability to make decisions to dynamically migrate or relocate resources, adapting to the changes of its environment.

1.4 Thesis Outline

This document will be structured as follows: the background and related technologies are introduced in chapter 2. Chapter 3 lists the objectives for this work and also explains the methodology which are mainly used in this project. Chapter 4, 5 and 6 are the results of this project, including system and network setup, experiment implementation, and the results measurement and analysis. The discussion in chapter 7 summarizes the decision-making through all the experiment. Especially analyze the probability of virtual machine migration, in order to predict the virtual machine behavior and make suitable policies. The conclusion and future work is presented in chapter 8.

Chapter 2

Background

2.1 Virtualization Technology

2.1.1 Introduction to Virtualization

In June 1959, on the International Conference on Information Processing at UNESCO, New York, Christopher Strachey published a paper entitled "Time Sharing in Large Fast Computers" [1]. In this paper, he introduced the concept of "time sharing" which became popular and much discussed in the 1960s. According to Strachey, time sharing means keeping several applications running simultaneously. Later on, in the 1990s, John McCarthy of Stanford University explained time sharing as "an operating system that permits each user of a computer to behave as though they were in sole control of a computer, and it is not necessarily identical with the machine on which the operating system is running" [2].

During the time sharing era, applications were not running completely isolated, and they interrupted each other. Further research about how to improve the reliability of the system was introduced. In order to isolate the applications, IBM first introduced and implemented virtualization in mainframe computers in the 1960s. The main idea was to partition the mainframe for multiple tasks, which were to be known later as *virtual machines* later on. The most known IBM virtualization system is IBM VM/370. [3]

Nowadays, virtualization is a popular topic in businesses and recently even for home use. The concept of time sharing has not changed significantly over the last 50 years, as we today loosely define virtualization as a methodology to combine multiple execution environment running simultaneously into one single computer by sharing the available hardware resources. So in practice, virtualization allows several different operating systems to run individually within one single physical host at the same time.

2.1.2 Advantages and Disadvantages of Virtualization

Taking the benefits of grouping multiple virtual machines into one single computer, the main use of virtualization is its highly favorable effect of reducing costs. It enables sav-

ings on hardware, environmental costs, management, and administration of the server infrastructure.

According to the article "Virtualization Basics" [4], there are 6 top reasons to adopt virtualization software:

- **Optimize infrastructure**: Since all the virtual machines are sharing the same hardware, virtualization allows administrators to prioritize few virtual machines for the significant functions and lower the others for common utilization. This helps to optimize the network and system infrastructure, and it also helps to improve utilization of the hardware.
- **Reduce physical infrastructure cost**: Numerous low-performance services can be configured as virtual machines running in one physical server. As a result, the number of servers, the space, power and cooling requirements are efficiently reduced.
- **Improve operational flexibility**: Virtualization leads to more possibilities for administrators to design, configure, monitor, maintain, and manage systems [5]. For example, it is possible to change the hardware specifications of virtual machines on the fly. And paired with virtualization, autonomic technology is simpler to implement, which again has the potential to rid administrators of redundant manual tasks.
- **Increase application availability**: Live migration of virtual machines increases application availability. Live migration can keep services running continuously while moving the platform to another server.
- **Improve desktop manageability**: Users can create and manage virtual machines both locally and remotely using a wide range of virtualization implementations.
- **Increase network and system security**: Virtualization allows service separation by running one service isolated on each virtual machine. As a result, if one service or one virtual machine is compromised, others will not be directly vulnerable.

Although virtualization has several obvious advantages, there are clear disadvantages as well.

The most important disadvantage is the physical host becoming a single point of failure. Hardware failure will cause virtual machines to go down as well. Now, instead of losing one service, all services are lost.

2.1.3 Classification of Virtualization

There are various approaches to implementing virtualization. They can be categorized depending on the different technological designs. From an architecture point of view, according to [3] and [6], the three major categories of virtualization techniques are:

- Full Virtualization
- Operating System Virtualization
- Hardware Virtualization (Paravirtualization)

Full Virtualization

Full virtualization means complete emulation of all hardware devices. As shown in figure 2.1, the virtual machine manager (VMM) runs on top of a host operating system, and creates the virtual hardware for virtual machines. As a result, in each virtual machine, the guest OS is running on virtual hardware, and the applications are running on top of the guest operating system.

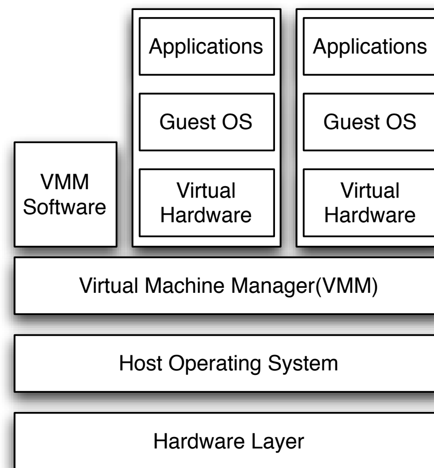


Figure 2.1: Full Virtualization Architecture

The main advantages for this architecture is an easy setup, and that guest operating systems can be installed and used directly on the physical hardware without modification. The disadvantage for this approach is that this complete emulation of computer hardware demands more resources from the VM Server. Accordingly, an operating system running in full virtualization mode could be performing rather poorly. A research [3] shows that the performance can be up to 30% less than running directly on the hardware.

The most popular program that uses this approach is VMware Workstation [1], while other programs including Parallels [2] and Virtual PC [3].

[1]http://www.vmware.com/products/ws/
[2]http://www.parallels.com/
[3]http://www.microsoft.com/windows/products/winfamily/virtualpc/default.mspx

Operating System Virtualization

Figure 2.2 shows the basic concept of OS-level virtualization. Here, the host operating system is virtualized, rather than the hardware. The virtual machines are running parallel on their parent virtualized operating system.

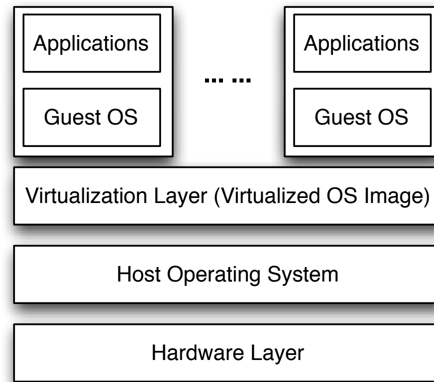


Figure 2.2: Operating System Layer Virtualization Architecture

Operating system virtualization provides very high performance, exactly the same as when running on the physical server. In this technology, system administrators can assign resources both when creating a VM as well as on the fly. A big disadvantage, however, is that all the virtual machines have to run the same operating system image as the physical server. Thus, for example, running Windows on top of Linux is impossible. As a further limitation, migration of virtual machines is limited to hosts running the same operating systems.

There are some products using this approach, such as Virtuozzo [4], OpenVZ [5], Solaris Containers [6], and Linux VServer [7].

Hardware Virtualization (Paravirtualization)

When the computer was developed, the hardware architecture was not developed with virtualization technology. Paravirtualization is introduced instead of real hardware virtualization, and it aims to implement CPU virtualization. In paravirtualization, the host operating system exposes hardware through an abstracted software layer, virtual machine monitor (VMM), to the guest systems. The guest systems must be modified to communicate with the hardware through this interface, thus limiting deployment

[4] http://www.parallels.com/en/products/virtuozzo/
[5] http://openvz.org/
[6] http://www.sun.com/bigadmin/content/zones/
[7] http://linux-vserver.org/Welcome to Linux-VServer.org

on legacy or proprietary operating systems. The architecture of paravirtualization is shown in figure 2.3.

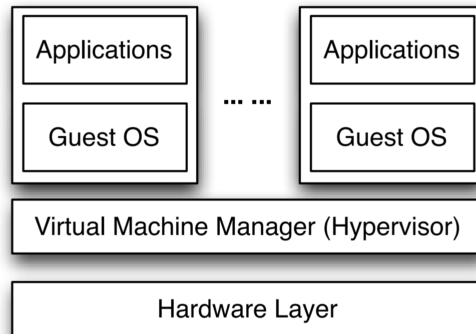


Figure 2.3: Hardware Virtualization Architecture

Paravirtualization is different from full virtualization because it provides the virtualized operating system to see both real and virtual resources. A paravirtual operating system is typically able to use virtualized hardware and also execute user space CPU instructions unmodified and un-virtualized on the host CPU, leading to near native execution speeds under good conditions [7].

Two contemporary and popular products that are based on paravirtualization, are VMWare ESX Server [8] and Xen [9]. Since Xen is an open and free implementation, it is the platform of choice for this project. Coming chapters will focus more on Xen, its details and how to deploy it.

Comparison of Different Virtualization Technologies

One informal conclusion to draw from the discussion above, is that an increase in flexibility and features provided by virtualization is inversely proportional to the average performance gain of the virtualized systems.

Illustrated in figure 2.4, full virtualization has the highest flexibility, while OS virtualization yields the highest performance. The compromise between the two, hardware virtualization (paravirtualization), has a good balance between flexibility and performance. This is likely one of the main reasons for its popularity and high deployment rates.

[8] http://www.vmware.com/products/vi/esx/
[9] http://www.xen.org/

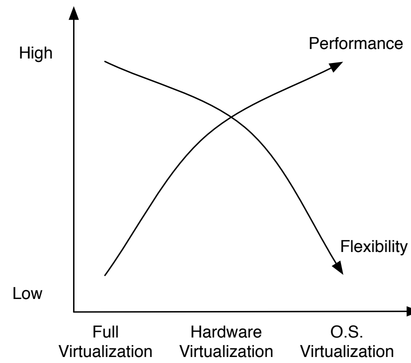


Figure 2.4: Relationship between Flexibility and Performance

2.1.4 Migration of Virtual Machines

Migration refers to moving of virtual machines from one physical server to another. There are two main ways of executing migrations, termed live and cold migration. Live migration means migrating virtual machines while they are still running, without interrupting any processes. This capability opens up a variety of practical uses.

For example, consider an IT company that needs to provide an uninterruptible web service for its clients. In case of server OS or hardware upgrades, using live migration, the service can be moved elsewhere during maintenance without affecting service availability. Therefore, maintenance can be done without downtime.

The minimum requirement for migration is two physical servers and a shared storage medium [8]. All the servers included need to run the same base hardware. A simple illustration of migration is shown in figure 2.5.

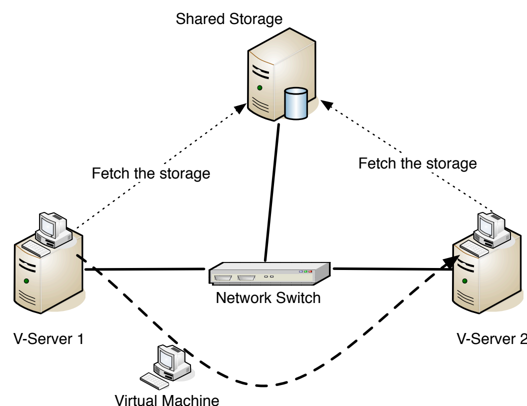


Figure 2.5: Migration Model

Firstly, the virtual machine is set up on V-Server 1. The virtual machine image is located on the local network shared storage server. During migration, the virtual machine is transfered to V-Server 2, while keeping the image unaltered on the storage server. Evidently, the shared storage is one of the most essential factors for migrating virtual machines.

2.2 Autonomic Computing and Self-management

2.2.1 Autonomic Computing

IBM released a paper in 2001 entitled "The Vision of Autonomic Computing" [9]. The author argues that the ever-increasing interconnectivity and diversity of systems leave system administrators with excessively massive and complex issues to deal with. It is foreseen that even the most skilled system integrators will not be able to install, configure, optimize, maintain and merge these systems any more. The only possible solution remaining is to develop autonomic computing systems, which can manage themselves.

The concept of autonomic computing is introduced here as computing systems that are able to manage themselves based on high-level objectives defined by administrators. This means that autonomic computing does not completely replace the role of system administrator. Instead, the task of system administration will gradually move to a higher level, defining the abstract implementation and policies for the autonomic system to attain. Figure 2.6 shows the structure of an autonomic element, as IBM perceives it.

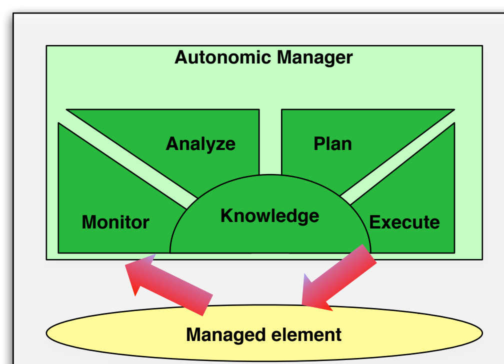


Figure 2.6: Autonomic Model

In this model, the autonomic manager has four steps in its autonomic cycle. Firstly, the autonomic manager monitors the managed element, and then analyzes any obtained information. Next, based on its knowledge base, it plans the possible solutions and actions. Finally, it executes its plan to modify the behavior of the managed element. In general, an autonomic element is composed of one or more managed elements,

which in turn can contain similar or different parameters. However, all these managed elements should be controlled by one autonomic manager.

2.2.2 Self-management

In order to manage themselves, the autonomic systems need to observe situations and events, "sense" their environment, and then make decisions about which actions to execute and when. The goal is to keep the system within defined limits as defined in a behavioral schema. These behaviors can be categorized as different functions of the autonomic systems, the essence of autonomic computing systems is self-management. According to [10], the definition of self-management is defined as:

> *"According to high-level management policies, the system should be able to reconfigure itself to handle changes in its environment or requirements without human intervention."*

At the lowest level, self-management means that the system should be able to automatically adjust itself and related components to handle common and frequent events. For instance, adding and removing nodes, load balancing between nodes, etc. At a higher level, self-management includes several system properties, and actions need to be executed in different system levels, e.g. self-configuration, self-healing, self-optimization and self-protection.

- **Self-configuration**: In accordance with the predefined policy by system administrators, autonomic systems can configure themselves automatically to reach desired states with minimal human intervention. This means the service architecture will continue to work when nodes are added or removed during execution. When a new node is introduced into an autonomic system, it will automatically learn about its environment, and then integrate itself. Meanwhile, if a node is removed, it will notify other nodes so that they can modify their own behavior to adapt the new situation.

- **Self-healing**: Instead of relying on manual interaction for identifying and debugging failures, an autonomic system can detect, diagnose, repair, and sometimes predict the problems and failures on its own, regardless of the origin and nature of the problem. The purpose of self-healing is to pull the nodes back from the wrong states into the desired states. This kind of behavior is called convergence [11]. If a system is fully convergent, whatever the initial state the system has, it will manage itself back to the predefined state.

- **Self-optimization**: After an extended period of gaining "experience", an autonomic system will be able to learn, and then continuously evaluate and change its run-time parameters to improve its operation. Experience and log memory are kept as a knowledge base by autonomic systems, and they can use it to find, verify and apply appropriate changes to upgrade their functionality.

- **Self-protection**: Like using firewalls and intrusion detection tools to protect systems from attacks, autonomic systems can benefit from self-protection as well. The goal of a self-protecting environment is to provide the right information to the right users at the right time [12]. The right user here means the one has authorized relative permission. Autonomic systems will be self-protecting in two senses [9]. One way is to defend the system as a whole against attacks or vulnerabilities left open by self-healing. The other one is to predict the problems based on early reports and take actions to avoid them.

2.3 Multi-agent Systems

2.3.1 Introduction to Multi-agent Systems

Multi-agent systems are introduced as an alternative to model and simulate complex environments that contain various connected or related entities. Such systems have the ability to represent the behavior, relationships and communication between entities in the same environment.

Multi-agent systems are composed of individual agents or entities. Each agent has independent behavior and characteristics. They function and influence on the environment differently. Because the agents are independent, the new types of entities or agents with their own model of behaviors can be added into the system easily at any time. Therefore, multi-agent systems are recognized as scalable and flexible. More details are introduced and discussed in the book "Multi-Agent Systems", written by Jacques Ferber [13].

2.3.2 Concepts in Multi-agent Systems

Agent

In an environment, agents are physical or virtual entities that can communicate directly with each other [13]. Their behavior can affect and modify the surrounding world, and their actions can also be influenced and limited by the environment.

Agents have the ability to attempt to optimize their own behaviors to achieve goals, which means they have self-management functions and do not need to be controlled by second or third-party entities. Thus, they have the freedom to make their own decisions, like accepting or rejecting requests from the environment. Compared with other similar concepts, such as objects and procedures, the agents have more self-management functions.

Multi-agent Systems

A Multi-agent system consists of a number of agents. Different relations link agents together, which makes a complex environment. Special-purpose agents within the multi-agent system can perceive, create, destroy and modify passive agents in the environ-

ment. Each single agent has functions for making itself adapt to environmental or operational changes, and also for evolving its capacity in a growing environment. In this context, multi-agent system has the following characterizes [13]:

- Distributed architecture.
- Caused reason and relative options happen locally.
- Agents are self-managed and adaptable.
- Agents can integrate, appear or disappear at any time.
- Ability to handle capacity growth.

2.3.3 Multi-agent Systems Models

Multi-agent systems are used to simplify the complicated behaviors of servers by representing the behaviors, relationships and communications between the entities in an environment. However, since all the agents are independent and have their own characteristics, this still makes multi-agent system a complex system. Therefore, models are needed in analyzing actual system behavior to be representative in programming languages. Also, models can simplify the main core and the representation of the programs in the following ways [13]:

- Eliminate unnecessary details in the programs
- Concentrate on the programming implementations
- Convert machine language and data structures into easy, human-friendly language
- Modify a system's behavioral consequences into a programmable data flow
- Separate complex system components into clear and easily analyzable parts

A Model of Production Systems

A production system (or rule-based system) is defined as the combination of a database, a production rule base and an interpreter – the inference engine [13]. They are generally given on the form:

```
if <list of conditions> then <list of actions>
```

where the `<list of conditions>` refers to database parameters, while the `<list of actions>` refers to database actions, e.g. add or delete conditions.

In the context of a multi-agent system, each agent is represented as the model in figure 2.7 based on the production system concept. If several rules or actions are valid under the same conditions, they are in conflict and the agent will pick up the rule with the highest priority, which in turn depends on the specific parameters predefined in the system.

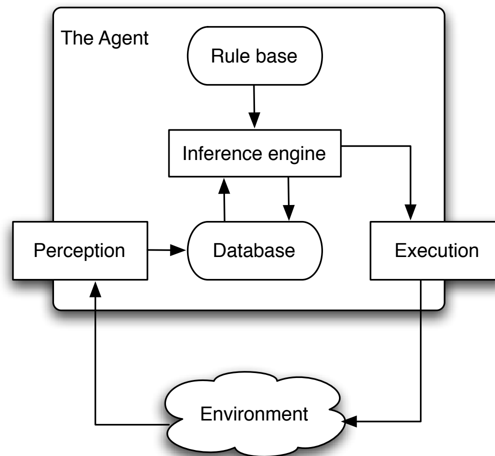


Figure 2.7: An Agent Based on Production Systems

Modeling Actions

In 1971, Fikes and Nilsson put forward a representation as a response to some planning problems, which is known as "STRIPS-like representation" [13]. This is illustrated in figure 2.8.

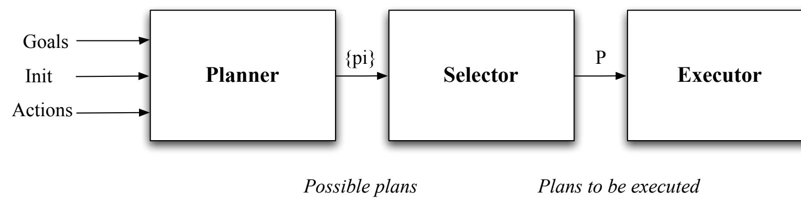


Figure 2.8: Model of a STRIPS-like Operator

A planner will produce a set of possible plans according to the collected initial states, final goals and executable actions. The initial states, final goals are represented as `<list of conditions>`in the multi-agent model language, and the possible actions mean `<list of actions>`. However, only one plan will be chosen and executed by the selector depending on the different priorities.

Modeling of Multi-agent Systems in BRIC

In multi-agent systems, Jacques developed a formalism BRIC (Basic Representation of Interactive Components) to give an operative representation of the functioning of

agents and the multi-agent systems. They use electronic circuits to describe the agents connectivities, also called components here.

Each component can be considered as a module. They interact by means of communication links established between the terminals. Figure 2.9 shows a simple example of BRIC model.

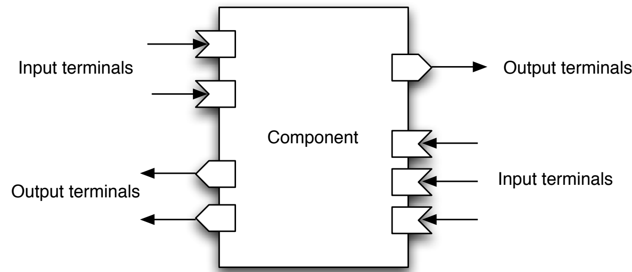


Figure 2.9: Structure of a Component and its Terminals

BRIC model will be used to describe the virtual machine functioning in the result chapter.

2.4 Virtual Private Network (VPN)

VPN can be explained as a private network that uses a public data communication infrastructure (usually the Internet) to connect remote sites or individual users together. The most important usage of VPN is that it can provide users the same community capabilities as the underlying network with much lower costs and better security.

Figure 2.10 shows an infrastructure of a virtual private networks which is normally used by companies. Three local area networks (LANs) are included in this wide area network (WAN).

2.4.1 VTun

VTun [14] is an open source networking application for setting up VPNs over TCP/IP networks. As described in its home page, VTun is the easiest way to create Virtual Tunnels over TCP/IP networks with traffic shaping, compression, and encryption. It was originally developed by Maxim Krasnyansky, and is currently maintained by Bishop Clark [15].

VTun is a client-server system. It creates a single connection between two machines. The client connects to a specified port on the server and multiplexes sessions over the initial connection. When the client machine sends a TCP connection to the server port, the VTun connection is initiated. And then, a UDP connection will be sent back to the client machine if it is requested.

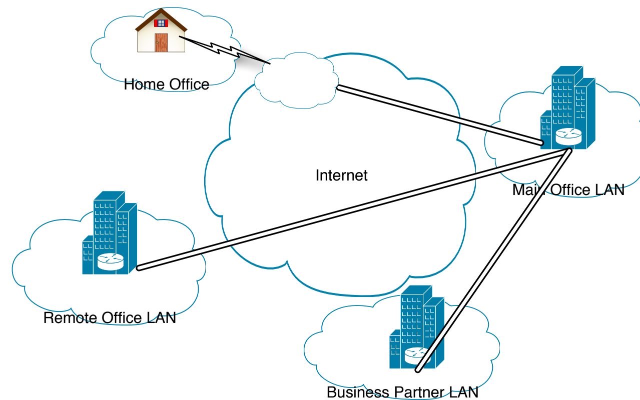


Figure 2.10: Virtual Private Network Model

Since the VTun creates the connection between two nodes, it means that both LAN networks have the same security after connecting, normally, the lower one. As such, it is very important to protect the LAN on each side. As an addition, VTun provides a private shared key to protect the connection, which means the server and client binary applications can be set up without SSL support. Consequently, when using these applications between two nodes, the encryption is very trivial to decode.

2.4.2 Creating a VTun VPN

To build a VTun, two configuration files need to be created: vtund-server.conf and vtund-client.conf. The tunnels are brought up using the following commands:

1. Start the server process on the server machine as root:

```
#The -s option tells vtund to run as the server
 vtund -f /usr/local/etc/vtund-server.conf -s
```

2. Run from the client to access server by using OpenSSH:

```
#e.g. the open port is 5000
 ssh mydesktop.work.com -L 5000:localhost:5000
```

3. Start the tunnel on the client side, run command as root:

```
#The my_tunnel parameter tells what tunnel is being created,
#and the localhost specifies the hostname of the VTun server
 vtund -f /usr/local/etc/vtund-client.conf my_tunnel localhost
```

4. To check if the tunnel is created, on each machine run the command "`ifconfig`".

2.5 Xen

As introduced in section 2.1.3, page 10, there are three main types of virtualization technologies. As a paravirtualization technology, Xen is a virtual machine monitor tool for the x86 architecture. Xen is open source software released under the GNU General Public License (GPL), and developed at the University of Cambridge. More information about Xen can be found in the overview paper "Xen and the art of virtualization" [7].

2.5.1 Xen Architecture

Since Xen is a paravirtualization technology, it provides a slightly altered hardware interface to the virtual machines. It works by dividing the host and guest operating systems into different parts, called domains, that run on top of this special hypervisor hardware interface. Domain0 refers to the host operating system, which is separated from DomainU (unprivileged) guest operating systems. Domain0 has administration rights and can for example create, shut down or reboot DomainUs. The Xen virtualization architecture is shown in figure 2.11.

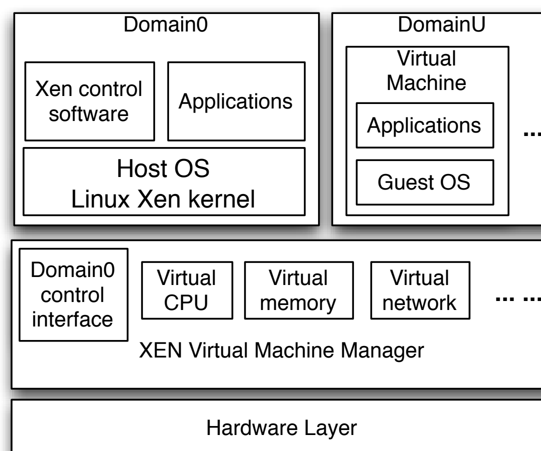


Figure 2.11: XEN Virtualization Architecture

2.5.2 Migration in Xen

Live migration allows continuous service operation by moving virtual machines to other physical machines if the current physical machine needs to be shut down. It can also provide load balancing by migrating the virtual machines to a high performance server from the current congested server. Since the entire operating system and all the applications are migrated as one unit, live migration reduces some of the configuration

complexity. More details of live migration are discussed in the paper "Live Migration of Virtual Machines" [16]. This paper was written in 2005, mainly focuses on making live migration a practical tool even for servers running interactive loads. They sufficiently implemented performance with minimal service downtime, which is 60ms.

Live migration has two main requirements:

1. shared storage

2. similar CPU architecture (identical features)

The process of live migration on an architectural level is shown in figure 2.12.

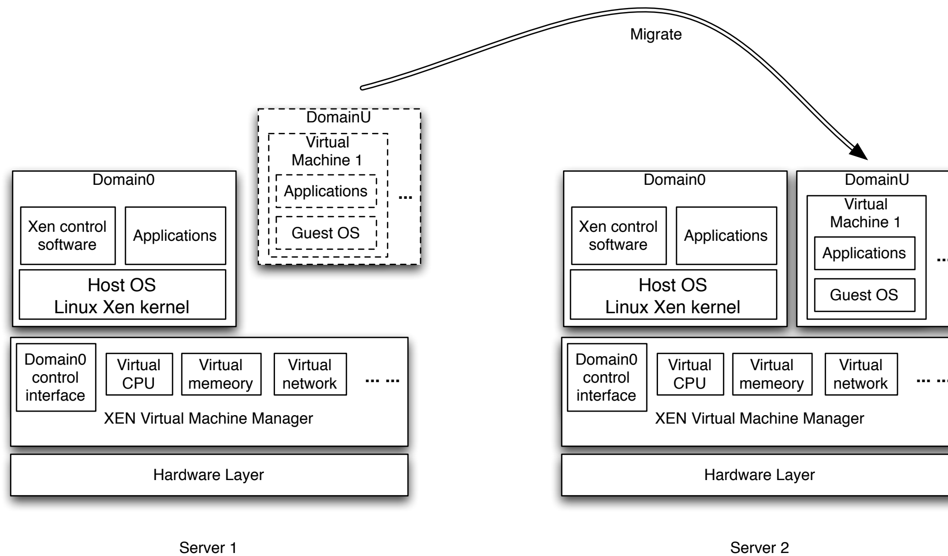


Figure 2.12: Migration in Xen

A virtual machine called virtual machine 1 is migrated from server 1 to server 2, with the entire operation system and all applications. Just like moving a physical machine to another location, live migration of virtual machines provides the minimal loss of availability.

2.6 Storage Area Network (SAN)

Shared storage is one of the primary requirements for live migration. In the virtualization field, a common approach is to set up network-based storage systems – storage area networks – for this purpose.

Basically, the SAN architecture allows operating systems to use remote storage devices as if they were directly attached. A SAN can connect different groups of storage

devices together, thus different operating systems and applications can get access and share all the disks as if they were available directly to each host. Each OS mounts or maintains its own file system on the SAN storage.

Some benefits of deploying a SAN are increased storage capacity utilization, and simplification of storage administration. It also brings more flexibility to redundancy and centralized backup methods.

2.7 Shared Storage Infrastructure

The shared storage used in virtualization is normally called SAN, however, SAN is not the only way of sharing storage for virtual machines. There are other architectures supporting shared storage as well, such as network attached storage (NAS). Figure 2.13 illustrates the different infrastructures of DAS (Direct Attached Storage), NAS (Network Attached Storage) and SAN (Storage Area Network).

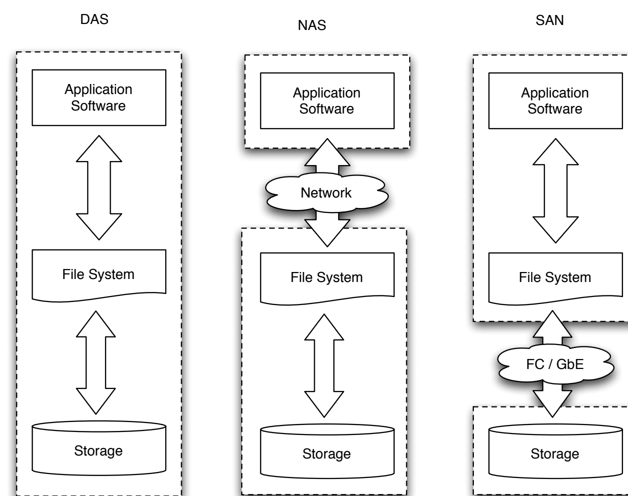


Figure 2.13: Comparison of DAS, NAS and SAN

In DAS, a storage device is directly attached in the server, with the file system mounted on top of it. Compared with the other two, NAS and SAN, there is no network needed between the storage and file system or between storage and applications.

NAS provides the file system and storage in one server, so that all the applications can share the same storage with the same file system. This means that NAS allows many computers to access the same file system over the network while synchronizing access. File-based protocols such as NFS or SMB/CIFS are widely used in this architecture. Essentially, a computer that connects to the shared storage requests a portion of an abstract file, rather than a disk block.

In contrast, SAN typically uses fiber channel links between the computer and storage to carry high-speed per-block I/O. Here, each computer connected to the SAN has its own file system. Since the typical fiber channel link operates at gigabit speeds, fiber channel-based SANs provide faster and more reliable access to the storage.

2.8 Manage Large Networks (MLN)

MLN (Manage Large Networks) is an open source management tool, used for managing large virtual networks. It is a Perl-based application, developed by Kyrre M. Begnum at the Oslo University College. It was initially released in 2004. The current version is 0.81.2.

As a management tool, MLN supports two popular open source virtual machine packages: Xen and User Mode Linux (UML). It can created a complete network of Xen or UML systems by setting up a short configuration file.

MLN builds and configures filesystem templates to manage large groups of virtual machines as logical groups. A logical group is defined as a project, which again is defined in the configuration file, together with the appropriate network setup. Since the MLN configuration language contains both variables and grouping mechanisms [17], it is possible to define several separate networks, projects, and connect them together to create larger networks. More information can be found in MLN's home page [10].

2.8.1 Defining Project in MLN

A quick start guide about downloading, installing, compiling and setting up MLN can be found in "Quick Guide for the MLN Impatient" [18], and more details, configuration and information about MLN are introduced in "The MLN Manual" [19].

As shown in the introduction of MLN webpage [20], designing a virtual network in MLN is very easy:

```
1  global {
2        project foobar
3  }
4
5  switch lan {
6  }
7
8  host one {
9      network eth0 {
10             switch lan
11             address 10.0.0.1
12              netmask 255.255.255.0
13     }
14 }
15
16 host two {
17     network eth0 {
18             switch lan
19             address 10.0.0.2
```

[10] http://mln.sourceforge.net/index.php

```
20              netmask 255.255.255.0
21      }
22  }
```

Generally, three parts are included in one project configuration. The "global" block normally defines the name of the project. Each host and network switch will have one block each. In the "switch" block, the type of the network can be specified, and each "host" block defines individual host network setup. However, not shown in the example, hosts in one project can be connected to different networks, which again leads to more flexibility to build and manage extensive virtual networks.

2.8.2 Building up Project in MLN

After defining projects, virtual machines can be setting up by running the configuration through a build phase. The necessary commands are run as follows:

1. Build a project (can be run without root privileges):

```
#The -s means to run a simulation of the project
 mln build -s -f project-file.mln
```

2. Upgrade a running project:

```
#The -S means booting the added machines
 mln upgrade -S -f new-project-file.mln
```

3. Start or stop a project (note that machines that boot first will also be shut down last):

```
mln <start | stop> -p project-name
```

4. Start or stop individual hosts:

```
mln <start | stop> -p project-name -h hostname
```

5. Issue a pause between each host boot:

```
mln <start | stop> -s seconds -p project-name -h hostname
```

6. Choose between xterm and screen:

```
mln start -s -p project-name -t screen
```

7. Check which project is running:

```
mln status
```

More information and parameters about mln can be found in the MLN manual [19].

2.9 Cfengine

Cfengine is a free software package for automating configuration and maintenance of networked computers [21]. Written by Mark Burgess, it was first published in 1993 at

Oslo University College. The current stable version is 2.2.6, which was released at the end of April, 2008. Meanwhile, the training version of cfengine 3 has been released in December, 2007, and the fully functional cfengine 3 is expected to be released at the end of 2008.

As an configuration management tool, cfengine operation is based on a policy specification. It can manage and simplify many cases of system configuration and maintenance, such as: copying files; editing files; creation, removal, and maintenance of symbolic links; file system access control lists; file and directory permissions and deletions; file system tidying; external command execution; system and user processes and so on.

Cfengine treats policies as goal states, i.e. the state that systems should be in. When it gets the information about the current system state, cfengine starts making changes to bring the system to the desired goal state. This basic mode of operation is known as convergence, see figure 2.14. The term was introduced as a concept in system administration by Mark Burgess in 1998. [22]

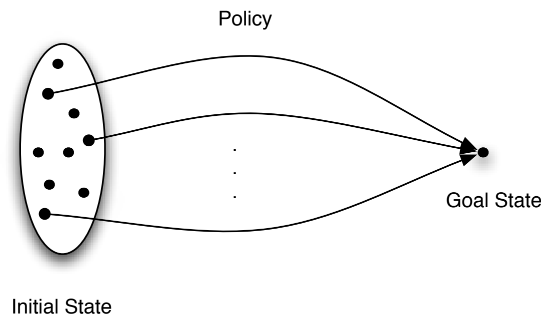


Figure 2.14: Convergent Policy

As shown in figure 2.14, if a system is convergent, no matter which initial states the system has, the redefined policies will manage to change the system to the goal state. As a result, the convergent policy is agreed as a strict policy. Since the convergent system can reach the final state by following policies only, the policies themselves are potentially difficult to be properly defined and built. The definition of convergence in computer systems is given in [21], page 4:

> *Convergence: an operation is convergent if it always brings the configuration of a host closer to its ideal, policy-conformant state and has no effect if the host is already in that state. This can be summarized in functional terms by the following meta-rules:*
>
> *cfengine(incorrect state) -> correct state*
>
> *cfengine(correct state) -> correct state*

2.9.1 Cfengine Components

There are a number of components in the cfengine software suite. Each component has its own purpose and will work together with the others to achive full operation with the expected cfengine behavior. The component list, as described in [21]:

- *cfagent:* interprets policy and implements it in a convergent manner
- *cfexecd:* executes cfagent and logs its output
- *cfservd:* server daemon for remote copy and execution, monitors the cfengine ports
- *cfrun:* contact remote hosts hosts and tells them to run cfagent
- *cfenvd:* state monitor, collects statistics about resource usage on each host for anomaly detection purposes
- *cfkey:* generates public-private key pairs (once) on a host
- *cfshow:* dumps the cfagent database contents in ASCII format
- *cfenvgraph:* dumps cfenvd's statistical database contents in a plotable form to show the observed behavior of a host in its environment

Cfagent is the main program in cfengine, and it does the main work of applying and performing operations according to a specified policy. It can be run manually, or periodically from cron, or via the cfexecd service. Generally, there are two files that describe the cfagent policy: update.conf and cfagent.conf. The former is used by clients to fetch the updated version of the configuration file, while the latter is the primate policy file to be applied in each host.

Cfexecd can be run in daemon mode or regularly via cron. It is a wrapper program to run cfagent periodically. It can also mail the cfagent output to the administrator.

Cfservd has two main functions: one is to monitor the cfengine listening ports, and the other is to provide remote copy functionality to other clients.

Cfrun is a command to start cfagent on the remote hosts. It maintains a list of hosts, and contacts them in serial.

Cfenvd mainly works as a collector to fetch statistical data from each running node. The data normally includes users, load, processes and sockets.

Cfkey must run before cfagent. However, it just needs to be run once, and it will generate a public-private key pair on each node to confirm the remote connection later.

Cfenvgraph can generate a directory of files that shows the snapshot status of the system by using the statistical database created by cfenvd.

2.9.2 Cfengine Configuration File

As shown in the "Getting started with a cfagent.conf file" tutorial [23], a cfengine configuration file is easy to set up. One example is presented here:

```
control:
  actionsequence = ( checktimezone files )
  domain         = ( example.com )
  timezone       = ( MET )

  # used by cfexecd
  smtpserver     = ( smtphost.example.org )

  # where to mail output
  sysadm         = ( me@example.com )

files:
  # Check some important files
  /etc/passwd mode=644 owner=root action=fixall
  /etc/shadow mode=600 owner=root action=fixall

  # Do a tripwire check on binaries!
  /usr                 # Scan /usr dir

  # all files must be owned by root or daemon
    owner=root,daemon
  # use md5 or sha
    checksum=md5
  # all subdirs
    recurse=inf
  # skip /usr/tmp
    ignore=tmp
    action=fixall
```

Chapter 3

Methodology

This chapter covers the basic design of the environment, including:

- Topology design and implementation
- Analysis of inter-agent behavior
- Design of the multi-agent system model
- Implementation of self-management functions in the infrastructure
- Experimental scenario testing

3.1 Objectives

The problem statements were previously discussed in section 1.3, page 5. They are reiterated here in a more formal manner, based on terms and concepts introduced in the background chapter.

1. *Design and implement self-management into virtualization administration.*
    (a) According to the problem statements, create models to analyze self-management functionality and virtual machine behavior.
    (b) Implement the models into the system configuration and network infrastructure.
    (c) Set up the experiment architecture, implement self-management functionality into the system.
2. *Test the self-managing functions through a quantitative approach. Apply statistical methods to measure and analyze observations.*
    (a) Simulate scenarios to test the network architecture and the system's self-managing functionality.

(b) Analyze result from the scenario testing, showing the virtual machine behavior in various situations and for different management purpose.

(c) Apply statistical methods to analyze measurements in order to predict virtual machine behavior and make suitable policy for the self-management functions.

(d) Discuss the decision-making while designing and implementing the experiment, the advantages and disadvantage of this system, and the usability of this approach.

This project is divided into two parts, each requiring an understanding of both virtualization and self-management. Throughout the chapter, the focus will remain on designing models, building infrastructures, developing self-management functions and measuring system behaviors.

3.2 Environment

This project is based on cooperation between three parties:

- Oslo University College, Oslo, Norway
- Gjøvik University College, Gjøvik, Norway
- Exponential Consulting, Wallingford, Connecticut, USA

A global networking infrastructure needs to be set up in order to connect all the physical nodes together. Hardware and bandwidth is provided by all three parties. Note, however, that the project is entirely managed and implemented from Oslo University College.

3.2.1 Physical Servers

As mentioned in section 2.5.2, page 22, live migration requires shared storage and same or similar CPU architectures that support the same features. Consequently, to guarantee live migration of virtual machines, all the physical servers have similar hardware, operating system and software configuration.

Table 3.1 shows the hardware information for each server. To make things simple, each server is given an alias instead of its host name. The alias will be the main naming reference throughout the paper.

As listed, four physical servers are configured for this experiment. Among them, three servers are used as virtualization servers: mln7, gjovik and gudinne – all equipped with similar CPUs. One server, huldra, is used as shared storage for all the virtual machines.

Alias	Host	Location	CPU	Memory
Master	huldra	Oslo Norway	Intel® Xeon™ CPU 3.00GHz	512MB
Oslo	mln7	Oslo Norway	AMD Athlon™ 64 X2 Dual Core Processor 6000+	8GB
Gjøvik	gjovik	Gjøvik Norway	AMD Athlon™ 64 FX-62 Dual Core Processor	2GB
Wallingford	gudinne	Wallingford USA	AMD Athlon™ 64 X2 Dual Core Processor 5000+	2GB

Table 3.1: Information of the Servers

3.3 Architecture Overview

As presented previously, a VPN is used to create a secure tunnel between the global nodes. Only nodes in possession of the shared secret will be able to connect to the VPN server. Virtual machines are set up by Xen with MLN running on top to administer large Xen deployments. Self-management functions are not provided by third parties, but are rather developed independently for the project. The automated configuration and maintenance tool, Cfengine, is used for monitoring and invoking self-managing scripts on the systems.

The architecture components form a five-layer model illustrated in figure 3.1.

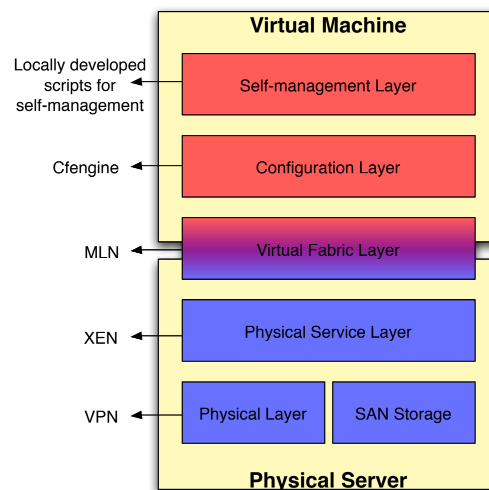


Figure 3.1: Architecture Overview

Of these five layers, the virtual fabric and self-management layers need some clarification:

In the virtual fabric layer [24], MLN provides virtual machine management. It also acts as the connection between the physical and management layers.

Arguably the most significant layer, self-management is the main functional part of this project. This layer is implemented using independently developed scripts that perform individual self-management functions. These scripts can be categorized into self-monitoring, self-analyzing, self-planning and self-executing classes. Besides the self-management scripts, other supporting scripts should be considered and developed as well, for instance, the user simulation, the web services, and so on.

3.4 Infrastructure Design Requirements

Due to the requirement of live migration and MLN management function, the first three layers require centralized network in this network architecture.

In the fourth layer, as a decentralized system, Cfengine aims to develop a flexible and stable network to avoid single point of failure. According to Cfengine strategy, each slave node needs to fetch the policy from the master node. Therefore, it will be more convenient to keep all policies in one node to ease management. If the network is set up decentralized, which means each slave node fetch the policy from different master node, the benefit of using Cfengine completely gets lost.

The main functioning part in this project is migration, which requires centralized network. Hence, if the central node goes down, even the virtual servers can still connect with each other, the migrating functions will not work anymore. Consequently, there is no meaning to keep the network decentralized. Decentralized network requires more work load, like setting up and maintaining more VPN tunnels. As a related, it causes more design and scripts in the self-management functions. As a conclusion, the network set up for this project is centralized.

3.5 Network and Storage Infrastructure

Since different technologies are involved in this experiment, their dependencies must be satisfied in the global infrastructure. This imposes limitations and also increases the difficulty of designing the network infrastructure. As discussed in section 3.4, the network has to be set up centralized.

As shown in the figure 3.2, the central server is located at the Oslo University College, named huldra, which is the central node for the VPN tunnels and also offers shared storage for the virtual machines. Three nodes are used for virtualization server, with alias name: Oslo, Gjøvik, and Wallingford.

There are several drawbacks for centralized networks. Some examples are:

1. **Single point of failure**: The entire network goes down if the central node is disconnected – data is lost if the shared storage is destroyed.

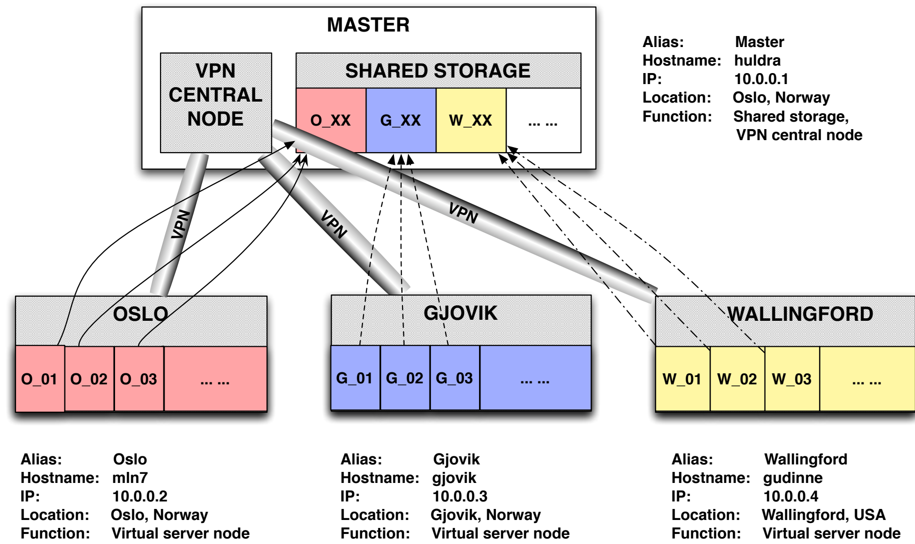


Figure 3.2: Global Server Network and Storage Infrastructure

2. **Unreliability**: A single connection for the entire network is not adaptable for the real complex Internet – no redundant path exists.

3. **Less options for routing**: A single network hub limits the possible routing paths – any given destination is reachable only through a single path.

However, the benefits of this kind of network are:

1. **Shared storage**: Migration of virtual machines depends on shared storage. Central storage is also backup-friendly and simple to maintain.

2. **Centralized management**: Easy to configure, manage and control the entire network and all virtual servers.

3. **Less configuration and maintenance for the VPN tunnels.**

3.6 Server and Software Environment

Table 3.2 lists basic network setup and task assignments for the physical servers. Table 3.3 lists software and operating system versions.

Alias	IP Address	Description
Master	10.0.0.1	Shared storage, VPN connection central node
Oslo	10.0.0.2	Virtual server node, contains virtual machines
Gjøvik	10.0.0.3	Virtual server node, contains virtual machines
Wallingford	10.0.0.4	Virtual server node, contains virtual machines

Table 3.2: Server Task Assignments

Software	Version	Function
O.S.	Linux 2.6.18-xen	Linux kernel image for xen
VTun	3.0.1-2	Virtual tunnel over TCP/IP networks
Xen	3.1.0	Virtual machine monitor
MLN	0.83.7	Virtual machine management tool
Cfengine	2.2.5-1	Tool for configuring and maintaining networked nodes

Table 3.3: Software and OS Versions

3.7 Model of Migration

3.7.1 Virtual Machines Migrate Through Tunnels

Since each virtual server connects to the central node and there are no connections between virtual servers directly, virtual machines have to move through the central node between two virtual servers and use the same shared storage in the server "Master". The migration of virtual machines and the sharing of storage are shown in figures 3.3 and 3.4.

Virtual machines have to migrate along a static path from the source server, through the central node, to the destination server. The movement is explained in figure 3.3. When one virtual machine, named "O_01" decides to migrate from "Oslo" to "Wallingford", it moves through two VPN tunnels: first from "Oslo" to "Master", and then from "Master" to "Wallingford".

After migration, the storage is distributed as illustrated in figure 3.4. The virtual server "O_01" still uses the same storage partition. This means that it will not migrate the stored data together , so that the migration time is dependent on the memory used by the virtual machine [25].

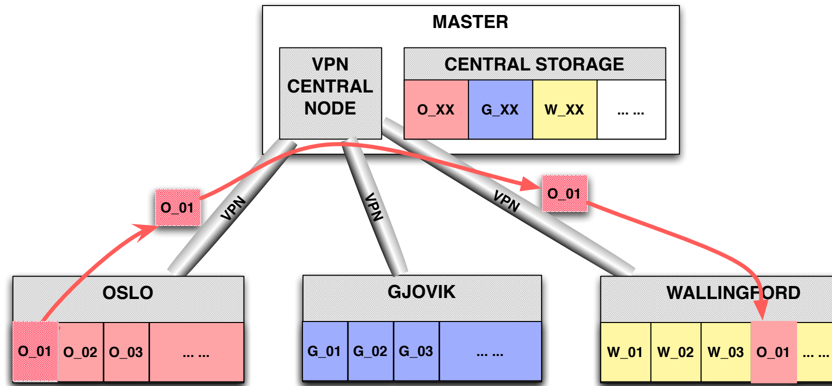


Figure 3.3: Virtual Machine Migration

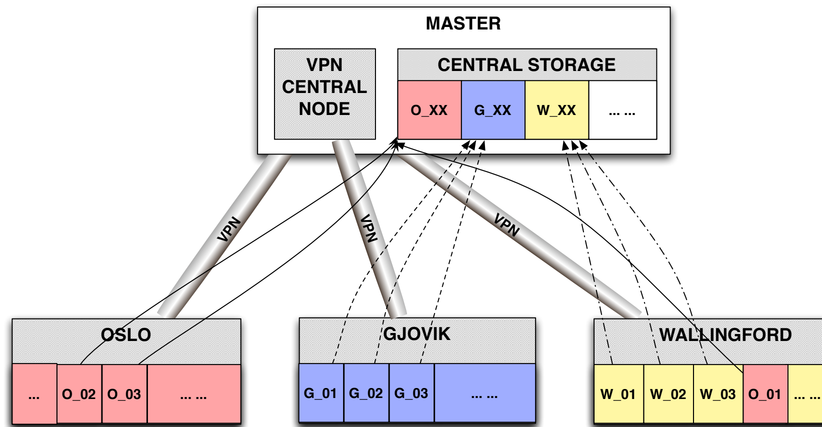


Figure 3.4: Virtual Machine Storage Sharing after Migration

3.8 Virtual Machine Self-management through Migration

According to IBM's autonomic computing model, section 2.2.1, an autonomic system relies on four basic functions: monitor, analyze, plan and execute.

In figure 3.5, the autonomic manager – virtual machine "O_01" – follows the same four steps to manage itself. All these steps are based on the knowledge inside the virtual machine itself.

As shown in figure 3.5, the virtual machine "O_01" monitors the surrounding environment, and retrieves data to populate its own knowledge. Secondly, O_01 ana-

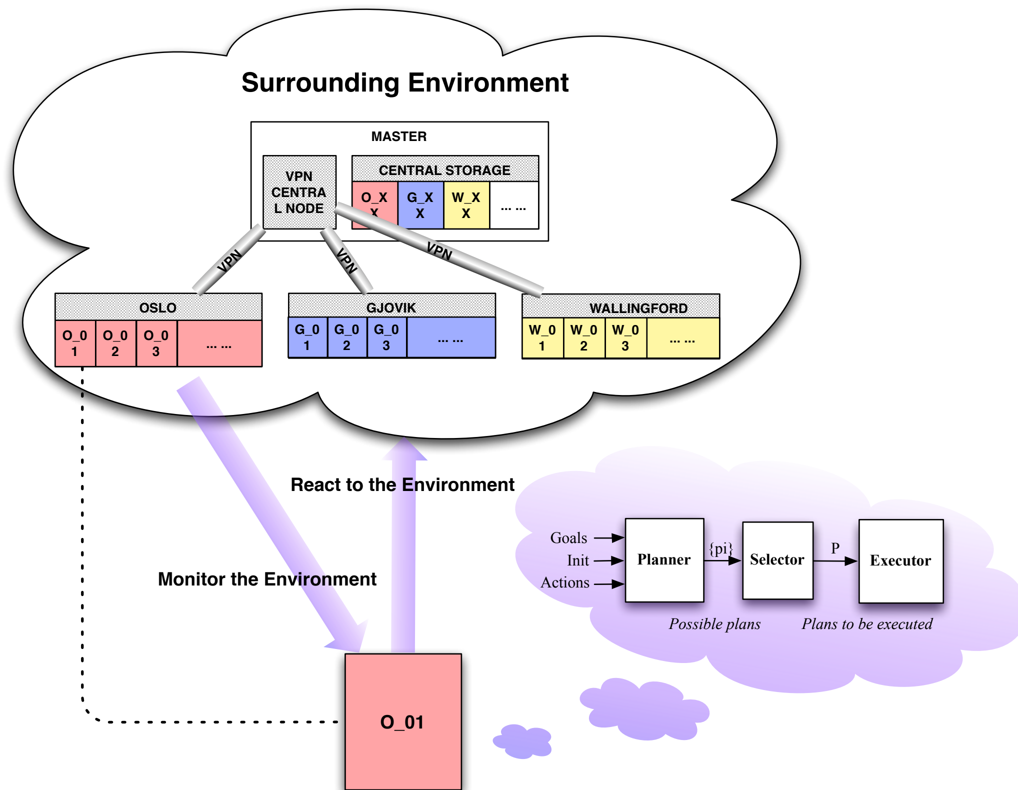


Figure 3.5: Virtual Machine Self-management through Migration

lyzes the information, and compares it with pre-defined and existing knowledge in its "brain". Based on this knowledge, it decides and executes the appropriate reaction in the environment to influence other agents to attain balance again.

In this case, the analysis and planning parts of O_01 follow a STRIPS-like operator model for multi-agent systems, introduced in section 2.3.3. O_01 gathers the initial environment status, actions causing changes, and the goals of the final states as its input information. After searching all the possible plans $\{pi\}$ in the planner, the selector of O_01 makes the decision to execute plan P. In this case, planner is response for analyzing functions and selector behaves to plan and make decision for O_01.

3.9 Scenarios

In order to test the functioning of this virtualized self-management system, two scenarios are designed to show the performance about service optimization and system integrity.

3.9.1 Scenario 1: Self-management in Service Optimization

Scenario 1 is about implementing self-management in the virtualization environment to optimize service performance. This scenario mainly focuses on reducing the service response time, which is simplified to mean packet round-trip time. To achieve this goal, scenario 1 gives one option of decreasing the distance between the service users and the servers themselves. Distance would here refer to geographical distance.

In this scenario, the virtual machines run a service and monitor its usage as part of the agents' environment. Generally, the virtual machines record the source IP addresses of connected users and their usage patterns. The servers then analyze the usage information and determine whether the largest amount of users come from a local subnet or not. If they originate from foreign nets, the virtual machines invoke the self-managing function to migrate themselves nearer to the most active source to optimize its performance.

Initial State

In figure 3.6, step 1 shows the initial state of the system for the first scenario. In the initial state, there are three virtual machines running in the server "Oslo", supporting services for the users coming from Oslo. Each virtual machine connects to the shared storage "Master", using the storage as the "virtual disk" to transfer and store data.

Monitoring State

After a period of time, the users in Gjøvik and Wallingford start to send requests for services. In this case, the virtual machine O_02 handles the requests from Wallingford, and O_03 handles Gjøvik. Since the distance between Gjøvik and Oslo, as well as the distance between Wallingford and Oslo is large compared to Oslo-to-Oslo, the service response time is affected and deemed to be non-optimal. In order to optimize the service, the virtual machines monitor the information, including the initial state. This step is showed in the picture 2 of figure 3.6.

Decision State

After monitoring the changes in the environment, the virtual machines start to analyze and plan the reactions that can be executed later. Figure 3.6 picture 3 explains the possible plan selected by the virtual machines.

Firstly, the virtual machines O_02 and O_03 monitor source IPs of requests. When they detect that the bulk of requests originate from a non-local network, O_02 plans to migrate to the Wallingford server, and O_03 plans to move to Gjøvik. This happens according to the self-management function predefined in the server applications.

Execution State

In order to adapt their own behaviors to the changed surrounding, virtual machines need to execute the decision – migrate to the desired state as already decided. The execution section is shown in figure 3.6, picture 4.

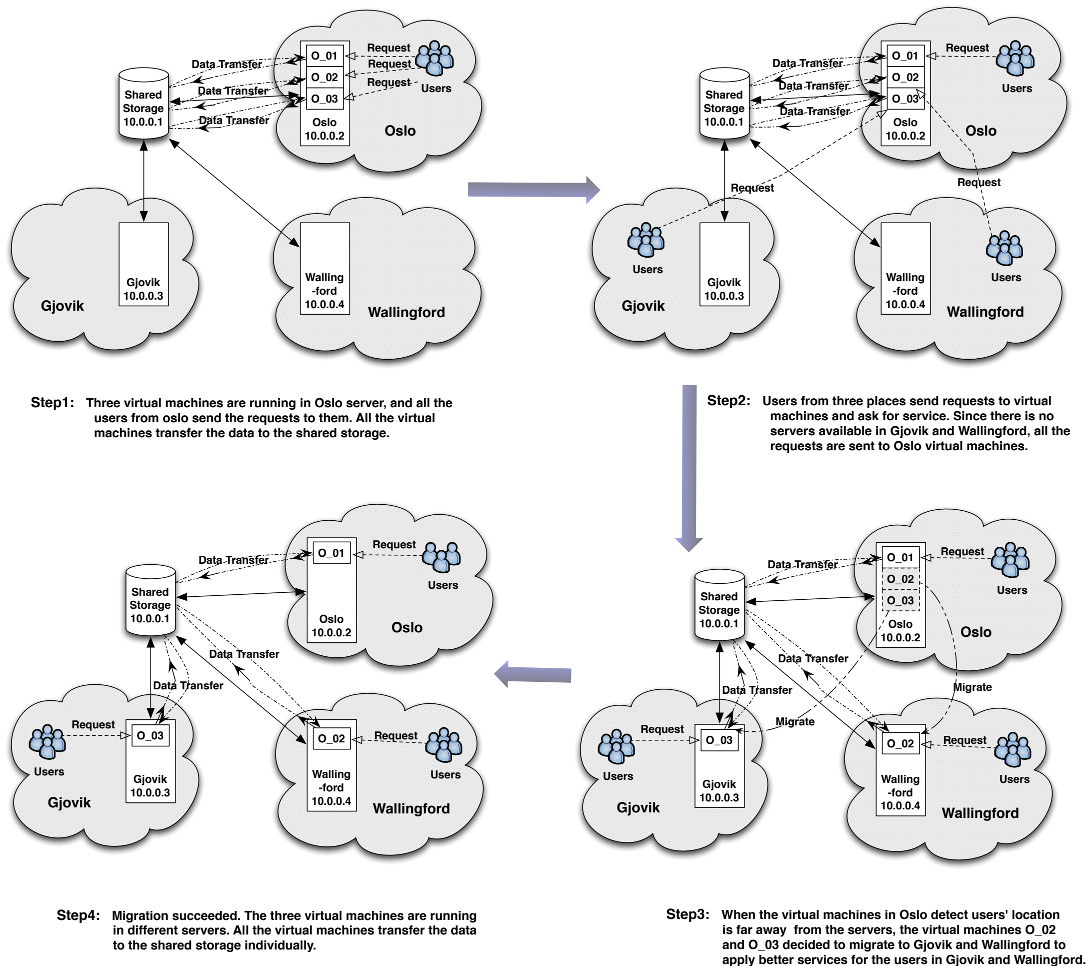


Figure 3.6: Scenario 1: Self-managemt in Service Optimization

3.9.2 Scenario 2: Self-management in System Integrity

Scenario 2 is about implementing self-management to maintain system integrity. As pointed out earlier, all the virtual machines' migration have to be based on central shared storage. To avoid the single point of failure in data storage and also to improve overall system integrity, the second scenario shows how the virtual machine manages an external backup for redundancy.

In this scenario, the virtual machines run the services and maintains all data in the shared storage. When they detect that the data has changed sufficiently – past a given threshold – since the previous backup, and they observe low network utilization, the virtual machines will make decisions about where to back up their data before continuing to run their services as before.

Initial State

In figure 3.7, step 1 shows the initial state of the system for the second scenario. In the initial state, there are three virtual machines running in the server "Oslo". All the virtual machines keep data in the shared storage.

Monitoring State

After a period of busy time, the virtual machines O_02 and O_03 detect that the data they have gathered has changed beyond an acceptable level. Also, there are quite few clients connected to them. Therefore, O_02 and O_03 start to plan to do the external redundancy backup for keeping the system integrity, shown in figure 3.7, picture 2.

Decision State

Related to the environment situation virtual machine monitored, O_02 and O_03 start to analyze and plan the behaviors that can be executed later. In figure 3.7 picture 3, O_02 decides to migrate to Gjøvik and O_03 decides to migrate to Wallingford to do the redundancy backup.

Execution State

After migration, figure 3.7 picture 4 illustrates that O_02 and O_03 transfer and store the data to the local redundancy storages, synchronizing from the shared storage.

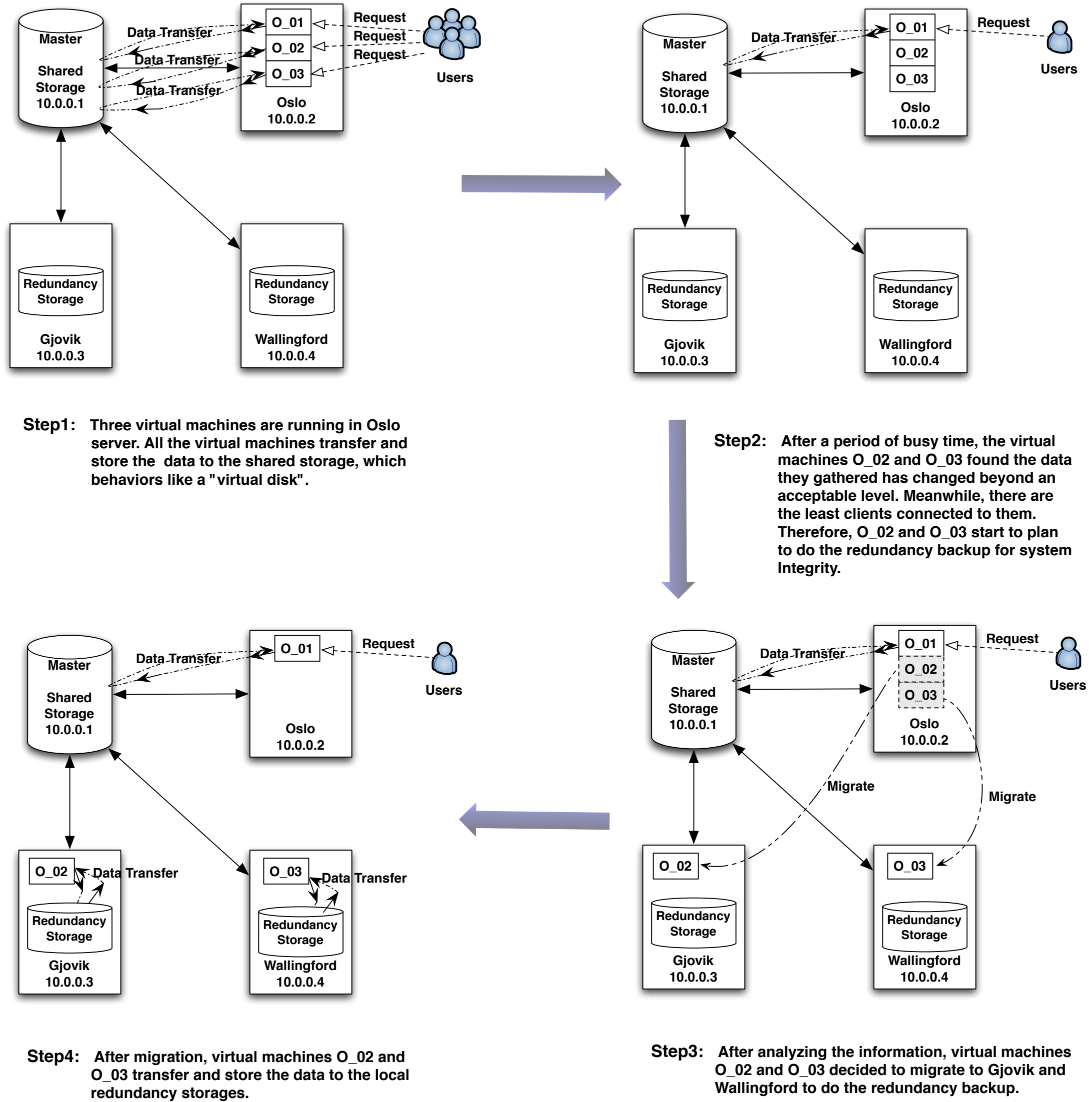


Figure 3.7: Scenario2: Self-managemt in System Integrity

Chapter 4

Result 1: System and Network Setup

In this chapter, part of the results, the system and network implementation, is presented, including creating virtual machines, setting up networks, and building the environment for the experiment.

4.1 Creating Virtual Machines

Xen and MLN are used to set up the virtualization environment and create virtual machines. The installation and configuration are introduced here.

4.1.1 Installation and Configuration of Xen

The operating system installed in all the severs is Debian 4.0. There are two ways to install Xen in Debian: Install the binary package from the Xen website, or compile Xen and kernels from the sources. The former is used, although it doesn't support quotas and iptables. These two features are not needed in this experiment.

Installing and Configuring Xen

Different versions of Xen can be found at the Xen website, http://xen.org/download/. There are several helpful tips about how to install Xen in different kernels. Detailed information can be found in [26], [27], [28]. After installation, the following commands are run to configure the Xen kernel for dom0:

```
#Disable the glibc with thread support:
mv /lib/tls /lib/tls.disabled

#Star Xen at boot time:
update-rc.d xend defaults 20 21
update-rc.d xendomains defaults 21 20
```

The device filesystem, devfs, is used by default, but causes problems when mounting the root VFS during Xen boot. To avoid this, an initrd that does not use devfs has to be created:

```
apt-get install yaird
depmod 2.6.18-xen
yaird -o /boot/initrd.img-2.6.18-xen 2.6.18-xen
```

Run the command "`update-grub`" to update grub or change the /boot/grub/menu.lst manually. After configuration, the menu.lst should contain the following as the primary boot entry:

```
title   XenLinux
kernel /xen.gz dom0_mem=512M
module /vmlinuz-xen0 root=/dev/hda1 ro console=tty0
module /initrd.img-2.6.18-xen
```

After rebooting, the server will run the Xen kernel for virtualization.

4.1.2 Using MLN to Set up Virtual Machines

After installing Xen, we can use MLN to set up the virtual machines and the network for the entire project. Since MLN can manage virtual machines from all the connected servers, creation and configuration of virtual machines become much easier. MLN is easy to set up and configure in the Xen environment. The latest MLN release is found on the MLN web page [1].

Creating Projects

To create and configure networks of virtual machines, a project needs to be defined. In this experiment, the project is called "roam". Detailed project setup is listed in table 4.1. The full configuration file can be found in appendix A.1.

Project Name	Host Name	Host Functions
roam	roamer	web service
	oslo_client	simulate clients requests in Oslo
	gjovik_client	simulate clients requests in Gjovik
	wallingford_client	simulate clients requests in Wallingford

Table 4.1: MLN Project for Creating the Network and System

MLN can also manage virtual machine migration and backup, by updating the project configuration file for the virtual machine. The update configuration files are included in table 4.2.

In the "creating system" configuration file, MLN defines the project name, the included hosts, and the configuration information for each host. For example, it specified the host "roamer" to use the Xen kernel, with 64 MiB memory. The template as well as the shared file that the virtual machine should use, and the location where the virtual

[1] http://mln.sourceforge.net

	Update File Name	Update File Functions
migration	migrate_to_oslo	migrate the virtual machine to Oslo
	migrate_to_gjovik	migrate the virtual machine to Gjovik
	migrate_to_wallingford	migrate the virtual machine to Wallingford
backup	backup_in_oslo	backup the files in Oslo
	backup_in_gjovik	backup the files in Gjovik
	backup_in_wallingford	backup the files in Wallingford

Table 4.2: MLN Project for Updating Projects

machine should be hosted are defined in this file too. "project_password" is used for certifying the actions to the project.

```
global {
        project roam
        project_password *****
}

host roamer {
        xen
        memory 64M
        term screen
        filepath /nfssan
        template roamvm.ext2
        service_host oslo
        network eth0 {
                address dhcp
    }
}
```

In the updating project configuration files, the migration update file is similar to the one for system creation. The only altered part is the "service_host". The virtual machine will migrate to different hosts depending on how it is defined in the update file. In the backup update file, the only change is to add the "xenLiveDisk" plugin function. This function aims to create a local hard drive for the virtual machine, and it also specifies the size and partition of this disk. The entire codes can be found in appendix A.3.

```
xenLiveDisk {
        /home/lux/xenodrive /dev/hdc w 1400M
}
```

Building and Managing Virtual Machines

After creating the project, virtual machines can be built and the network can be set up by using following commands:

To build roam project:

```
./mln build -r -f roam.mln
```

To start the project, run:

```
./mln start -p roam.mln
```

To update the running projects, run:

```
./mln client -h huldra.vlab.iu.hio.no -f migrate_to_oslo.mln -c upgrade
```

To run the daemon in the background console:

```
mln daemon -D /var/run/mln.pid
```

To summarize the virtual machines status:

```
mln daemon_status -s
```

4.2 Setting up Network Infrastructure

4.2.1 Configuring Shared Storage

Since there is no SAN storage available in this experiment, and also to make the infrastructure simple, the central node "Master" is used as the shared storage for virtual machine migration. NFS connects the clients and the shared storage together, so the clients can access files across a network and treat them as if they resided in a local file directory.

In the NFS server, we need to specify a directory that should be shared. In this experiment, the shared storage is mounted in /huldra/nfssan, and all directories under this will be shared as well. Since the NFS is mounted over VPN, the connections are more secure. More information about NFS can be found at its web page [29].

4.2.2 Creating Virtual Tunnels

Server Configuration File

The VTun configuration file is stored in the Master server (/etc/vtund.conf). The full configuration file is included in appendix A.4. An excerpt follows, showing how to set up one virtual tunnel between the central node "Master" and one virtual server node "Gjøvik":

```
1  # set up virtual tunnel for gjovik
2  gjovik {
3   passwd  ****;          # Password
4   type  ether;           # Ethernet tunnel
5   device tap0;           # Device tap0
6   proto udp;             # UDP protocol
7   compress  lzo:9;       # LZO compression level 1
8   stat  yes;             # Log connection statistic
9   keepalive yes;         # Keep connection alive
10 };
11 up {
12    ifconfig "%%10.0.0.3 pointopoint 10.0.0.1 mtu 1200";
13 };
```

```
14
15 down{
16     ifconfig "%%down";
17 };
```

With VTun running, IP addresses should be assigned to each client, and the network connection should be set up as well. In case the networks are not set up automatically, there are still possibility to set up the network connection manually:

```
1 #create a new interface for virtual tunnel:
2 tunctl -t tap0
3
4 #set up connection between client and server:
5 vtund  -m  gjovik huldra.vlab.iu.hio.no
6
7 #sign IP address for the client interface:
8 ifconfig tap0 10.0.0.3 netmask 255.255.255.0 mtu 1200 up
```

After the connections are created, mount the NFS through them, so that the virtual machines can get access to the shared storage through VPN tunnels.

4.3 Building Experiment Environment

4.3.1 Dynamic DNS

A web server is set up on the virtual machines to represent a service for end-users. Because the virtual machines have to migrate between different LANs, the IP address and subnet of the web server will change as well. As a result, static DNS will fail to work after each migration. In this case, dynamic DNS helps to solve the problem.

In the web server (one virtual machine), we set up a dynamic DNS update client, and register teh domain name "roamer.dynalias.org" for it. This is easily done through the dynamic DNS website (http://www.dyndns.com/).

Install an Update Client

The update client needs to monitor the current IP of the virtual machine for updates, and send it to DynDNS whenever a change occurs. This function helps to keep the IP address synchronized with DNS. The DynDNS service uses a DNS record time-to-live value of 60 seconds, which means that any intermediate DNS server between the client and server can cache the record for that amount of time. It also means that upon migration (IP address change), the web server will be unavailable for up to one minute.

The configuration file for ddclient is kept in /etc/ddclient.conf, and configured as follows:

```
1 #Configuration file for ddclient generated by debconf
2 pid=/var/run/ddclient.pid
3 protocol=dyndns2
4 use=if, if=eth0
5 server=members.dyndns.org
6 login=vmagents
```

```
password='******'
roamer.dynalias.org
```

To update the IP address for the dynamic DNS, we just need to run the command:

```
ddclient
```

Now the experiment infrastructure is completed built: virtual machines are created, network is set up, and dynamic DNS client is ready for updating the IP address to keep the web service available.

Next, we are going to implement the self-management functions and test them with scenarios.

Chapter 5

Result 2: Implementation of the Experiment

This chapter details the experiment design and implementation, including the design of the operational strategies, developing the user request generator and developing self-management functions.

5.1 Self-management Operational Strategy

Before developing the scripts, various components of the entire self-managing system have to be classified into individual functions. Figure 5.1 shows the operational strategy, how the virtual machine traverses all self-management steps.

1. All clients need a script to generate requests towards the web server on the virtual machine. The script is called generator.pl and is written in Perl. Using this script, clients can send requests mimicking human behavior.

2. Self-monitoring requires the web server to record client activity into a log file. Also in the web server, a web page is needed to support the web service. A PHP script is deployed to provide both web service and self-monitoring.

3. Based on the log file, the web server needs to analyze the data and retrieve useful parameters for decision-making later. The web server keeps the log files as its own knowledge-base for further functions, like predicting user behavior. Different self-analyzing scripts are developed for various requirements.

4. As a self-planning function, the web server plans possible actions to execute depending on the parameters it analyzed before. Cfengine can be used here to manage the self-planning (decision-making) function as it is essentially a policy decision about what to do next.

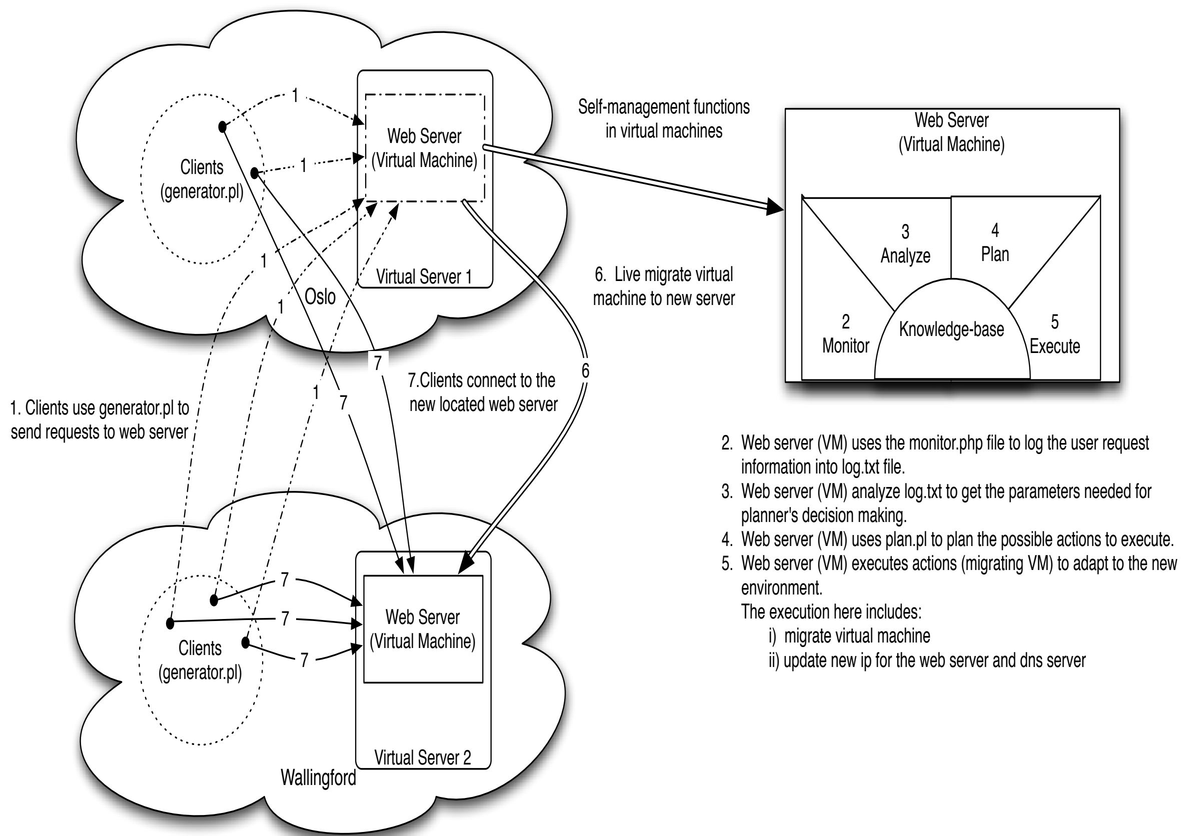


Figure 5.1: Virtual Machine Self-management Operational Strategy

5. For the final execution, the virtual machine implements actions to migrate, with the purpose being to better adapt to the environment. The executed actions include:

   - Migrating virtual machine.
   - Getting a new IP address.
   - Updating the new IP to the dynamic DNS server.
   - Copying data to an external storage (only for system integrity scenario).

6. All clients (re)connect to the newly relocated web server once the dynamic DNS is updated and the potentially cached TTL expires.

5.1.1 Developed Scripts and Their Functions

To realize the self-management functions, there are several scripts are developed. The name of the scrips and their functions are listed in the table 5.1.

Function		Script Name	Log File Name
Monitoring		roamer.php	client_track.txt
Analyzing	Scenario 1	user_percentage.pl	analyze_user_percentage.txt
		user_code.pl	analyze_user_code.txt
		user_boolean.pl	analyze_user_boolean.txt
	Scenario 2	storage_clients.pl	analyze_storage_clients.txt
		storage_logfile.pl	analyze_storage_logfile.txt
		storage_boolean.pl	analyze_storage_boolean.txt
		storage_code.pl	analyze_storage_code.txt
Decision-making	Scenario 1	cfuser	cfuser.txt
	Scenario 2	cfstorage	cfstorage.txt
Actions	Scenario 1	migration.pl	current_location.txt
	Scenario 2	backup.pl	executed_time.txt
Generating Requests		generator_{8,24}h.pl	client_log_{8,24}h.txt
RTT		rtt.pl	rtt_log.txt
Time Synchronizing		synchronizing.pl	client_track.txt

Table 5.1: Developed Scripts and Functions

5.1.2 Defining Parameters for Analysis Script

Fist of all, we need to decide which parameters we need to consider in the decision-making process.

In the first scenario, server optimization, we define three parameters: highest request percentage, user_percentage; highest requests location, user_code; and the trend of the user requests, user_boolean.

- **User_percentage**: The initial input value for decision-making. It is important because it describes the behavior of the users, and gives us an overview of virtual machine work load. Without this parameter, we can not investigate the virtual machine's states and make informed decisions.
- **User_code**: it is input as the goal, where the virtual machine should manage to arrive. It gives virtual machine the final state it should be. Without it, we can not make any policy and actions.
- **User_boolean**: this value is not as crucial as the two previous ones, as its primary use is to optimize the policy. It analyzes request rates, and tells the decision

process if user activity is increasing or not. This helps to avoid unnecessary migration of virtual machines. The point should be noticed here is that it analyzes the highest user requests trend, not the total user request trend.

Other parameters can be considered and included here like CPU or memory utilization. These parameters require more monitoring for the virtual machine, for example, by making the virtual machines talk to each other and learn the states of other neighbors. This makes the system more complicated, and time considerations keeps us from designing and realizing this. However, this can be done as a future work to improve the system and add more functions to it.

In the second scenario, system integrity, four parameters are defined: storage_clients, storage_code, storage_logfile, and storage_boolean.

- **Storage_clients**: One of the initial state values. It counts the number of requests recorded in the last 15 minutes to show how active the clients are at the moment. The virtual machine will only perform actions when it is determined that not many clients will be affected by the increased load. This parameter realizes the need for a "good" backup scheme: when few clients are connecting to the service.

- **Storage_logfile**: as one of the two important initial status values, it returns how many lines have changed in the log file since the previous backup. There are other ways of defining this measure, such as intelligent content inspection. This would inevitably require a more complex approach, though. In this scenario, the log file will not be rewritten, so we simply choose checking the size of the log file to see if it has changed since the previous check.

- **Storage_code**: like user_code, it tells the virtual machine which location it should achieve after doing the actions. It can not be removed or replaced by other parameters in this scenario.

- **Storage_boolean**: like user_boolean, this parameter helps improving the actions. In this scenario, it helps to confirm that less and less requests would be interrupt during the migration.

5.1.3 Service Optimization Process

Figure 5.2 shows the self-management processes in the service optimization scenario, described in BRIC.

There is only one process on the client side: generating user requests. At the server side, four steps are carried out: self-monitoring, self-analyzing, decision-making and executing actions.

The self-analyzing scripts will read the log file as recorded by the self-monitoring script (client_log.txt). A single invocation of the analysis process will run three functions, each returning one value: user_boolean, user_code, and user_percentage. User_code returns the location code where the highest rate of user requests comes from. User_percentage

compares the percentage of user requests from each location and returns the highest percentage. User_boolean returns zero when the highest user request rate is increasing, and one when it is decreasing.

The three values are forwarded to the self-planning process, which is primarily used for mainly for decision making. Cfengine is used to realize this function. In Cfengine, if user_percentage matches the predefined policy and user_boolean also shows that the highest request rate is still increasing, Cfengine will invoke the self-action script to migrate the virtual machine to the location corresponding to user_code.

In the action script, assuming the Cfengine policy matches, the virtual machine is first migrated to the user_code location. Next, the IP address and dynamic DNS are updated to resume normal operation of the web service.

The information and design of each function and how it works can be found in the following sections.

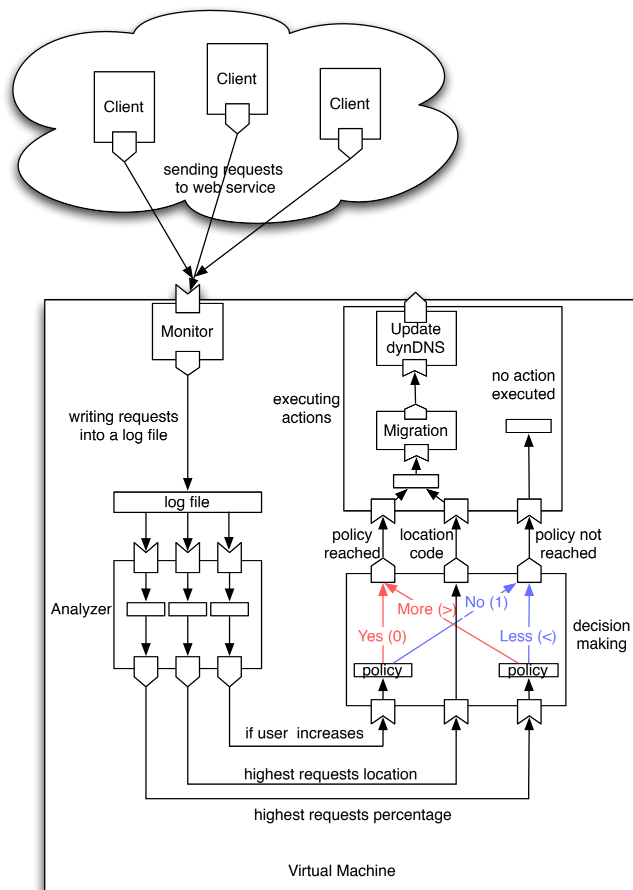


Figure 5.2: Self-management in Service Optimization Process Strategy Written in BRIC

5.1.4 System Integrity Process

Figure 5.3 shows the self-management processes in the system integrity scenario.

Both the user request generator and self-monitoring scripts are the same as the first scenario.

The self-analyzing process consists of four functions, each returning one value: storage_boolean, storage_code, storage_clients and storage_logfile. Storage_clients shows the total requests count during the last 15 minutes, a measurement of recent user activity. Storage_boolean returns zero if this number is decreasing; one if it is increasing. Storage_code returns the location code of the trusted server where the virtual machine wants to do the backup. Finally, storage_logfile returns the number of lines of the ana-

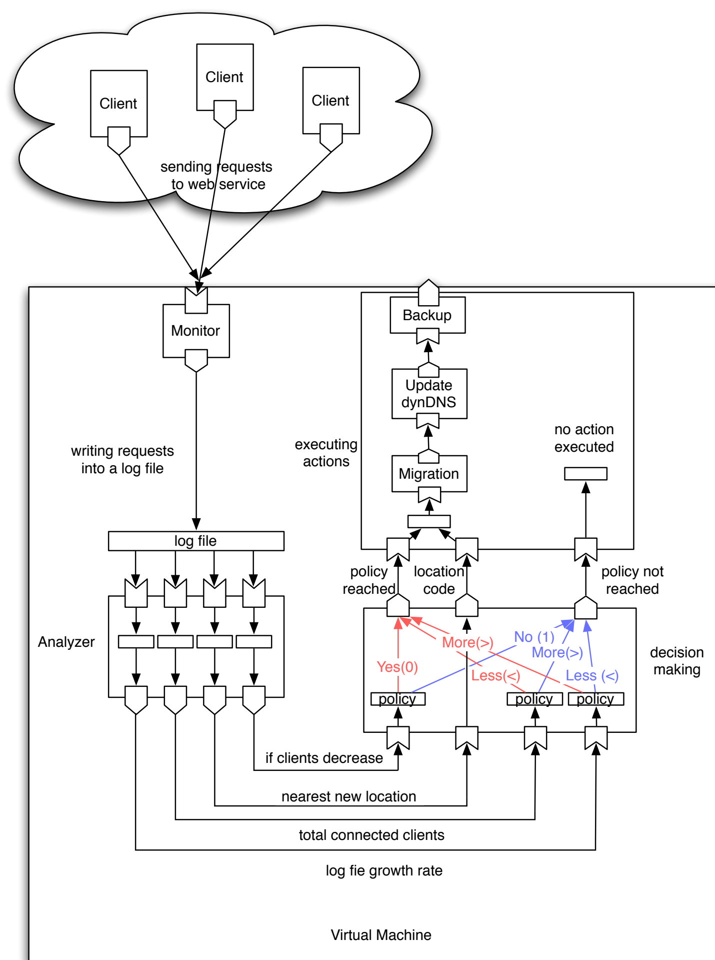


Figure 5.3: Self-management in System Integrity Process Strategy

lyzed log file that have changed since the previous backup.

When these four values are input into the self-planning script, Cfengine starts the comparison and planning functions again. In this scenario, the action will be executed only when the following three statements are true:

- Log file line count has increased above the predefined policy.
- User request rate is lower than the predefined policy.
- User request rate is decreasing.

Then Cfengine will invoke the action script to migrate the virtual machine to the storage_code location, update the IP address and dynamic DNS, and then do the local backup.

5.2 User Request Generator

Since the virtual machine self-management process depends on user activity, a request generator is written to simulate user behavior.

5.2.1 Formula for Generating Requests

We simulate the fundamental user behavior as a sine curve where one cycle corresponds to 24 hours. In its most basic form, the formula we use to generate the requests is:

$$Hits = A \cdot sin\left(\frac{2\pi x}{86400} - \frac{\pi}{2}\right) + C \quad (5.1)$$

In this formula, A controls the amplitude of the sin curve. C is a constant value that specifies the Y-axis offset. To avoid negative values, C is typically larger than A. The term $-\frac{\pi}{2}$ makes the sine curve start at the lowest value when x is zero. 86400 is the number of seconds in 24 hours, and x is the number of seconds passed since day start. This makes the formula easy to understand and calculate to match any time stamps later.

These parameters control the simulated user behavior. To yield a suitable result, the parameters must be tuned appropriately. The following sections focus on the design of this user request generator and how to define the input parameters.

5.2.2 Using Generators with Different Magnitudes

First, we must decide whether all client generators should generate the data with the same magnitudes or not. The related parameters are A and C.

In a real-world scenario, there is no chance of two clients having the same behavior all the time, so it follows that the generator should be unique for each client. However,

too much difference will cause the virtual machine self-management functions to fail, for instance the action policy is never reached. The following example shows this case.

Clients in Oslo, Gjøvik and Wallingford send user requests following different magnitudes:

$$Oslo_hits = 100 \cdot sin\left(\frac{2\pi x}{86400} - \frac{\pi}{2}\right) + 120$$
$$Gjovik_hits = 50 \cdot sin\left(\frac{2\pi x}{86400} - \frac{\pi}{2}\right) + 60$$
$$Wallingford_hits = 20 \cdot sin\left(\frac{2\pi x}{86400} - \frac{\pi}{2}\right) + 30 \qquad (5.2)$$

Accordingly, the maximum request rate will be 220, 110 and 50 for Oslo, Gjøvik and Wallingford, respectively. The resulting user request trends are shown in figure 5.4, along with the sum of all generators.

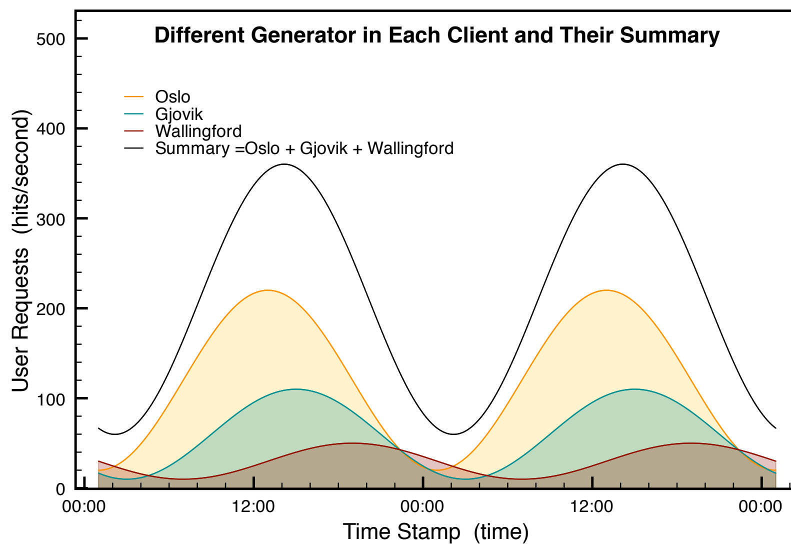


Figure 5.4: Individual Client Generators

As illustrated, every client starts its generator at 00:00 local time. With Oslo time as the reference time, Gjøvik is simulated to be two hours later. Wallingford is 6 hours behind Oslo, which corresponds with the actual time zone offset.

The user request rate in Oslo is higher than the rates in Gjøvik and Wallingford most of the time. For service optimization, the virtual machine will never get a chance to move to Gjøvik or Wallingford. Therefore, this kind of generator does not help with testing any self-management functions. To resolve this, we use similar formulas in all

locations – the slight difference will be discussed in the following section. The following is the base formula used in each generator:

$$Hits = 20 \cdot sin\left(\frac{2\pi}{86400} - \frac{\pi}{2}\right) + 30 \tag{5.3}$$

5.2.3 Adding Noise in Generator Scripts

Obviously, the user request trend will not follow the sine curve exactly. To achieve more realistic user behavior, we add some noise in the generators. This noise is gaussian, with 0 as the mean value and 0.15 as the standard deviation, i.e. the noise can be both negative and positive. Simply adding the noise to the base formula yields the following:

$$Hits = 20 \cdot sin\left(\frac{2\pi}{86400} - \frac{\pi}{2}\right) + 30 + \Delta n \tag{5.4}$$

where Δn is a random value with gaussian distribution.

The Perl module Math::Random::OO::Normal features a gaussian pseudo random generator, ideal for use in our scripts:

```
1  #generate values with 0 as mean and 0.15 as standard deviation.
2  my $grand = Math::Random::OO::Normal->new(0,0.15);
3
4  #keep seed at random value
5  $grand->seed(rand);
6
7  #formula for sine wave hits:
8  my $hits = 20*sin((2*$x*pi/86400)-pi/2)+30;
9
10 #generate noise, multiply by 5 to get useful magnitude:
11 my $noise = $grand->next()*5;
12
13 #final requests data:
14 my $data = $hits + $noise;
```

5.2.4 Adding Sleep Time in Generator Scripts

In relation to the formula, the generator will send maximum 50 requests per second. However, this will cause congestion on the server. Specially at the peak time, the web server does not have the ability to log all the requests, and most user data is lost. The output is shown in figure 5.5. Obviously, in the summary trend, most of the data is not logged around 12:00 every day.

To avoid this problem, we tune the generator to send requests once every five minutes, and sleep one second between each request. Therefore, for instance, the generator should send 35 requests during the next 5 minutes according to the formula. It will use approximate 34 $\pm$ 1 seconds to send requests, and sleep during the rest of time until five minutes have passed. The 34 $\pm$ 1 here means the total sleeping time between each two requests (34 seconds) and the time of processing the actions, which should be less than 1 second.

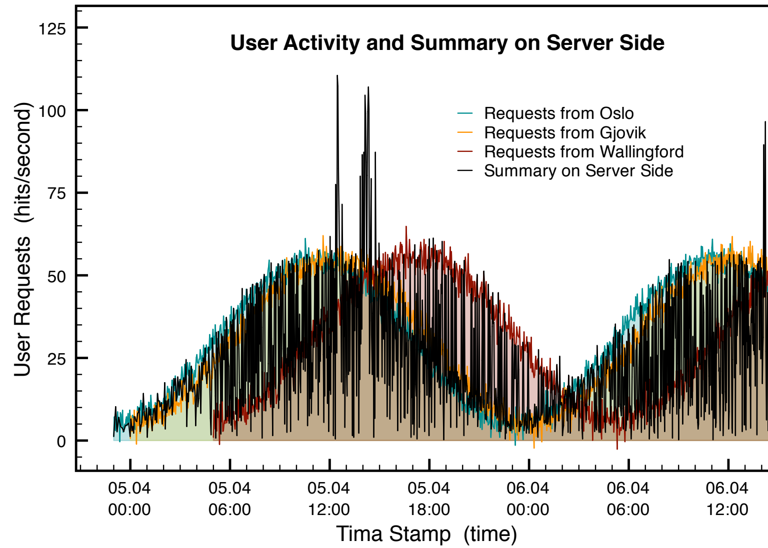


Figure 5.5: Generator without Sleeping Time

5.2.5 Using Generators with Different Sleep Time

Using different sleep time in the generator can be a way to differ the user requests from each client. Figure 5.6 shows the output of this case.

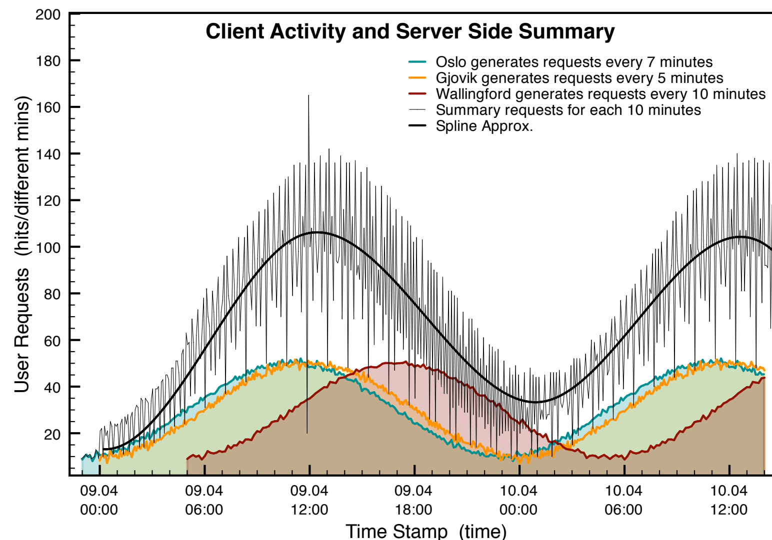


Figure 5.6: Generator with Different Sleeping Time

Oslo generates requests every 7 minutes, Gjøvik every 5 minutes and Wallingford every 10 minutes. The summary graph shows the total number of received requests per 10 minute interval. Evidently there is a lot of jumps and gaps there. The reason is an unfortunate grouping of requests because of the sleep times. Peaks mean all of three clients are active, while the gaps means that none or one client is active at that moment.

This kind of user request trend is unusable for the analysis and decision-making processes. The result would be that the virtual machine is moved continuously back and forth, without being able to properly serve requests.

To solve this, we decide to make all the generators sleep for the same time, 5 minutes, so the user activity and summary should look like figure 5.7

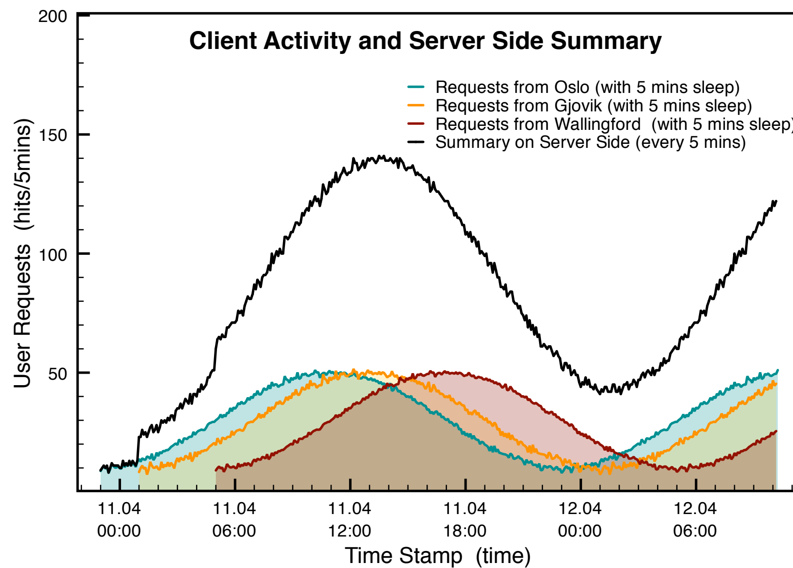


Figure 5.7: Generator with the Same Sleeping Time

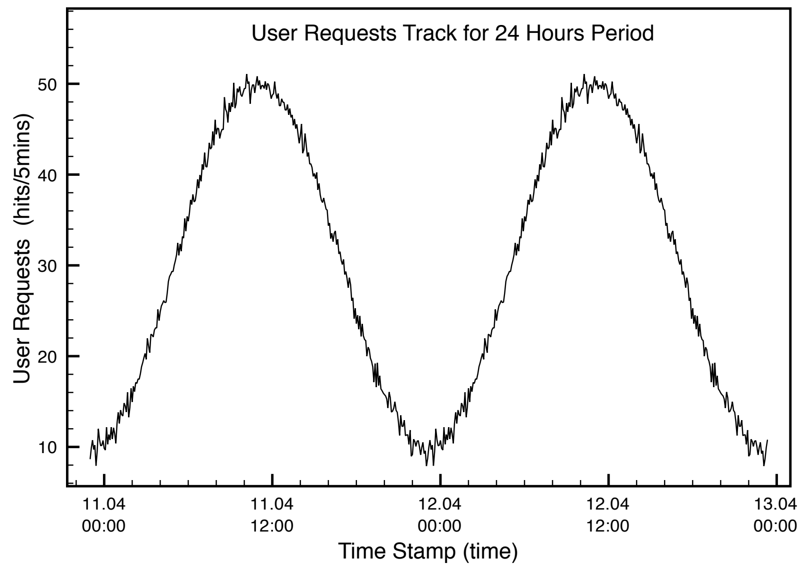
5.2.6 Using Generators with Different Periods

Two different generators are developed, so that the self-management functionality can be compared in different situations.

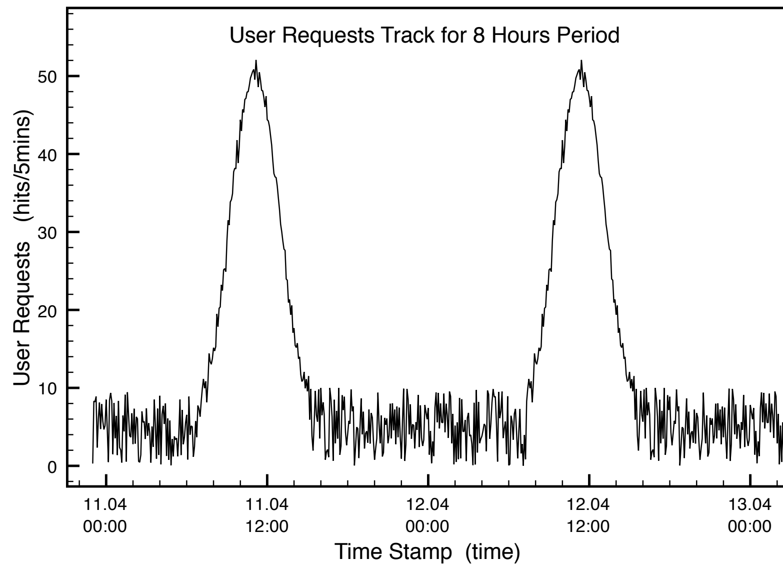
Figure 5.8a shows the user requests trend, with 24 hours as one sine curve period. The new curve stars at 00:00 am every day and the peak time is around 12:00 in the middle of the day.

Figure 5.8b is also a sine curve, but with an 8 hour period. This sine curve stars at 8:00 am, ends at 16:00 each day. The peak time is normally between 11:00 and 12:00 of the day. It simulates the requests according to the normal working hours for users. Outside of the working hours, the request rate jumps randomly between 0 and 10 hits per 5 minutes.

Both figures are the summary of user requests in 5 minute intervals.



(a) Sine Curve for 24 Hours Period



(b) Sine Curve for 8 Hours Period

Figure 5.8: User Requests Track Created by Generator

The entire scripts for two kinds of generators can be found in appendix B. Both of them will write a log file in the local user client with time stamp and total requests per 5 minutes. The log files are used for plotting client behavior.

5.3 Self-monitoring Function

In the web server, a PHP file is set up for the web service as well as monitoring the web server usage. The domain name is http://roamer.dynalias.org/roamer.php. The web service page is kept in the folder /var/www/, and the log file it created will be kept in /var/www/logfile/.

When connecting to the web page, the following information will be shown:

User's IP is: 128.39.73.85 with the time stamp: 1209426996.
The user comes from: Norway Oslo.
The location code is: 001.

Each line in the log file contains user IP, time stamp and the location code of the request. This file is very important, since all the self-management functions are based on it. The self-analyzing scripts will read and analyze this file, and return some predefined parameters to the decision-making script to trigger the virtual machine's actions. Example log file excerpt:

```
128.39.73.147    1207655281   001
128.39.73.147    1207655282   001
69.182.111.198   1207655409   003
69.182.111.198   1207655410   003
128.39.143.99    1207655414   002
69.182.111.198   1207655415   003
128.39.143.99    1207655415   002
128.39.143.99    1207655416   002
69.182.111.198   1207655416   003
128.39.143.99    1207655417   002
128.39.143.99    1207655418   002
```

The entire self-monitoring scripts can be found in appendix C.1.

5.4 Self-analysis Function

Since two scenarios are designed and tested in this experiment, the self-analysis function differs from each other.

5.4.1 Self-analysis Function in Service Optimization Scenario

This scenario is about how to implement self-management in the virtualization environment to optimize service performance. In other words, the virtual machine will monitor where most users come from, so that it can migrate to the same location. To implement this function, the virtual machine needs to analyze:

1. The user request percentage from each location relative to time.
2. Which location is creating the most user requests.
3. Whether the highest number of requests keeps increasing.

So three self-analyzing scripts are developed here: user_boolean, user_code, and user_percentage. All of them are written in Perl.

5.4.1.1 Analyzing Highest Request Rate Percentage

This script is called user_percentage, and it analyzes the log file written by the self-monitoring script. It reads the entries recorded during the last 15 minutes. If it reads more than 15 minutes, for instance one hour, the data it analyzed has become too old to show the current situation. If it reads less than 15 minutes, for instance 5 minutes, the data is too little to show the real situation. Then the noise in the second case might be too much compared with the useful data.

The user_percentage script returns the highest user request percentage, which will be compared with the policy in the decision-making script later. It also writes one log file with the time stamp, and the highest user request percentage at that time. This log file is called analyze_user_percentage.txt. Example log excerpt:

```
1207682717   0.50561797752809
1207683318   0.531835205992509
1207683918   0.553846153846154
1207684518   0.569767441860465
1207685118   0.59375
1207685718   0.612903225806452
1210443205   0.645962732919255
1210443806   0.650862068965517
1210444406   0.678733031674208
```

This log file will be used as a result to summarize the user behavior, as well as a knowledge base to decide the policy in the decision-making function later. The full script can be found in appendix C.2.1.1.

5.4.1.2 Analyze Highest Requests Location

Similarly to the previous script, this one, called user_code, reads the last 15 minutes data from the log file and returns the location where most user requests come from. This location code will be used in the action scripts later, to tell which location the virtual machine should migrate to.

The log file written by user_code is called analyze_user_code.txt. Cooperating with other log files, it also helps to show the user behavior and analyze the virtual machine's migration. The log file records the time stamp, and the location code of the highest request rate.

```
1207680917    1
1207681517    1
1207682117    1
1207682718    2
1207683318    2
1207683918    2
```

When the location code changes in this file, it does not mean the virtual machine has migrated. It only means that the highest request location has changed. However, the parameters that decide if the virtual machine should move or not are the highest percentage and the user_boolean, which shows whether the highest request rate keeps increasing or not.

The full script can be found in appendix C.2.1.2.

5.4.1.3 Analyze Highest Requests Rate Tendency

After finding where most of requests come from, we need to see the tendency of highest requests rate. This means we need to know if the highest request rate is growing or decreasing. If it is going down, there is no reason to migrate the virtual machine. Only when the highest percentage reaches the policy, and the highest requests still keeps growing, the action script will be active to migrate the virtual machine to the highest request location. The boolean value shows 0 when the highest percentage keeps increasing, otherwise, it is 1. The entire scripts can be found in appendix C.2.1.3.

The user_boolean logs the time stamp and the boolean value into its own log file: analyze_user_boolean.txt.

```
1207670715    1
1207671315    1
1207671915    1
1207672516    0
1207673116    0
```

Figure 5.9 shows the changes in the user boolean log file. When the value changes in this file, it has two meanings:

1. When 0 changes to 1, it means the highest percentage just attained its largest value, and starts to decrease. This is normally the peak time of one client's requests during the day.
2. When 1 changes to 0, it means the highest requests location changes. This is normally the intersection when the two request trends meet each other.

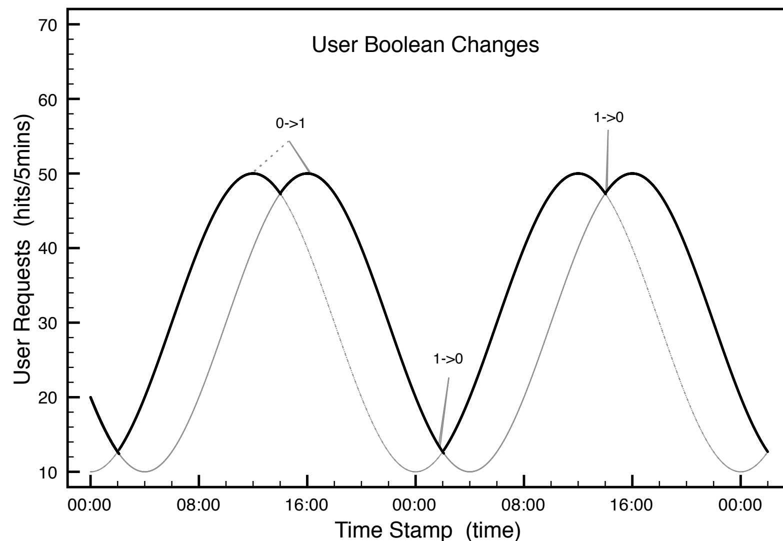


Figure 5.9: Changes in the User Boolean Log File

All the three log files created by the analysis scripts for service optimization can give us some hints about user behavior. We can also predict virtual machine behavior by reading and analyzing the data in these log files. Therefore, they are kept as the knowledge base for the further research. How to use these log files to predict virtual machine behavior is discussed in section 7.8.

5.4.2 Self-analyzing Function in System Integrity Scenario

This scenario is about how to implement self-management in the virtualization environment to keep system integrity. Therefore, in this scenario, the virtual machine migrates itself to the various locations to do the redundancy external backup. The migration only happens when the data has changed sufficiently since last backup, and only when there are few clients connected to the servers.

To make this function, the virtual machine needs to analyze:

1. The current client activity, meaning the client request rates at the moment.
2. Whether the client activity increases.
3. How many lines in the log file have changed since last backup.
4. The trusted location where the virtual machine wants to do the backup.

The self-analysis function of this scenario consists of four scripts: storage_clients, storage_boolean, storage_code and storage_logfile.

5.4.2.1 Analyze Client Activity

In this Perl script, called storage_clients, the virtual machine counts the total user requests sent to the web server during the last 15 minutes. This number shows how active the clients are at the moment. It will be compared with the client activity policy in the decision-making script later. To activate the migration action, this number needs to be lower than the policy, which means that few user requests will be interrupted. Normally, the smaller this number is, the more suitable is the time for the virtual machine to execute the migration.

The log file for this is called analyze_storage_clients.txt. It contains the time stamp and the total user requests in that time. The scripts can be found in appendix C.2.2.1.

Excerpt from log file:

```
1207733996  204
1207734596  217
1207735196  226
1207735796  236
1207736397  244
1207736997  249
```

5.4.2.2 Analyze Clients Tendency

This Perl script, called storage_boolean, aims to check if the number of user requests increases or decreases. It returns 0 if the number decreases and 1 if it increases. Ideally, the virtual machine should do the migration when the total user requests keep decreasing. More user activity means higher risk. It is considered not a good time to do the migration if an increasing number of users connect to the web service.

Similar to the user_boolean, this script will write a log file with the time stamp and the boolean value at that time. The log file is called analyze_storage_boolean.txt.

Figure 5.10 shows the changes in the storage boolean log file.

The changes normally happen when the two user request trends intersect. This is also the time when the two request numbers are nearly the same.

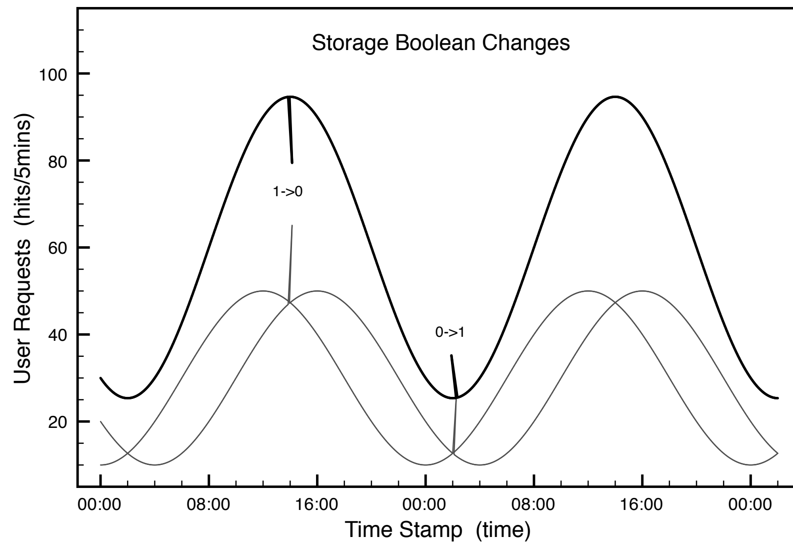


Figure 5.10: Changes in the Storage Boolean Log File

1. When 0 changes to 1, it indicates when the user has lowest activity. The migration normally happens before this time. After this, the user requests start to increase again.
2. When 1 changes to 0, the number of user requests are also equal to each other, but that is the highest total user activity. No migration action will be executed around that time.

The full scripts can be found in appendix C.2.2.2.

5.4.2.3 Analyze Clients Location

This perl script aims to find the current virtual machine's location and also the next location where it should go. Depending on the least interruption policy, this script will return the nearest location code as the migration destination and do the backup. Table 5.2 shows the policy defining how to find this location. The distances shown here is the direct distance between each location, referring to the information from Google Earth [30].

The log file kept by this script is similar to the user_code log file, with the time stamp and new location code. The entire scripts can be found in appendix C.2.2.3.

Current Location	New Location	Distance
Oslo	Gjovik	95km
Gjovik	Oslo	95km
Wallingford	Oslo	5,783km

Table 5.2: New Location for Storage Backup Policy

5.4.2.4 Analyze Log File's Growth Rate

To find the the log file's growth rate since last backup, we need another analysis script, storage_logfile. This script will first determine when the last action was executed and then count the total growth of the log file since last action execution. It return one value showing how many new lines there are in the log file. The entire scripts can be found in appendix C.2.2.4. Excerpt from log file:

```
1207735796   559
1207736397   623
1207736997   789
1207737597   960
1207738197   1139
```

5.5 Self-planning Function (Using Cfengine for Decision-making)

The self-planning function is implemented in Cfengine. The main purpose of this script is to make the predefined policy and invoke the other self-managing scripts according to these policies. The reason to use Cfengine for decision-making can be found in discussion, section 7.6. The full configuration file can be found in appendix C.3.

Cfengine works through these steps to implement the decision-making function:

1. In the configuration file, Cfengine invokes the self-analysis scripts to gather useful information and parameters.

2. It compares these parameters with the policies predefined by system administrators.

3. If the parameters match the policies, Cfengine invokes the action scripts to execute migration. Otherwise, it keeps analyzing and gathering information, until the requests are satisfied.

5.5.1 Decision-making in Service Optimization Scenario

This Cfengine configuration file is made for service optimization. It invokes the three analysis scripts made for the first scenario, and executes the migration action script when the policy is matched.

1. According to Kepharts' goal, system administrators only supply higher level goals [9]. In the control block, we need to gather the information and define the policy. Since the policy controls the start point of actions, it is one of the factors that influences the final results. In this experiment example, we define 60% as the policy for the highest user percentage.

```
1 #!/usr/sbin/cfagent -K -f
2 # this is where we gather information and make policy
3 control:
4  actionsequence = (  editfiles shellcommands )
5
6  userpercentage = ( ExecShellResult("/var/www/analyze/user/user_percentage.pl") )
7  usercode = ( ExecShellResult("/var/www/analyze/user/user_code.pl") )
8
9  migrationthreshold = ( 0.6 )
```

2. In the classes block, Cfengine compares the highest user percentage returned by user_percentage with the predefined policy. If the user percentage value is greater than the policy, the first request of migration is reached. The second request of migration is defined by the user_boolean. When it returns 0, meaning the highest user percentage is still increasing, the second request is satisfied as well.

```
1 # this is where we make decisions
2 classes:
3  any::
4    migrationThresholdReached =
5         ( IsGreaterThan(${userpercentage},${migrationthreshold}) )
6    userboolean =
7         ( ReturnsZero("/var/www/analyze/user/user_boolean.pl ${usercode}") )
```

3. When both requests are satisfied, Cfengine invokes the migration action. It migrates the virtual machine to the location determined by the user_code script before, and updates the IP address and dynamic DNS to make the web service available again.

```
1 # this is where we do the migration actions
2 shellcommands:
3  migrationThresholdReached.userboolean::
4         # execute actionscript for migration
5         "/var/www/action/migration.pl ${usercode} "
6         owner=root group=root
```

4. While doing the self-management functions, Cfengine helps to log the information that states where virtual machine is. Excerpt from log file:

cfengine:roamer: User percentage is 0.639325842696629.
cfengine:roamer: Usercode is 1.
cfengine:roamer: Userboolean is defined as 0.
cfengine:roamer: Migrationthreshold 0.6 is reached, migrating to 1.
cfengine:roamer: migration.pl: The virtual machine is located where it should be.

5.5.2 Decision-making in System Integrity Scenario

This Cfengine configuration file is made for system integrity. It invokes the four analysis scripts, and executes the migration action script as well as the backup action script when the policy is matched.

1. Two policies are define here: one for the maximum client activity, and the other for least log file growth.

```
#!/usr/sbin/cfagent -K -f
# this is where we gather information and make policies
control:
 actionsequence = ( editfiles shellcommands )

 storageclients  =
                ( ExecShellResult("/var/www/analyze/storage/storage_clients.pl") )
 storagecode = ( ExecShellResult("/var/www/analyze/storage/storage_code.pl") )
 storagefile = ( ExecShellResult("/var/www/analyze/storage/storage_logfile.pl") )

 migrationthreshold = ( 180 )
 logfileincrease = ( 500 )
```

2. Cfengine compares two groups of values here. If the total user requests is less than the defined maximum clients activity policy, and the log file has grown more than the defined least log file growth policy, Cfengine marks the first two requests reached. The third request for migration is decided by the storage_boolean. When it returns 0, meaning the client activity is decreasing, the third request is satisfied.

```
# this is where we make decisions
classes:
 any::
        migrationThresholdReached =
                ( IsGreaterThan(${migrationthreshold},${storageclients}) )
        logfileincreaseReached =
                ( IsGreaterThan(${storagefile},${logfileincrease}) )
        storageboolean =
                ( ReturnsZero("/var/www/analyze/storage/storage_boolean.pl") )
```

3. When all the requests are satisfied, Cfengine first invokes the migration action, and then invokes the backup action.

```
# this is where we do actions
shellcommands:
 # execute actionscript for migration
 migrationThresholdReached.logfileincreasitivityReached.storageboolean::
```

```
5        "/var/www/action/useraction.pl ${usercode} "
6        owner=root group=root
7        "/var/www/action/storageaction.pl"
8        owner=root group=root
```

4. Cfengine still helps to log the analyzed information, as well as the executed actions. Excerpt:

cfengine:roamer: Storagelogile has increased 4264.
cfengine:roamer: Storageclients is 93.
cfengine:roamer: Storageboolean is true.
cfengine:roamer: Migrationthreshold 180 is not reached.
cfengine:roamer: Log file 4264 is longer than the policy 500.
cfengine:roamer: Everything is set to execute the storage action, migrating to 2.

5.6 Action Scripts

Two actions are executed in this experiment: migration and backup. For the first scenario, service optimization, only migration is executed. While in the second scenario, system integrity, the virtual machine is migrated first, and then runs the backup action. The entire scripts can be found in appendix C.4.

5.6.1 Migrating Virtual Machine

The migration function is administrated by MLN. The virtual machines can migrate to another location by updating a new project file. The new location is analyzed and decided by the self-analyzing scripts. So in this migration action script, these functions are executed:

1. Check the virtual machine's current location, compared with the new location code. If they are the same, it returns the message "Virtual machine is already where it should be." Otherwise, migrate to the new location.
2. According to the new location code returned by the self-analyzing scripts, we need to choose the corresponding project files to do the migration.
3. After the migration, update the IP address for the virtual machine. Run this command as a loop, until the IP is determined.
4. Update the new IP address to the dynamic DNS server to make the service available again.

Whenever the migration script is executed, it writes the time stamp and the virtual machine current IP address into a log file, to see the virtual machine's migration trends. Excerpt:

1210455208	inet addr:128.39.140.202 Bcast:128.39.143.255 Mask:255.255.252.0
1207698991	inet addr:128.39.73.238 Bcast:128.39.73.255 Mask:255.255.255.128
1210456473	inet addr:128.39.140.202 Bcast:128.39.143.255 Mask:255.255.252.0
1207714654	inet addr:128.39.73.238 Bcast:128.39.73.255 Mask:255.255.255.128

5.6.2 Back up Files to Local Redundancy Storage

MLN also has the function to manage the virtual machine to do the local backup. This action is implemented by updating a new configuration file for this project. MLN creates a hard drive first, then formats and mounts it. After attaching to the hard drive, the virtual machine can do the backup to this local redundancy storage, by copying files from the central shared storage. The other functions are developed in the backup action script. It follows these steps to do the backup:

1. Check the virtual machine current location, and choose the corresponding update file to do the backup action.
2. After a hard disk is created, formated and mounted, copy the files from the central shared storage to the local redundancy disk.
3. Unmount the disk for the virtual machine
4. Run the original project file to detach the local disk from the virtual machine.

The files backed up in the redundancy disk includes all the configuration files for the web service, self-management function scripts, log files from each function, etc. When the backup action is executed, it writes the execution time into a log file. This time stamp is used for checking the growth of the user data. So the storage_logfile will count the data increases from this time stamp until its current time.

1208037751
1210811435
1208058232
1210819914
1208066718
1210840408
1208087217

Note, in some log files shown above, there are errors about the time stamp.

This is one problem in this experiment: When the virtual machine migrates, the time does not always synchronize with the dom0 virtualization server, instead introducing a severe time shift. It is a bug in Xen, and has not been fixed when this thesis is written. This problem is clearly discussed in the section 7.5.

Chapter 6

Result 3: Measurement and Analysis

This chapter covers the output from the experiment and the final results. The following information is presented: user requests analysis, self-management log file analysis, and virtual machine behavior.

6.1 User Requests Analysis

The web server and each client record user requests into log files. In the web service log file, the information includes: user IP address, time stamp, and user location code. Each client log records the time stamp and the total requests in the last 5 minutes. All the log files keep growing as time passes by. Two scripts are developed to filter the useful information from the log files to plot graphs and analyze behavior later. The entire scripts can be found in the appendix D.1.

Both scripts filter the log file to get the data within the nearest one or two days. The output of the script is the time stamp and total requests per 5 minute intervals.

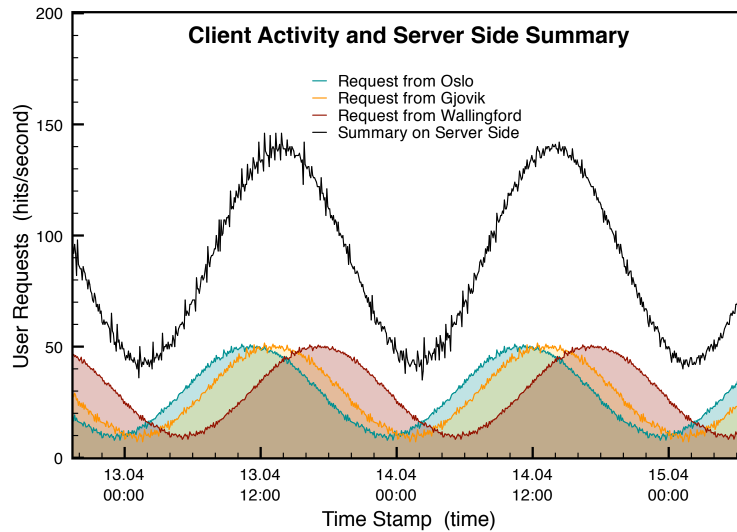
6.1.1 User Request Trends and Their Summary

A virtual machine is given the role as user client at each location. All of them run the same generator script to send the user requests to the web service. Each client starts the script at 00:00 local time. Since they are in different time zones, the start times will not be the same. Figure 6.1 shows the user activity in each location, as well as the requests summary observed on the web server.

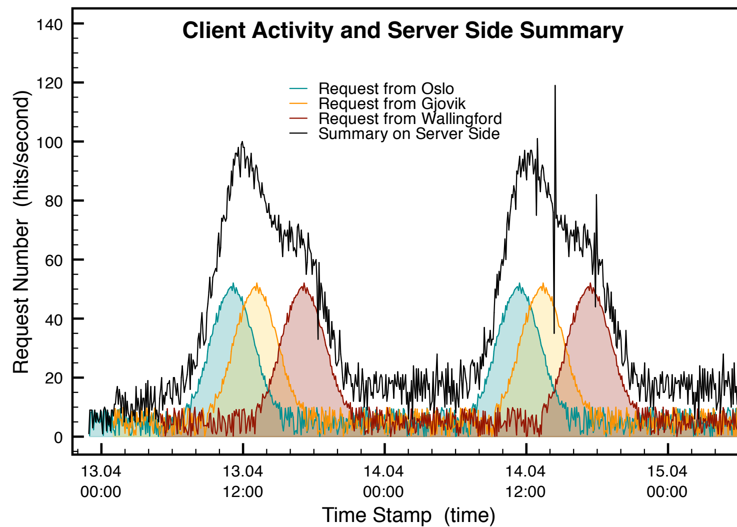
Also mentioned previously, the time difference between Oslo and Wallingford is 6 hours, which corresponds to the actual geographic time zone offset. The difference between Oslo and Gjøvik is simulated to 2 hours.

Figure 6.1a shows the user activity and server-side summary with 24 hours corresponding to one sine cycle. The log file in the server side is important, not only for later analysis, but also for keeping as knowledge base. It clearly shows that the summary still closely resembles the sine curve.

Similar to figure 6.1a, figure 6.1b shows the user activity and server side summary with 8 hours corresponding to one sine cycle. The main difference is the server-side summary, which deviates from the base sine curve. This makes it an interesting comparison to the 24-hour graph, as it will impose varied behavior and actions on the virtual machine.



(a) User Request Summary for 24 Hours Period



(b) User Request Summary 8 Hours Period

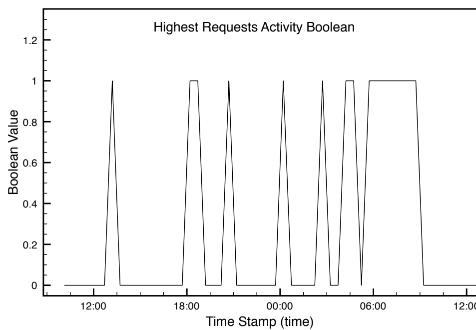
Figure 6.1: User Requests Trends and Server Side Summary

6.2 Self-management Log File Analysis

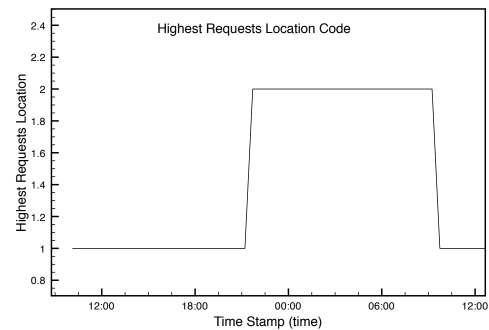
6.2.1 Log File Output

Figure 6.2 and 6.3 show the output of the log files that are written by the self-analyzing scripts in the two scenarios: service optimization and system integrity.

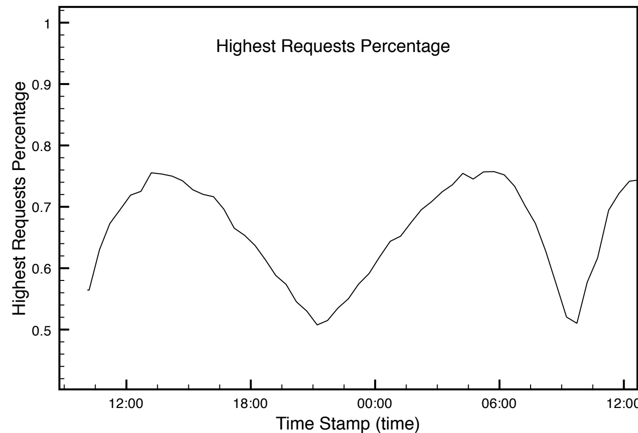
For the service optimization, figure 6.2a is the output of user_boolean script. It has only two values: 0 and 1. When the value is 0, the highest requests rate is still increasing, and the related policy in decision making is reached. However as shown in the figure 6.2a, there are more changes than we expected. That is caused by the noise. Sometimes, compared with the sine curve data, the noise is too much to change the trend of it. Therefore, our system is sensitive to noises. Figure 6.2b shows the highest requests location, and figure 6.2c is the highest requests percentage trend.



(a) Output of Highest Requests Trend (Boolean)



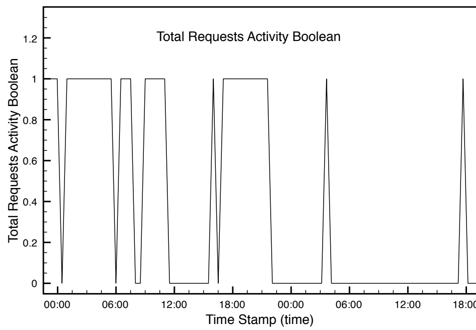
(b) Output of Highest Location (Code)



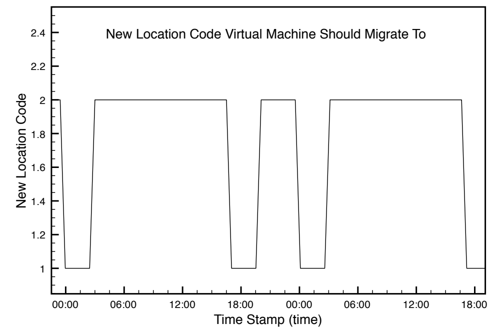
(c) Output of Highest Requests Percentage

Figure 6.2: Log File Output for the Service Optimization Scenario

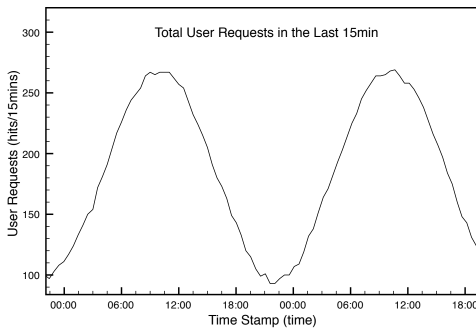
For the system integrity scenario, four analysis log files are kept. Figure 6.3a is the boolean value for the total user requests activity. A value of 0 means that the total requests rate is decreasing, which matches the decision-making policy. A new location code where the virtual machine should migrate to is shown in figure 6.3b. Figure 6.3c shows the total requests during the last 15 minutes, and figure 6.3d is the number of new lines in the log file (logged by monitor scripts) since the previous backup. Both of them need to match the policies in Cfengine to execute the actions for virtual machine.



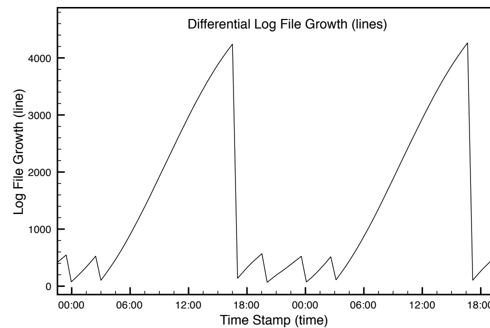
(a) Output of Total Requests Trend (Boolean)



(b) Output of New Location Where Virtual Machine Should Migrate To (Code)



(c) Output of the Total Requests in Last 15mins



(d) Output of the Growth Lines in the Log File since Last Backup

Figure 6.3: Log File Output for the System Integrity Scenario

6.2.2 Highest User Requests Percentage Trend

From the log files processed by the analysis, we can analyze them and find more detailed information about these data. Here, we mainly focus on the highest requests percentage log file, since other files are either simple to directly get information or complicated to analyze.

To simplify the analysis, we take only two locations into consideration, Oslo and Gjøvik. The time difference between them is changed so that the local time in Gjøvik is 6 hours behind Oslo.

Figure 6.4 represents the highest request percentage to show how the virtual machine analyzes user behavior. The bold black line is the highest requests trend. In this example, it is always higher than 50%.

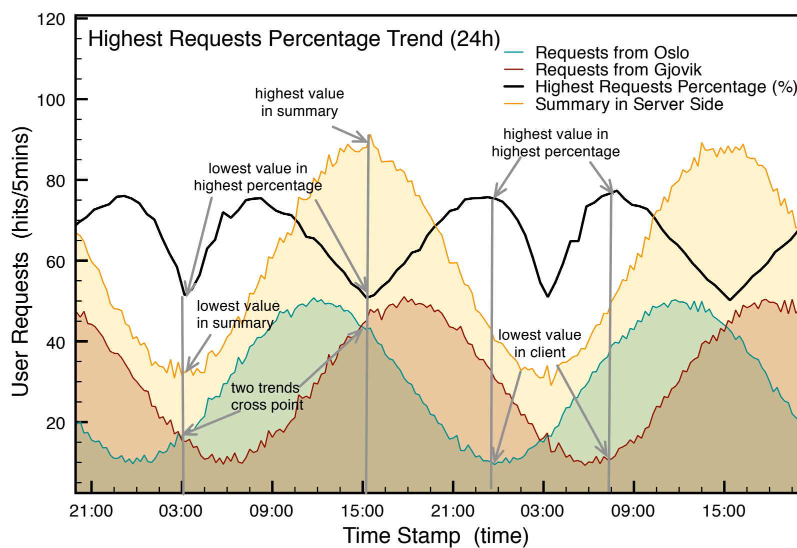


Figure 6.4: Highest Request Percentage Trend

Shown in the figure, the lowest value in highest request percentage trend always happens when the two user request trends cross each other. That is also the time when the total request reaches the highest value or the lowest value. The reason is that when the two trends meet, the request rates are equal for both clients. Consequently, the percentage is around 50% in each location, which is the lowest value the highest request percentage can get. While the highest value in the highest request percentage trend happens near the lowest value any one client can attain.

6.2.3 Highest User Requests Percentage and the Corresponding Policy

The migration policy defined in this case is 60%, shown as the red line in figure 6.5. Whenever the highest user request percentage crosses over this line, the environment satisfies the first of two requirements for virtual machine migration.

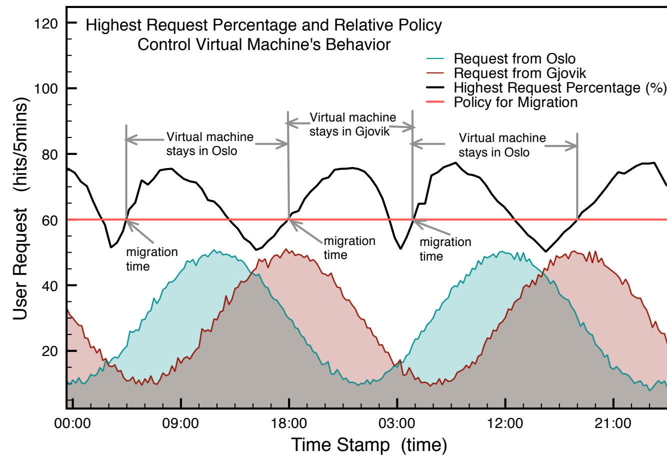


Figure 6.5: Highest Request Percentage and Relative Policy Control Migration

Next, we want to find out how policy influences virtual machine behavior and how to make a suitable policy. In figure 6.6, we define two policies, 70% and 55%. Both of them can be reached in the highest request percentage log file.

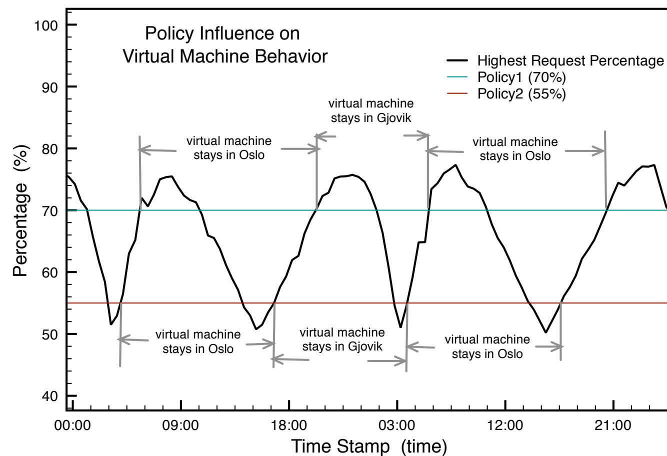


Figure 6.6: Policy Influence Virtual Machine Behavior

It shows that higher policy values means shorter time for the virtual machine to migrate to and stay in Gjøvik. Other than this, there is not much difference between the two policies. Therefore, as long as the policy value is included in the highest percentage log file, the virtual machine is able to do the migration. However, if the policy is too high or too low, for example using the minimum or maximum value as the policy, the request might only be reached once.

6.3 Virtual Machine Behavior Analysis

6.3.1 Measuring the Round Trip Time

In the problem statement chapter 1.3, round trip time is represented as a way to measure the service optimization: shorter round trip time means better service. In addition, round trip time trends also describe the virtual machine's migration behavior. For example, taking only two locations in consideration, Oslo and Gjøvik, the round trip time from client to web server varies when the virtual machine changes location. Shorter round trip time means the virtual machine is at the same location as the client, while longer round trip time means the locations are different.

A measurement script is developed to log the round trip time between user client and web server. The entire script can be found in appendix D.2.

This round trip time measurement script sends 3 ICMP ping requests every time to the virtual machine, and logs the time stamp, average round trip time and deviation into a log file. This is how the log file looks like, an excerpt from the Oslo client:

```
1210513709  0.053  0.029
1210513719  0.074  0.025
1210513729  0.053  0.031
1210513739  0      0
1210513749  0      0
1210513759  0      0
1210513769  0      0
1210513779  0      0
1210513789  0      0
1210513799  0      0
1210513809  0      0
1210513819  2.207  0.028
1210513829  2.258  0.036
1210513839  2.281  0.041
```

Through the log file, we can clearly see the difference when the virtual machine stays in each location. When the round trip time is below 0.1ms, the virtual machine is located in Oslo. When the round trip time is more than 2ms, the virtual machine stays

in Gjøvik. This log file also shows how long the migration takes around 70 seconds, the period where the measurements are zero.

6.3.2 Using Round Trip Time to Measure Virtual Machine's Migration

Continuing with Oslo and Gjøvik, figure 6.7 shows the round trip time from both sides. The round trip time trends are square waves. When the round trip time is on the lower bound, the virtual machine is located in the same place as the ping originates from. Migration happens in the steep change from low to high, or vice versa.

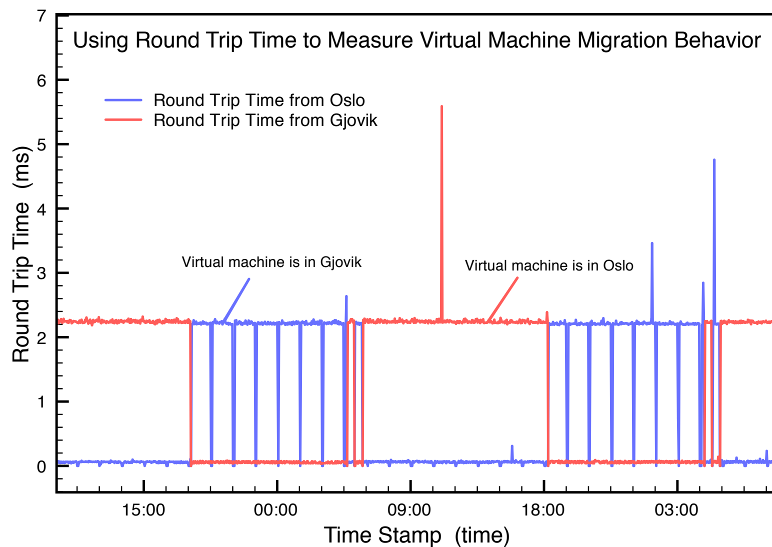


Figure 6.7: Using Round Trip Time to Measure Virtual Machine's Migration

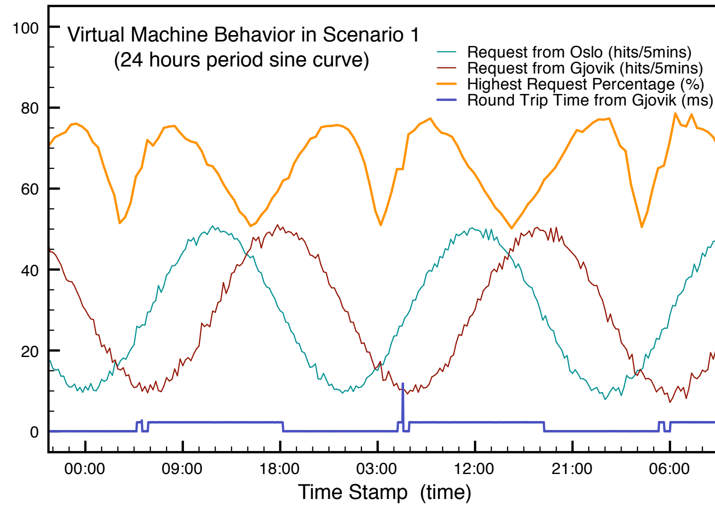
There are some interesting information in the figure, such as the Oslo round trip time that drops to zero every 30 minutes. The causes for this could be various factors. Perhaps it is caused by the special network setup in the Oslo server, or because the server has filter tools or intrusion detection installed that mistakenly drop valid traffic. However, it is unlikely that the dips are caused by script errors, since Gjøvik does not have the same problem, although it runs the exact same scripts. Because of time constraints, the cause has not been investigated further. However, this issue does not put a stop to the experiment.

6.3.3 Virtual Machine Behavior in Scenarios

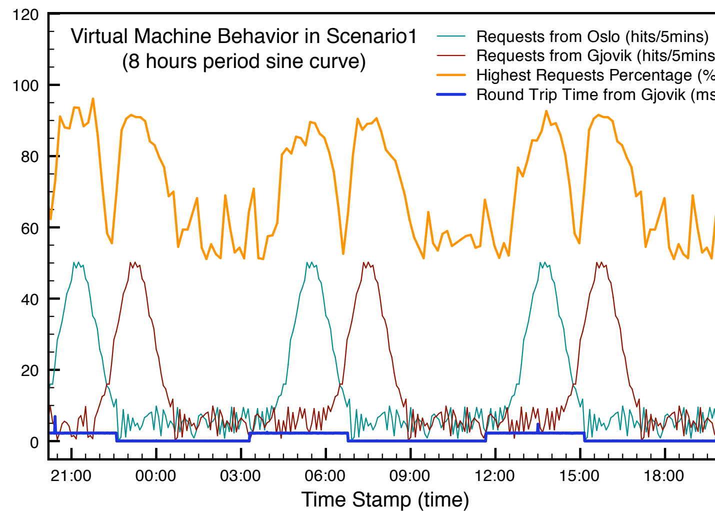
Because the network between Wallingford and Oslo or Gjøvik is unstable, the scenario testing is mainly done between Oslo and Gjøvik. To facilitate self-management, the time in Gjøvik is again simulated to be 6 hours behind Oslo.

6.3.3.1 Virtual Machine Migration in Service Optimization

Figure 6.8 shows the output of the self-management function for the purpose of service optimization.



(a) 24 Hours Period of Sine Curve



(b) 8 Hours Period of Sine Curve

Figure 6.8: Virtual Machine Migration in Service Optimization

The green and red lines show the request rate trend from Oslo and Gjøvik, and the yellow curve is the highest request percentage logged by the analyzer. The blue square wave is the round trip time taken from Gjøvik. So the lower bound means the virtual machine is located in Gjøvik and the higher bound means it stays in Oslo.

As shown, this self-management system can handle different kinds of user requests. The virtual machine behavior varies with different input parameters, mainly the user request rate trends.

6.3.3.2 Virtual Machine Migration in System Integrity

Figure 6.9 shows the output of virtual machine behavior when implementing self-management for the purpose of system integrity.

The green and red lines are the request rate trends from Oslo and Gjøvik. The yellow curve is the log file growth rate, and the black curve shows the current user activity, both of them are logged by the analyzer. The blue square wave is again the round trip time taken from Gjøvik, showing the behavior of virtual machine.

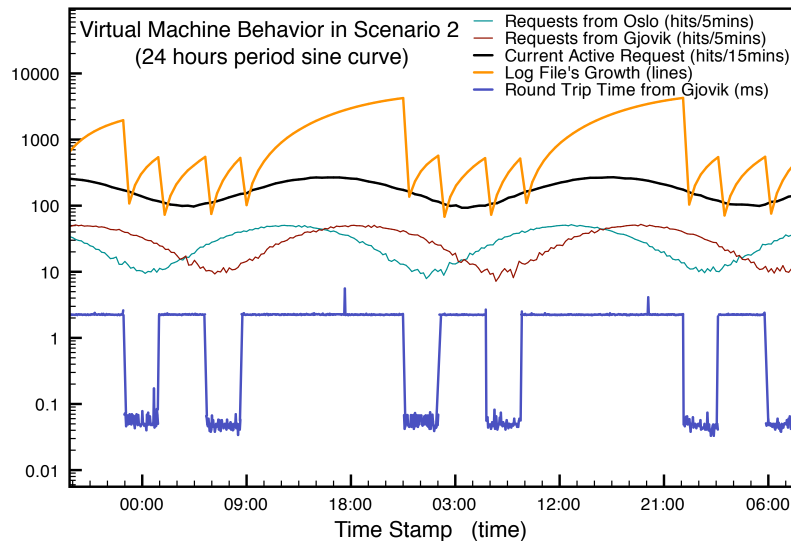


Figure 6.9: Virtual Machine Migration in System Integrity

Compared with figure 6.8a, although they use the same basic request generator, the different self-management purposes and policies cause the virtual machine to behave completely different.

As a conclusion, both input parameters, request rate trends, and self-management goals influence the virtual machine behavior. How to use the known input parameters

and self-management purpose to predict virtual machine behavior and define suitable policy, is a significant and useful approach. Like the proverb says: "Give a man a fish; you have fed him for today. Teach a man to fish; and you have fed him for a lifetime." Therefore, how to predict virtual machine behavior and define suitable policies to achieve different needs is more important than only implementing and analyzing few scenarios.

In the following discussion chapter, section 7.8, we mainly focus on the probability of virtual machine's migration.

That is also one of the purposes of this project: showing other system administrators how to implement self-management based on the regular administration tools. The probability of migration is analyzed and discussed in the section 7.8.

Chapter 7

Discussion

7.1 Simulation versus Realistic of Experiment Environment

7.1.1 The Reason Why to Implement the Experiment in Realistic

Generally, experiments are implemented in a simulated environment to show the eventual real effects of alternative conditions and courses of action. In the simulated environment, we can select key characters and behaviors of the realistic world, and simplify approximations and assumptions. Consequently, a simulated environment is easier to control and more ideal than realistic. However, the products made in simulation might not be able to handle the real complex situations. This issues also make some inventions hard to be implemented into practical scenarios.

Therefore, this experiment is implemented in the real global infrastructure. Various uncertainties in realistic bring more challenges and discussions while designing and implementing this experiment. For more important sense, this project shows how to implement self-management and virtualization into real system administration. It can be a good suggestion and inspiration for system administrators optimizing their system.

There are also some parts simulated in this project, such as generating the user requests. This simulation is based on the realistic investigate and conclusion. The reason to simulate user requests is that we do not have enough time to create an attractive web page for gathering enough data. Also, this is not the main function of this experiment. Spending too much time in realizing this is not reasonable.

7.1.2 Practical Problems

A few practical problems were encountered throughout the project:

1. Wallingford network: The network link between Wallingford and Norway is not as stable as expected. When migrating from Wallingford back to Oslo or Gjøvik, the traffic slows down and even halts. This appears to be because of the long distance and somewhat poor links. Sometimes, the migration from Wallingford even

causes lost network connections. So although this experiment is configured and implemented globally, the scenarios are tested mostly between Oslo and Gjøvik.

2. Firewall at Gjøvik: The firewall policy in Gjøvik University College is very strict. There is only one port open on the virtual server. When the virtual machine migrates to Gjøvik, the user requests from Oslo are completely blocked. A consequence of this is that the virtual machine will never be able to migrate away from Gjøvik. With the help from Erik Hjelmås and Jon Langseth, one client with a specific IP address from Oslo was registered and unblocked in the Gjøvik firewall.

3. VPN tunnel died under high traffic load: When testing the VPN tunnel by copying the full virtual machine image file to different locations, the tunnel died because of the high traffic load. After searching and debugging, Vtun was set to run with the parameter "m". This parameter is used to force memory pages to be locked memory-resident to prevent potential VM deadlock. It is especially useful with NFS traffic. It should be noted that the "m" parameter has no configuration file equivalent.

7.2 Defining Good Backup

In the second scenario, how to define a good backup scheme is a challenge.

Generally, system administrators choose to run backups during the night frequently to minimize system load or because the backup should take a full day's changes into account. Since there are few connections during this time, less options also means less risks. The chance of losing data is much lower than the daytime as well. This points out the fact that when the servers start to do the back up, the data might have got lost already.

Another way to do the backup is to back up the data only before any risky operations. This method can be called autonomic as well, since the server needs to monitor the environment, and analyze the information to detect risks. It seems like an an ideal model, since it is very effective: It only starts the back up when needed, which means just before the data is likely to be lost. However, the problem for this method is that the backup operation might cause more traffic, CPU and memory utilization, which will push the servers to the down time.

Consequently, in this experiment, good backup is defined as backing up the data when the data has changed sufficiently since the last backup and there are few clients connected to the server. This method requires self-management, so when and where to do the backup is up to the virtual machines' decision process. Compared with the former scheme, this method does the backup more effectively. While contrast with the latter scheme, this method will not cause too much interrupts and damages.

7.3 Keeping the Service Available

The purpose of live migration is to keep services available at all times. This means while migrating a virtual machine, the service in the virtual machine is still running. When migrating virtual machines inside a local network, its IP address need not be changed. This is also an important factor to keep the service available without interruption.

However, in a global network using IPv4 the virtual machine's IP address has to be changed after migration to attach to a new network. This issue can be solved by using IPv6 [31] [32], but IPv6 is still not widely deployed, so it is not configured in this experiment.

An alternative is to use dynamic DNS to restore service availability after migration. Instead of connecting directly to the IP address, clients connect to the service by first looking up its registered domain name. A dynamic DNS client is run to update the dynamic IP address for the domain after migration has changed the IP address. This method does not solve the problem completely, since the service is not reachable during IP and dynamic DNS update. However, the virtual machine is alive all the time even it is not reachable for a short while.

7.4 Optimizing Generator Scripts

A big issue with the user requests generator is that it is time consuming. The current generator is designed to start at 00:00 of each individual location every day. Consequently, if one of the clients is having problems and fails to start the generator, the entire day will be wasted. Various reasons can cause problems for the generator, and there were several occurrences that required generator restart during the experimental phase.

We have tried to simulate one real day as one virtual week in this experiment. However, the migration time will then correspond to several hours when shown in the graphs, instead of several seconds or minutes in real life. This causes inconsistencies in the result data, so this simulation was not implemented.

An approach to solve this problem is to modify the generator that can start at any given offset of the day. For example, if the generator is started at 9:00 in the morning, instead of sending 0 hits, it will generate the data as it should be. Because of time constraints, this function is not present in the scripts. It should be added as an advanced function in the future. If this project is implemented with real user behavior later, this problem is resolved.

7.5 Problems Caused by Time Shift in Virtual Machine

As mentioned before, when migrating virtual machines to different locations, the virtual machine's system time fails to synchronize with the dom0 clock. So, in one location, the system time is synchronized correctly, while in other locations, the system time

shifts with some constant value. For example, when the virtual machine is located in Gjøvik, the system time is fully synchronized with the dom0 host. When it migrates to Oslo, the time will shift 2756849 seconds back in time. So the log file written by web server monitoring script will seem like this:

128.39.73.147	1210456211	001
128.39.73.147	1210456212	001
128.39.73.147	1210456213	001
128.39.73.147	1210456214	001
128.39.143.99	1207699695	002
128.39.143.99	1207699696	002
128.39.143.99	1207699697	002
128.39.143.99	1207699698	002
128.39.73.147	1207699715	001

This problem is reported as an existing bug (no.195) [33] currently unresolved in Xen. Evidently, it causes severe problem in this experiment.

All the time stamps in the log files jump back and forth before and after migration. Since every analysis script determines which log lines are relevant based solely on the time stamp, this bug causes failure in all the analysis scripts. Consequently, the decision-making does not get proper information, so the action function will never have a chance to be executed.

To solve this problem, one synchronize_time script is developed. It will be run every time before the analysis scripts are executed. It synchronizes all the time stamps in the log files so that they correspond to the current time of the virtual machine. This means the time stamps in the log file are completely different when the virtual machine changes location. The synchronize_time script is invoked in the cfengine configuration file:

```
1  #!/usr/sbin/cfagent -K -f
2  # this is where we gather information and make policy
3  control:
4   actionsequence = (  editfiles shellcommands )
5
6   synchronizetime = ( ExecShellResult("/var/www/analyze/sychronize_time.pl") )
7   userpercentage = ( ExecShellResult("/var/www/analyze/user/user_percentage.pl") )
8   usercode = ( ExecShellResult("/var/www/analyze/user/user_code.pl") )
9
10  migrationthreshold = ( 0.6 )
```

After this change, all the self-management scripts will function well. The script for synchronizing the time can be found in appendix D.3.

7.6 Using Cfengine for Decision Making

Cfengine is used for the decision making function because it has following benefits matching our requirements:

- It can execute commands and scripts. This helps us to manage our scripts and invoke them in an easy and secure way.
- It can keep the system convergent, which means it can always move the system towards its desirable state.
- It has been developed for a long time with a solid track record.
- It gives a simple way to describe policy. We can leave only the cfengine configuration file for users to define policy, and other scripts will not be interrupted and changed by users.

Although cfengine has so many benefits, it still has some limits for executing some actions. That is also the reason why we do not use it for the migration and backup functions.

Cfengine is a simple and user-friendly language. However, it can not handle very specialized actions, for example, controlling the time interval between two actions. For this reason, we have to wrap the virtual machine actions into external scripts to fit special needs.

7.7 Problem Caused by Update IP Address

As presented before, three basic steps need to be executed by the virtual machine: migration, renewing IP address and updating the dynamic DNS entry. The problem is that the virtual machine does not itself know when the migration has finished and when it is connected to the new network. Hence, how to make the decision about when and how often to run the "dhclient" and "ddclient" to update the IP address and domain name becomes an issue. If running the commands too often, they might interrupt the virtual machine's migration. If running the commands too rarely, the web service would be unreachable for extended time periods.

After running the migration command, we define a reasonable idle period to let the migration complete. Then the virtual machine will run "dhclient" to try to get the new IP address. Before updating the IP address to the dynamic DNS server, the virtual machine continues sending requests and checks the IP address until it detects that it has changed subnets.

This design makes the actions more effective, and avoids unnecessary work. For example, the dynamic DNS entry is only updated when the IP address has changed. However, the "dhclient" loop consumes a lot of memory since it spawns new daemon processes on each iteration.

This issue is fixed by killing all the running "dhclient" processes when running the action files. Therefore, in the migration actions, the virtual machine will first do the migration and then run the command "killall dhclient" before getting new IP address.

7.8 The Probability of Migration

The probability of executing a migration depends on the probability of different predefined policies. By analyzing this probability, we can predict the virtual machine's behavior and also define suitable policies in the decision-making process. Since the second scenario has a more complicated policy, we mainly focus on the probability of migration in the service optimization scenario.

In this scenario, we define two policy parameters: highest user requests percentage, and the trend of the user activity. Migration only happens when the highest requests percentage is higher than the policy, and the requests coming from the highest request location is still increasing.

7.8.1 Analyze Total Request Trend

First of all, we define the formula which will be used for generating user requests. Take the basic sine curve as an example:

$$Hits_{(i)} = A \cdot sin\left(\frac{2\pi}{86400}(x - t_{diff(i)}) - \frac{\pi}{2}\right) + C \tag{7.1}$$

In the formula, $t_{diff(i)}$ means the time difference between different locations. It varies with different standard starting times. However, it is a constant value if the locations are decided. So each t_{diff} defines one $Hits_{(i)}$ formula.

Therefore, the total request rate monitored in the server side is:

$$\begin{aligned} Total_Hits &= \sum_{i=1}^{n} Hits_{(i)} \\ &= \sum_{i=1}^{n}\left(A \cdot sin\left(\frac{2\pi}{86400}(x - t_{diff(i)}) - \frac{\pi}{2}\right) + C\right) \end{aligned} \tag{7.2}$$

This Total_Hits is still a sine curve, which can be proved as follows:

$$\begin{aligned}
Total_Hits &= \sum_{i=1}^{n} Hits_{(i)} \\
&= Hits_{(1)} + Hits_{(2)} + ... \\
&= \left(A \cdot sin \left(\frac{2\pi}{86400} x - \frac{\pi}{2} \right) + C \right) + \left(A \cdot sin \left(\frac{2\pi}{86400} (x - t_{diff1}) - \frac{\pi}{2} \right) + C \right) + ... \\
&= A \cdot \left(sin \left(\frac{2\pi}{86400} x - \frac{\pi}{2} \right) + sin \left(\frac{2\pi}{86400} (x - t_{diff1}) - \frac{\pi}{2} \right) \right) + 2C + ... \\
&= 2A \cdot sin \left(\frac{1}{2} \cdot \left(\frac{4\pi}{86400} x - \pi - \frac{2\pi}{86400} t_{diff1} \right) \right) \cdot cos \left(\frac{1}{2} \cdot \frac{2\pi}{86400} t_{diff1} \right) + 2C + ... \\
&= 2A \cdot sin \left(\frac{2\pi}{86400} x - \frac{\pi}{2} - \frac{\pi}{86400} t_{diff1} \right) \cdot cos \left(\frac{\pi}{86400} t_{diff1} \right) + 2C + ...
\end{aligned}$$

When the value of t_{diff1} is defined, $cos\left(\frac{\pi}{86400} t_{diff1}\right)$ becomes a constant value. So the summary of first two Hits is still a sine curve. Thus we arrive at the conclusion that the summary of the hits from all locations is always a sine curve.

7.8.2 Highest Request Percentage

The request percentage from each location is:

$$P_i = \frac{Hits_{(i)}}{\sum_{i=1}^{n} Hits_{(i)}} \tag{7.3}$$

Then we can represent the highest request percentage (P_h) in this way:

$$P_h = \{P_i < P_h\} \qquad \text{for} \forall Hits_{(i)} \in \{Hits_{(n)}\}$$

7.8.3 Highest User Activity Boolean

To show another decision policy, the trend of user activity, we need to determine the slope of the curve. We need to calculate the derivative of $Hits_{(n)}$, which shows the trend of each request curve.

$$\begin{aligned}
\frac{d' Hits_{(i)}}{dx} &= \left(A \cdot sin \left(\frac{2\pi}{86400} (x - t_{diff}) - \frac{\pi}{2} \right) + C \right)' \\
&= -A \cdot cos' \left(\frac{2\pi}{86400} (x - t_{diff}) \right) \\
&= A \cdot sin \left(\frac{2\pi}{86400} (x - t_{diff}) \right)
\end{aligned} \tag{7.4}$$

When the derivative is positive, the user activity is increasing. Correspondingly, when the derivative is negative, the user activity is decreasing. Figure 7.1 shows the

user activity boolean. This boolean only focuses on the user activity from the highest request location.

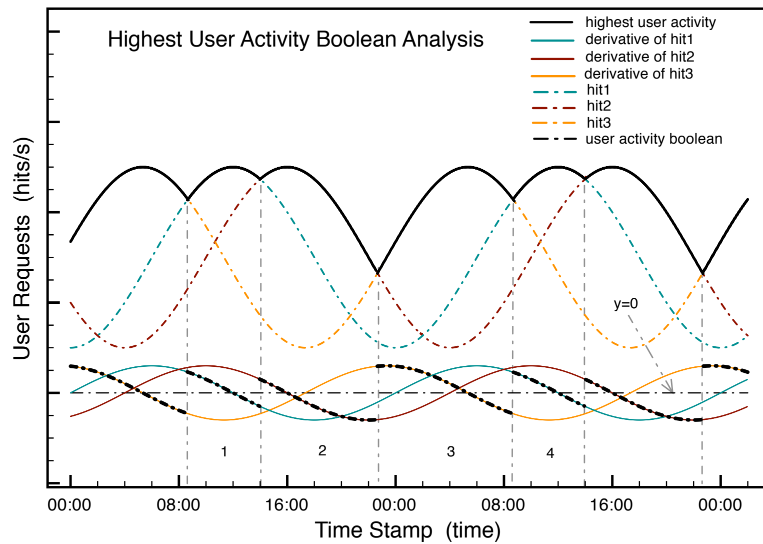


Figure 7.1: Highest User Activity Boolean Analysis

In the area marked 1, the highest request rate is from hit1, so the boolean is shown as the derivative of hit1 in this area. It can be assigned both 1 and 0 in this range. The same happens in area 2 and 3: the boolean is represented as the derivative of hit2 and hit3.

Therefore, when the derivative is positive, the boolean value returns 0, which means the policy is reached. When the derivative is negative. the boolean value returns 1, which means the policy is not satisfied.

7.8.4 The Probability of Migration

If we want to analyze the probability of virtual machine migration, we need to consider the probability of both policies.

Figure 7.2 shows the possible areas where migration might occur. As the two policies we defined, highest percentage and user activity boolean, migration will only happen when the policy of percentage exist in the highest percentage area, meanwhile, the user activity is still increasing. The yellow area shows when the two policies are reached, which is the possible area of migration.

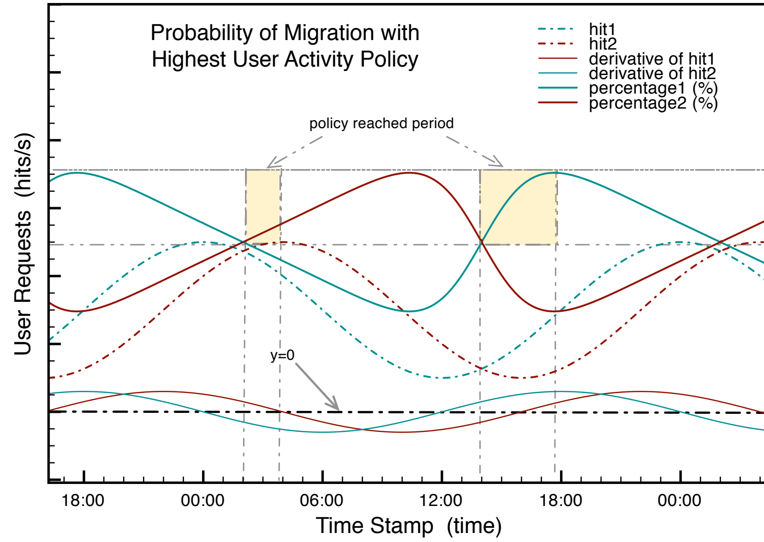


Figure 7.2: Probability of Migration

7.8.5 Other Parameter for Policy

There are other parameters that can be used as a basis for migration policies as well. For example, the total user activity.

Total user activity means the total request rate trend of all clients, which also can be a boolean value indicating general increase or decrease. This parameter differs from what we defined and implemented in this experiment – highest user activity – which only shows the request rate trend from the highest request location.

In this case, we need to calculate the derivative of total requests $Total_Hits$:

$$\frac{d'Total_Hits}{dx} = \left(\sum_{i=1}^{n} Hits_{(i)}\right)' \tag{7.5}$$

We'll use two locations as an example, thus the request from each location is:

$$Hits1 = A \cdot sin\left(\frac{2\pi}{86400}x - \frac{\pi}{2}\right) + C$$
$$Hits2 = A \cdot sin\left(\frac{2\pi}{86400}(x - t) - \frac{\pi}{2}\right) + C \tag{7.6}$$

Therefore, the total requests from the server side is:

$$
\begin{aligned}
Total_Hits &= Hits1 + Hist2 \\
&= \left(A \cdot sin\left(\frac{2\pi}{86400}x - \frac{\pi}{2}\right) + C\right) + \left(A \cdot sin\left(\frac{2\pi}{86400}(x-t) - \frac{\pi}{2}\right) + C\right) \\
&= A \cdot \left(sin\left(\frac{2\pi}{86400}x - \frac{\pi}{2}\right) + sin\left(\frac{2\pi}{86400}(x-t) - \frac{\pi}{2}\right)\right) + 2C \\
&= 2A \cdot sin\left(\frac{2\pi}{86400}x - \frac{\pi}{2} - \frac{\pi}{86400}t\right) \cdot cos\left(\frac{\pi}{86400}t\right) + 2C
\end{aligned} \tag{7.7}
$$

So the derivative of total requests is:

$$
\begin{aligned}
\frac{d'H}{dx} &= 2A \cdot \left(sin\left(\frac{2\pi}{86400}x - \frac{\pi}{2} - \frac{\pi}{86400}t\right) \cdot cos\frac{\pi}{86400}t + 2C\right)' \\
&= 2A \cdot \left(sin'\left(\frac{2\pi}{86400}x - \frac{\pi}{2} - \frac{\pi}{86400}t\right) \cdot cos\frac{\pi}{86400}t + sin\left(\frac{2\pi}{86400}x - \frac{\pi}{2} - \frac{\pi}{86400}t\right) \cdot cos'\frac{\pi}{86400}t\right) \\
&= 2A \cdot sin'\left(\frac{2\pi}{86400}x - \frac{\pi}{2} - \frac{\pi}{86400}t\right) \cdot cos\frac{\pi}{86400}t \\
&= 2A \cdot \left(-cos\left(\frac{2\pi}{86400}x - \frac{\pi}{86400}t\right)\right)' \cdot cos\frac{\pi}{86400}t \\
&= 2A \cdot sin\left(\frac{2\pi}{86400}x - \frac{\pi}{86400}t\right) \cdot cos\frac{\pi}{86400}t
\end{aligned} \tag{7.8}
$$

Based on the analysis above, the total user activity can be represented in the figure 7.3. When the derivative is positive, the total request rate is increasing. When the derivative is negative, the total request rate is decreasing.

Therefore, the probability of migration in this case is much higher than the policy we used in the experiment.

If we define both of the highest percentage policy and the total user activity policy to test the first scenario, the migration will never get chance to happen. See figure 7.4: The yellow area shows when migration is possible. However, all the possible migration periods happen in hits1, which means in location 1. Location 2 never gets a chance to host the virtual machine. Therefore, migration will never happen.

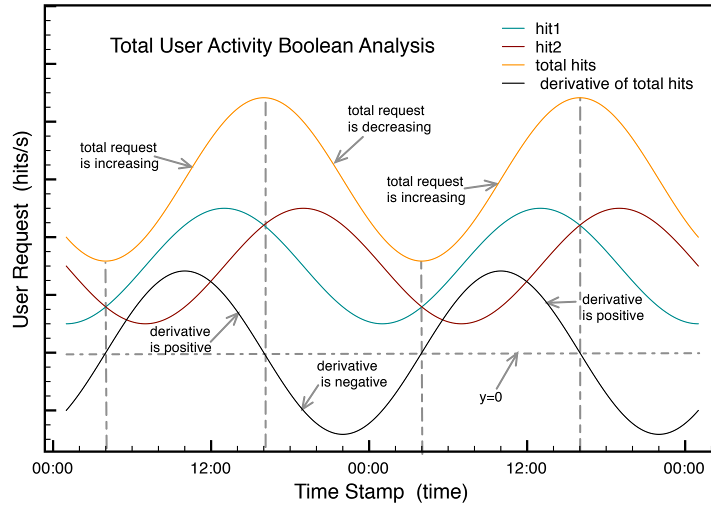


Figure 7.3: Total User Activity Boolean Analysis

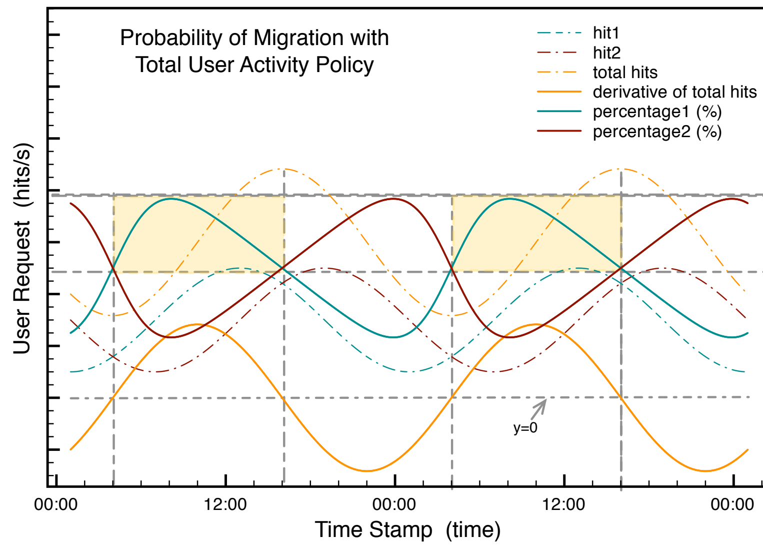


Figure 7.4: Probability of Migration with Total Request Boolean as Policy

7.9 System Convergency

According to the definition of convergence, section 2.9, we can consider our system to be convergent.

There are four start states that can happen in this experiment:

- The request for migration has not been reached.
- The request is reached, but the virtual machine is already at the place where it should be.
- The request is reached, virtual machine will migrate to a new location.
- The virtual machine is "broken" because of some serious issue.

The only final state is:

- The virtual machine is at the place where it should be. This meaning differs in the different scenarios.

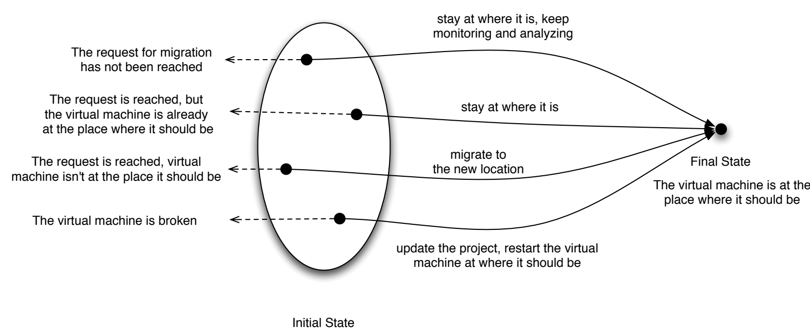


Figure 7.5: System Convergency

As shown in figure 7.5, no matter what the virtual machine's initial state is, it can manage itself to the final state – staying at the place where it should be.

7.10 System Drawback

Although the system can function well, there are still some drawbacks.

Lack of Control

The biggest drawback of this system is the lack of a stable network. Since the experiment is implemented on a global scale, we need to remote control other servers that are not in Oslo. The stability of the network influences the progress of the entire project. If the network or servers go down, getting remote help is consuming considerable amounts of time.

Network Limit

Because of the IPv4 problem, the virtual machine appears disconnected while migrating. This drawback reduces one big benefit of live migration: keeping the service continuously online.

Implementation Limit

Also related to the network problem, dynamic DNS is used to keep the service online when switching subnets. Dynamic DNS only allows one IP related to one domain name. Consequently, we cannot implement more virtual machines to cooperate with each other in this self-management system. An alternative would be IP-based load balancing, but this falls outside the scope of the project.

Demanding High Quality Networks

In order to reduce the down time for the web service, the network throughput must be as high as possible. If the network security policy is too strict, e.g. due to rate-limiting firewalls, the destination server will spend a long time trying to migrate the virtual machine. This issue happened between the Wallingford and Oslo servers. It normally takes 4 minutes to migrate virtual machines from Oslo to Wallingford. However, because of the network quality and policy problems in Wallingford, Oslo needs around 13 minutes to get the virtual machine back. Therefore, the test of virtual machine behavior is normally taken between Oslo and Gjøvik.

Sensitive to Noise

Noise is added in the generator to simulate realistic user behavior. However, on a small scale, this noise sometimes influences the trend enough to trigger unnecessary migrations that otherwise should not happen according to the analysis. Therefore, our system is noise sensitive.

This issue should be easy to fix by higher the amplitude of requests generator, and lower the noise mean value. Other flap-detection or dampening approaches are also possible.

Time Consuming

This project is time consuming, especially when doing the scenario tests. A successful test needs at least 30 hours to show the virtual machine behavior. If any issues occur while doing the testing, such as the generator being interrupted for some reason, an entire day is waisted.

Chapter 8

Conclusion

> *"System administration is like keeping the train on track. Nobody will notice it until the train got delayed."*
>
> *–Æleen Frisch*

System administration is a complex task. It is a combination of hardware, software, network, system, configuration, management as well as all kinds of protocols and standards. A system administrator's tasks could easily become tedious and repetitive. The primary goal of this thesis is to make simple approaches to implement self-management functions based on the regular administration tools as a way to simplify tedious tasks for the system administrator.

The implementation of this experiment includes setting up the system infrastructure, developing self-management functions, and testing the system with different scenarios.

When designing and building the system infrastructure, only regular administration tools and technologies are used: NFS, VPN, Xen, MLN, Cfengine and dynamic DNS. These general administration tools makes the learning curve more friendly for admins, and they are very easy to install and configure in real production environments. From a higher point of view, this project introduces a way of optimizing existing systems and networks with self-managing functions. It also shows the steps how to transform an autonomic model into an operational strategy, and then finally implement it into real system administration scenarios.

A fully functioning self-management system was developed for two essential administration tasks in this project: service optimization and system integrity.

Each virtual machine can make its own decision when and where to migrate itself between the physical nodes in order to reach the predefined policies. Live virtual machine migration is the main action to be executed here for self-management purposes. This method minimizes the maintenance tasks for the system administrators, only leaving the high-level administrative policies for the users to decide.

The virtualization-based environment provides a flexible platform for deploying autonomic technologies. The self-management functionality developed in this experiment presents a possible way to optimize the regular administration tools for different self-management purposes. Other than these two tasks, live migration and external backup, other users can develop their own self-management functions to reach their individual goals, such as better power management and intrusion detection.

By testing with various scenarios, this system shows good scalability since it can deal with different user requests and achieve different management goals. A detailed analysis shows the probability of executing self-management actions, which can be used to predict virtual machines behavior, as well as to define suitable policies.

8.1 Future Work

This project only implements self-management into a single virtual machine. In the future, we can think about how to implement self-management function into multiple networked virtual machines.

This kind of network and system can be easily set up and maintained by MLN. However, it requires more self-management functions. For instance, the monitoring function should not only gather information from the clients, but also communicate with other virtual neighbor nodes. When virtual machines analyze the data and plan courses of action, more decision-making policies have to be involved, such as communicating with destination host to check if it has enough resources to handle one extra virtual machine.

By implementing self-management functionality for all virtual machines, we can set up a self-managing network. Take FTP as an example. Multiple virtual machines are offering the FTP service with one domain name, and they synchronize the available files continuously. They can be hosted in different locations to balance the global work load or be hosted at one place for some high level request. With the self-management functions, when the user request trends change, each virtual machine can decide whether to migrate to different locations or gather in one place depending on their own policy. This benefit gives system administrators more opportunity to optimize their services. For a further view, after this kind of self-management system is implemented, there is not much need for human attention. In words, it can reduce the administration maintenance tasks significantly.

Bibliography

[1] Christopher Strachey. Time sharing in large fast computers. In *IFIP Congress*, pages 336–341, 1959.

[2] John McCarthy. Reminiscences on the history of time sharing. *IEEE Annals of the History of Computing archive*, 14, January 1992.

[3] Dan Marinescu and Reinhold Kroger. State of the art in autonomic computing and virtualization. *http://wwwvs.informatik.fh-wiesbaden.de*, September 1, 2007.

[4] Vmware. Virtualization basics. http://www.vmware.com/overview/why.html, February 14, 2008.

[5] Kyrre Begnum and Matthew Disney. Scalable deployment and configuration of high-performance virtual clusters. 2006.

[6] Sandya Mannarswamy. A hitchhiker's guide to virtualization. *Linux for You*, 05(08):54–56, October 2007.

[7] Fraser Keir Hand Steven Harris Tim Ho Alex Neugebauer Rolf Pratt Ian Barham Paul, Dragovic Boris and Warfield Andrew. Xen and the art of virtualization. *ACM Press*, page 164–177.

[8] Vishnu Saran M. Xen live migration using iscsi target. *Linux for You*, 05(08):57–59, October 2007.

[9] Jeffrey O. Kephart and David M. Chess. The vision of autonomic computing. *Computer*, 36(1):41–50, January, 2003. DOI: 10.1109/MC.2003.1160055.

[10] Seif Haridi Jean-Bernard Stefani Thierry Coupaye Alexander Reinefeld Ehrhard Winter Peter Van Roy, Ali Ghodsi and Roland Yap. Self-management of large-scale distributed systems by combining peer-to-peer networks and components. *Electronic Notes in Theoretical Computer Science (ENTCS)*, 182:201–217, June, 2007. DOI: 10.1016/j.entcs.2006.12.043.

[11] Mark Burgess. *Analytical Network and System Administration*. Wiley, May 28, 2004.

[12] Botvich D. Strassner J.-Lehtihet E. Donnelly W. Agoulmine N., Balasubramaniam S. Challenges for autonomic network management. October 2006.

[13] Jacques Ferber. *Multi-Agent Systems*. Addison-Wesley, 1999.

[14] Maxim Krasnyansky. Virtual tunnel home page. http://vtun.sourceforge.net/.

[15] Ryan Breen. Vtun. http://www.linuxjournal.com/article/6675, August 1st, 2003.

[16] Steven Hand Jakob Gorm Hansen Eric Jul Christian Limpach Ian Pratt Christopher Clark, Keir Fraser and Andrew Warfield. Live migration of virtual machines. May 2005.

[17] Kyrre M Begnum. Managing large networks of virtual machines. *USENIX Association Berkeley*, pages 16 – 16, 2006.

[18] John Secrest Kyrre M Begnum. Getting started with mln - 0.80. http://mln.sourceforge.net/doc/mln-getting-started.pdf, April 2 2006.

[19] John Secrest Kyrre M Begnum. The mln manual mln version 0.80. http://mln.sourceforge.net/doc/mln-manual.pdf, April 2, 2006.

[20] Kyrre M Begnum. The mln introduction web page. http://mln.sourceforge.net/index.php?page=Introduction.

[21] Mark Burgess and Æleen Frisch. *A System Engineer's Guide to Host Configuration and Maintenance Using Cfengine*. USENIX Association, 2007.

[22] Mark Burgess. Automated system administration with feedback regulation. *Source Software—Practice and Experience archive*, 28(14):1519 – 1530, December 1998.

[23] Mark Burgess. Cfengine.org. http://www.cfengine.org/confdir/getstarted.html.

[24] Mark Burgess Kyrre Begnum and John Sechrest. Adaptive provisioning using virtual machines and autonomous role-based management. *Proceedings of the International Conference on Autonomic and Autonomous Systems*, 2006. ISBN:0-7695-2653-5.

[25] Espen Braastad. Management of high availability services using virtualization. *http://research.iu.hio.no/theses/master2006/*, 2006.

[26] Falko Timme. The perfect xen 3.0.1 setup for debian version 1.0. http://howtoforge.com/perfect_setup_xen3_debian, March 20,2006.

[27] Falko Timme. Installing xen on an ubuntu 7.10 (gutsy gibbon) server from the ubuntu repositories version 1.0. http://howtoforge.com/ubuntu-7.10-server-install-xen-from-ubuntu-repositories, October 30, 2007.

[28] Stephan Böni. Installing xen3. http://en.opensuse.org/Installing_Xen3, February 19, 2008.

[29] Christopher Smith. Nfs howto guide. http://nfs.sourceforge.net/nfs-howto/, May 2, 2006.

[30] google earth. Google earth. http://earth.google.com/.

[31] Darrin Miller Sean Convery. Ipv6 and ipv4 threat comparison and bestpractice evaluation (v1.0). *www.cisco.com/security_services/ciag/documents/v6-v4-threats.pdf*, 2006.

[32] Yurie Rich Jeff Doyle. Ipv6 an internet evolution. *http://cav6tf.org/articles/ipv6-aninternetevolution.pdf*, January 14, 2003.

[33] Alexander Junghans. Many timer isr: Time went backwards logs. http://bugzilla.xensource.com/bugzilla/show_bug.cgi?id=195, May 7, 2008.

Appendix A

Network and Environment Implementation

A.1 MLN Configuration File for Creating Virtual Machine and Networks

```
global {
        project roam
        project_password *****
}

host roamer {
        xen
        memory 64M
        term screen
        filepath /nfssan
        template roamvm.ext2
        service_host oslo
        network eth0 {
                address dhcp
        }
}

oslo_client {
        xen
        memory 64M
        term screen
        filepath /nfssan
        template oslovm.ext2
        service_host oslo
        network eth0 {
                address dhcp
        }
}

gjovik_client {
        xen
        memory 64M
        term screen
        filepath /nfssan
```

```
35          template gjovikvm.ext2
36          service_host gjovik
37          network eth0 {
38                  address dhcp
39          }
40  }
41
42  wallingford_client {
43          xen
44          memory 64M
45          term screen
46          filepath /nfssan
47          template wallingfordvm.ext2
48          service_host wallingford
49          network eth0 {
50                  address dhcp
51          }
52  }
```

A.2 MLN Configuration File for Virtual Machine Migration

Only the migrate_to_oslo file is shown here, other files are similar, only the "service_host" is changed.

```
1   global {
2           project roam
3           project_password *****
4   }
5
6   host roamer {
7           xen
8           memory 64M
9           term screen
10          filepath /nfssan
11          template roamvm.ext2
12          service_host oslo
13          network eth0 {
14                  address dhcp
15          }
16  }
```

A.3 MLN Configuration File for Virtual Machine Backup

Only the backup_in_oslo file is shown here, other files are similar, only the "service_host" is changed.

```
1   global {
2           project roam
3           project_password *****
4   }
5
6   host roamer {
7           xen
8           memory 64M
9           term screen
```

```
10         filepath /nfssan
11         template roamvm.ext2
12         service_host oslo
13         network eth0 {
14                 address dhcp
15         }
16         xenLiveDisk {
17                 /home/lux/xenodrive /dev/hdc w 1400M
18         }
19 }
```

A.4 VTun Configuration File for VPN Tunnel

VTun configuration file for server side.

```
1  options {
2    # Listen on this port.
3    port 15000;
4    # Syslog facility
5    syslog        daemon;
6
7    # Path to various programs
8    ppp           /usr/sbin/pppd;
9    ifconfig      /sbin/ifconfig;
10   route         /sbin/route;
11   firewall      /sbin/iptables;
12   ip            /sbin/ip;
13 }
14
15 # Default session options
16 default {
17   compress no;          # Compression is off by default
18   speed 0;              # By default maximum speed, NO shaping
19 }
20
21 # set virtual tunnel for oslo
22 mln7 {
23   passwd  ****;                 # Password
24   type  ether;          # Ethernet tunnel
25   device tap2;          # Device tap0
26   proto udp;            # UDP protocol
27   compress  lzo:9;      # LZO compression level 1
28   stat  yes;            # Log connection statistic
29   keepalive yes;        # Keep connection alive
30 }
31 up {
32     ifconfig "%%10.0.0.2 pointopoint 10.0.0.1 mtu 1200";
33 };
34 down{
35     ifconfig "%%down";
36 };
37
38 # set virtual tunnel for wallingford
39 wallingford {
40   passwd  ****;             # Password
41   type  ether;          # Ethernet tunnel
42   device tap0;          # Device tap0
43   proto udp;            # UDP protocol
44   compress  lzo:9;      # LZO compression level 1
```

```
45   stat  yes;              # Log connection statistic
46   keepalive yes;          # Keep connection alive
47 };
48 up {
49     ifconfig "%%10.0.0.4 pointopoint 10.0.0.1 mtu 1200";
50 };
51 down{
52     ifconfig "%%down";
53 };
54
55 # set virtual tunnel for gjovik
56 gjovik {
57   passwd  ****;                  # Password
58   type  ether;            # Ethernet tunnel
59   device tap0;            # Device tap0
60   proto udp;              # UDP protocol
61   compress  lzo:9;        # LZO compression level 1
62   stat  yes;              # Log connection statistic
63   keepalive yes;          # Keep connection alive
64 };
65 up {
66     ifconfig "%%10.0.0.3 pointopoint 10.0.0.1 mtu 1200";
67 };
68 down{
69     ifconfig "%%down";
70 };
```

VTun configuration file for client side, only Oslo side is shown here.

```
1  # set virtual tunnel for oslo
2  mln7 {
3    passwd  ****;                  # Password
4    type  ether;            # Ethernet tunnel
5    device vtun;            # Device tap0
6    proto udp;              # UDP protocol
7    compress  lzo:9;        # LZO compression level 1
8    stat  yes;              # Log connection statistic
9    keepalive yes;          # Keep connection alive
10 };
11
12 up {
13     ifconfig "%%10.0.0.2 pointopoint 10.0.0.1 mtu 1200";
14 };
15
16 down{
17     ifconfig "%%down";
18 };
```

Appendix B

Generating User Requests

B.1 Generator for 24 Hours Period Sine Curve

```
1  #!/usr/bin/perl
2  use warnings;
3  use strict;
4  use LWP::Simple;
5  use Math::Trig;
6  use Math::Random::OO::Normal;
7
8  for (;;){
9    #it generates value with 0 as mean, 0.15 as standard deviation.
10   my $grand = Math::Random::OO::Normal->new(0,0.15);
11
12   # just keep seed at random value
13   $grand->seed(rand);
14
15   my $x;
16
17   for ( $x = 0; $x <= 86400; $x += 300 ){
18
19          #formular for sine wave hits:
20          my $hits = 20 * sin ( ( 2 * $x * pi / 86400 ) - pi / 2 ) + 30;
21
22          #noise:
23          my $noise = $grand->next() * 5;
24
25          # get system time
26          my $timestamp = time();
27
28          my $data = $hits + $noise;
29
30          open FILE, ">>log.txt" or die $!;
31          print FILE "$timestamp  $data  \n";
32          close FILE;
33
34          my $i = 0;
35          while ( $i <= $data) {
36                  get("http://roamer.dynalias.org/roamer.php");
37                  $i++;
38                  sleep 1;
39          }
40
```

```
41          my $diff = time();
42          my $time_diff = 300 - int($diff - $timestamp);
43          sleep $time_diff;
44    }
45  }
```

B.2 Generator for 8 Hours Period Sine Curve

```
1  #!/usr/bin/perl
2  use warnings;
3  use strict;
4  use LWP::Simple;
5  use Math::Trig;
6  use Math::Random::OO::Normal;
7
8  for (;;){
9    my $x;
10   for( $x =0; $x <=28800; $x +=300 ){
11          my $data = rand()*10;
12          my $timestamp = time();
13
14          open FILE, ">>log_8hs.txt" or die $!;
15          print FILE "$timestamp $data \n";
16          close FILE;
17
18          my $i = 0;
19          while ( $i <= $data) {
20                  get("http://roamer.dynalias.org/roamer.php");
21                  $i++;
22                  sleep 1;
23          }
24
25          my $diff = time();
26          my $time_diff = 300 - int($diff-$timestamp);
27          sleep $time_diff;
28   }
29
30   #it generates value with 0 as mean, 0.2 as standard deviation.
31   my $grand = Math::Random::OO::Normal->new(0,0.2);
32
33   # just keep seed at random value
34   $grand->seed(rand);
35
36   for ( $x = 0; $x <= 28800; $x += 300 ){
37          #formular for sine wave hits:
38          my $hits = 20 * sin ( ( 2 * $x * pi / 28800 ) - pi / 2 ) + 30;
39
40          #noise:
41          my $noise = $grand->next() * 5;
42
43          # get system time
44          my $timestamp = time();
45
46          my $data = $hits + $noise;
47
48          open FILE, ">>log_8hs.txt" or die $!;
49          print FILE "$timestamp $data \n";
50          close FILE;
```

```
51
52          my $i = 0;
53          while ( $i <= $data) {
54                  get("http://roamer.dynalias.org/roamer.php");
55                  $i++;
56                  sleep 1;
57          }
58
59          my $diff = time();
60          my $time_diff = 300 - int($diff-$timestamp);
61          sleep $time_diff;
62    }
63
64    for( $x =0; $x <=28800; $x +=300 ){
65          my $data = rand()*10;
66          my $timestamp = time();
67
68          open FILE, ">>log_8hs.txt" or die $!;
69          print FILE "$timestamp $data \n";
70          close FILE;
71
72          my $i = 0;
73          while ( $i <= $data) {
74                  get("http://roamer.dynalias.org/roamer.php");
75                  $i++;
76                  sleep 1;
77          }
78
79          my $diff = time();
80          my $time_diff = 300 - int($diff-$timestamp);
81          sleep $time_diff;
82    }
83 }
```

Appendix C

Self-management Functioning Scripts

C.1 Self-monitoring Function

```
<?php

  #showing in the web page, user IP, time stamp, country, and country code.
  $user_ip = $_SERVER['REMOTE_ADDR'];
  $timestamp = time();
  $printout = "User's IP is: $user_ip with the time stamp:$timestamp.<br>";
  echo $printout;

  $country = find_country($user_ip);
  $city = find_city($user_ip);
  echo "The user comes from: $country $city.<br>";

  $code = find_code($user_ip);
 echo "The location code is: $code.<br>";

  #writing in the log file, user IP, time stamp, and country code.
  $logfile = "/var/www/monitor/logfile/client_track.txt";
  $file = @fopen($logfile,'a') or die("can not open file");
  $log = $user_ip . " " . $timestamp . " " . $code . "\n";

  fwrite($file,$log);
  fclose($file);

  function find_country($ip_address){
        $no = preg_split("/\./",$ip_address);
        if($no[0] == "69"){
          return "USA";
        }

        else{
          return "Norway";
        }
  }

  function find_city($ip_address){
        $no = preg_split("/\./",$ip_address);
```

```
37          if($no[0] == "69"){
38            return "Wallingford";
39          }
40          else{
41            if(($no[0] == "128") && ($no[2] == "73")){
42                  return "Oslo";
43            }
44            else{
45                  return "Gjovik";
46            }
47          }
48    }
49
50    function find_code($ip_address){
51          $no = preg_split("/\./",$ip_address);
52          if($no[0] == "69"){
53            return "003";
54          }
55          else{
56            if(($no[0] == "128") && ($no[2] == "73")){
57                  return "001";
58            }
59            else{
60                  return "002";
61            }
62          }
63    }
64
65 ?>
```

C.2 Self-analyzing Function

C.2.1 Self-analyzing in Service Optimization

C.2.1.1 Analyze Highest Requests Percentage (user_percentage.pl)

```
1  #!/usr/bin/perl
2  use warnings;
3  use strict;
4  use File::ReadBackwards;
5
6  #define the different time period:
7  my $current_time = time();
8  my $start_time = $current_time - 900;
9
10 #read backwards of the log file:
11 my $log_file = "/var/www/logfile/client_track.txt";
12 my $analyze = File::ReadBackwards->new($log_file) or die "can't read file: $!\n";
13
14 my $hits_total = 0;
15 my $hits_001 = 0;
16 my $hits_002 = 0;
17 my $hits_003 = 0;
18 my $line;
19
20 #count the requests from each location:
21 while ( defined($line = $analyze->readline) )
22 {
```

```
23          my ($user_ip, $timestamp, $code) = split(" ",$line);
24
25          if($timestamp>= $start_time){
26                  $hits_total++;
27                  if($code=="001"){
28                          $hits_001++;
29                  }
30                  else{
31                          if($code=="002"){
32                                  $hits_002++;
33                          }
34                          else{
35                                  if($code=="003"){
36                                          $hits_003++;
37                                  }
38                          }
39                  }
40          }
41          else{last;}
42  }
43
44  my $oslo = $hits_001 / $hits_total;
45  my $gjovik = $hits_002 / $hits_total;
46  my $usa = $hits_003 / $hits_total;
47  my $percentage;
48
49  #compare the percentage from each location to find the highest one
50  if($oslo > $gjovik && $oslo > $usa){
51          $percentage = $oslo;
52  }
53  else{
54          if($gjovik > $oslo && $gjovik > $usa){
55                  $percentage = $gjovik;
56          }
57          else{
58                  if($usa > $oslo && $usa > $gjovik){
59                          $percentage = $usa;
60                  }
61          }
62  }
63
64  open FILE, ">>/var/www/analyze/user/logfile/percentage.txt" or die $!;
65  print FILE "$percentage\n";
66  close FILE;
67
68  print "$percentage\n";
```

C.2.1.2 Analyze Highest Requests Location (user_code.pl)

```
1   #!/usr/bin/perl
2   use warnings;
3   use strict;
4   use File::ReadBackwards;
5
6   #define the different time period:
7   my $current_time = time();
8   my $start_time = $current_time - 900;
9
10  #read backwards of the log file:
```

```
11 my $log_file = "/var/www/logfile/client_track.txt";
12 my $analyze = File::ReadBackwards->new($log_file) or die "can't read file: $!\n";
13
14 my $hits_total = 0;
15 my $hits_001 = 0;
16 my $hits_002 = 0;
17 my $hits_003 = 0;
18 my $line;
19
20
21 #count the requests from each location:
22 while ( defined($line = $analyze->readline) )
23 {
24         my ($user_ip, $timestamp, $code) = split(" ",$line);
25
26         if($timestamp>= $start_time){
27                 $hits_total++;
28                 if($code=="001"){
29                         $hits_001++;
30                 }
31                 else{
32                         if($code=="002"){
33                                 $hits_002++;
34                         }
35                         else{
36                                 if($code=="003"){
37                                         $hits_003++;
38                                 }
39                         }
40                 }
41         }
42         else{last;}
43 }
44
45 #get the request percentage of each location:
46 my $oslo = $hits_001 / $hits_total;
47 my $gjovik = $hits_002 / $hits_total;
48 my $usa = $hits_003 / $hits_total;
49 my $high_code;
50
51 #find the highest requests location:
52 if($oslo > $gjovik && $oslo > $usa){
53         $high_code = 001;
54 }
55 else{
56         if($gjovik > $oslo && $gjovik > $usa){
57                 $high_code = 002;
58         }
59         else{
60                 if($usa > $oslo && $usa > $gjovik){
61                 $high_code = 003;
62                 }
63         }
64 }
65
66 open FILE, ">>/var/www/analyze/user/logfile/code.txt" or die $!;
67 print FILE "$high_code\n";
68 close FILE;
69
70 print "$high_code\n";
```

C.2.1.3 Analyze Highest Requests Tendency (user_boolean.pl)

```
#!/usr/bin/perl
use warnings;
use strict;
use File::ReadBackwards;

#define the different time period:
my $current_time = time();
my $start_time = $current_time - 900;
my $tail_time = $current_time - 360;
my $head_time = $current_time - 720;

#read backwards of the log file:
my $log_file = "/var/www/logfile/client_track.txt";
my $analyze = File::ReadBackwards->new($log_file) or die "can't read file: $!\n";

#use hash to define the highest requests location code:
my %area_hash = ( "1" => "001", "2" => "002", "3" => "003" );
my $highest_code = $area_hash{$ARGV[0]};

my  $new_clients = 0;
my  $old_clients =0;
my  $line;

while ( defined($line = $analyze->readline) )
{
        my ($user_ip, $timestamp, $country) = split(" ",$line);

        #count the requests in different time periods:
        if($country eq $highest_code){
                if($timestamp>= $start_time){
                        if($timestamp>=$head_time && $timestamp<=$tail_time){
                                $old_clients++;
                        }
                        else{
                                if($timestamp>=$tail_time){
                                        $new_clients++;
                                }
                        }
                }
                else{last;}
        }
}

#define the requests trend
my $boolean;
if ($new_clients<$old_clients){
        $boolean = 0;
}
else{
        $boolean = 1;
}

open FILE, ">>/var/www/analyze/user/logfile/boolean.txt" or die $!;
print FILE "$boolean\n";
close FILE;

print "$boolean\n";
```

C.2.2 Self-analyzing in System Integrity

C.2.2.1 Analyze Clients Connectivity (storage_clients.pl)

```
#!/usr/bin/perl
use warnings;
use strict;
use File::ReadBackwards;

#define the different time period:
my $current_time = time();
my $start_time = $current_time - 900;


#read backwards of the log file:
my $log_file = "/var/www/logfile/client_track.txt";
my $analyze = File::ReadBackwards->new($log_file) or die "can't read file: $!\n";

my $total_clients = 0;
my $new_clients = 0;
my $old_clients = 0;

my $line;

#counte the total request during the last hour:
while ( defined($line = $analyze->readline) )
{
        my ($user_ip, $timestamp, $country) = split(" ",$line);

        if($timestamp>=$start_time){
                $total_clients++;
        }
        else{last;}
}

open FILE, ">>/var/www/analyze/storage/logfile/hits.txt" or die $!;
print FILE "$total_clients\n";
close FILE;

print "$total_clients\n";
```

C.2.2.2 Analyze Clients Tendency (storage_boolean.pl)

```
#!/usr/bin/perl
use warnings;
use strict;
use File::ReadBackwards;

#define the different time period:
my $current_time = time();
my $start_time = $current_time - 900;
my $tail_time = $current_time - 360;
my $head_time = $current_time - 720;

#read backwards of the log file:
my $log_file = "/var/www/logfile/client_track.txt";
my $analyze = File::ReadBackwards->new($log_file) or die "can't read file: $!\n";

```

```
16 my $new_clients = 0;
17 my $old_clients =0;
18 my $line;
19
20 while ( defined($line = $analyze->readline) )
21 {
22         my ($user_ip, $timestamp, $country) = split(" ",$line);
23
24         #count the requests in different time periods:
25         if($timestamp >= $start_time){
26                 if($timestamp >=$head_time && $timestamp <=$tail_time){
27                         $old_clients++;
28                 }
29                 else{
30                         if($timestamp >=$tail_time){
31                                 $new_clients++;
32                                 }
33                 }
34         }
35         else{last;}
36 }
37
38 #define the requests trend
39 my $boolean;
40 if ($new_clients <$old_clients){$boolean = 1;}
41 else{$boolean =0;}
42
43 open FILE, ">>/var/www/analyze/storage/logfile/boolean.txt" or die $!;
44 print FILE "$boolean\n";
45 close FILE;
46
47 print "$boolean\n";
```

C.2.2.3 Analyze Clients Location (storage_code.pl)

```
1 #!/usr/bin/perl
2 use warnings;
3 use strict;
4
5 my $current_code;
6 my $new_code;
7
8 #get current IP address:
9 my $ip_old = `/sbin/ifconfig eth0 | grep "inet addr:"`;
10 chomp $ip_old;
11
12 #get current subnet address:
13 my $subnet_old = $ip_old;
14 $subnet_old =~ s/^.*inet addr:(\S+)\.(\S+)\.(\S+)\.(\S+) .*/$1\.$2\.$3/g;
15
16 my ($n1, $n2, $n3) = split(/\./,$subnet_old);
17
18 #find the current location:
19 if($n1=="69"){
20         $current_code=003;
21 }
22 else{
23         if($n1=="128" && $n2=="39" && $n3=="73"){
24                 $current_code=001;
```

```
25          }
26      else{
27          $current_code=002;
28          }
29  }
30
31  #define the new location to migrate:
32  if($current_code=="003"){
33          $new_code=001;
34  }
35  else{
36          if($current_code=="001"){
37                  $new_code=002;
38          }
39          else{$new_code=001;}
40  }
41
42  open FILE, ">>/var/www/analyze/storage/logfile/code.txt" or die $!;
43  print FILE "$new_code\n";
44  close FILE;
45
46  print "$new_code\n";
```

C.2.2.4 Analyze Log File Growth (storage_logfile.pl)

```
1   #!/usr/bin/perl
2   use warnings;
3   use strict;
4   use File::ReadBackwards;
5
6   #read backwards of the log file:
7   my $time_log = "/var/www/action/logfile/executed_time.txt";
8   my $start_file = File::ReadBackwards->new($time_log) or die "can't read file: $!\n";
9
10  #find when was the action executed last time
11  my $start_time = $start_file->readline;
12
13  #read backwards of the log file:
14  my $log_file = "/var/www/logfile/client_track.txt";
15  my $analyze = File::ReadBackwards->new($log_file) or die "can't read file: $!\n";
16
17  my $total_hits = 0;
18
19  my $line;
20
21  #count the changes of the logfile since last backup
22  while ( defined($line = $analyze->readline) )
23  {
24          my ($user_ip, $timestamp, $country) = split(" ",$line);
25
26          if($timestamp >= $start_time){
27                  $total_hits++;
28          }
29          else{last;}
30  }
31
32  open FILE, ">>/var/www/analyze/storage/logfile/logfile.txt" or die $!;
33  print FILE "$total_hits\n";
34  close FILE;
```

```
35
36 print  "$total_hits\n";
```

C.3 Decision-making Function (Cfengine Configuration File)

C.3.1 Decision-making in Service Optimization

```
1  #!/usr/sbin/cfagent -K -f
2
3  # this is where we gather information and make policy
4  control:
5    actionsequence = (  editfiles shellcommands )
6
7    synchronizetime = ( ExecShellResult("/var/www/analyze/sychronize_time.pl") )
8    userpercentage = ( ExecShellResult("/var/www/analyze/user_percentage.pl") )
9    usercode = ( ExecShellResult("/var/www/analyze/user_code.pl") )
10
11   migrationthreshold = ( 0.6 )
12
13 # this is where we make decisions
14 classes:
15   any::
16         migrationThresholdReached =
17                 ( IsGreaterThan(${userpercentage},${migrationthreshold}) )
18         userboolean =
19                 ( ReturnsZero("/var/www/analyze/user_boolean.pl ${usercode}") )
20
21 # this is where we do actions
22 shellcommands:
23   migrationThresholdReached.userboolean::
24         # execute actionscript for migration
25         "/var/www/action/migration.pl ${usercode} "
26         owner=root group=root
27
28 # this is where we keep log files
29 editfiles:
30   { /tmp/output.txt
31         BeginGroupIfNoSuchLine "nonexistentline"
32          Append "user percentage is ${userpercentage}"
33          Append "usercode is ${usercode}"
34          Append "migrationtreshold ${migrationtreshold}"
35         EndGroup
36
37         BeginGroupIfDefined "migrationThresholdReached"
38          Append "We will migrate to ${usercode}"
39         EndGroup
40   }
41
42 alerts:
43   userboolean::
44         "Userboolean is defined as 0"
45   migrationThresholdReached::
46         "Migration treshold is reached"
47   migrationThresholdReached.userboolean::
48         "Migrationthreshold ${migrationtreshold} reached, migrating to ${usercode}"
```

C.3.2 Self-analyzing in System Integrity

```
#!/usr/sbin/cfagent -K -f

# this is where we gather information and make policy
control:
  actionsequence = ( editfiles shellcommands )

  synchronizetime = ( ExecShellResult("/var/www/analyze/sychronize_time.pl") )

  storagehits  = ( ExecShellResult("/var/www/analyze/storage_hits.pl") )
  storagecode = ( ExecShellResult("/var/www/analyze/storage_code.pl") )
  storagefile = ( ExecShellResult("/var/www/analyze/storage_logfile.pl") )

  migrationtreshold = ( 180 )
  logfileincreasitivity = ( 500 )

# this is where we make decisions
classes:
  any::
        migrationThresholdReached =
                ( IsGreaterThan(${migrationtreshold},${storagehits}) )
        logfileincreasitivityReached =
                ( IsGreaterThan(${storagefile},${logfileincreasitivity}) )
        storageboolean =
                ( ReturnsZero("/var/www/analyze/storage_boolean.pl") )

# this is where we do actions
shellcommands:
  # execute actionscript for migration
  migrationThresholdReached.logfileincreasitivityReached.storageboolean::
        "/var/www/action/migration.pl ${storagecode}"
        owner=root group=root
        "/var/www/action/backup.pl"
        owner=root group=root

# this is where we keep log files
editfiles:
  { /tmp/output.txt
        BeginGroupIfNoSuchLine "nonexistentline"
         Append "Storagefile has increased ${storagefile}"
         Append "Storagehits is ${storagehits}"
         Append "Storagecode is ${storagecode}"
         Append "Migrationtreshold ${migrationtreshold}"
         Append "Logfileincreasitivity ${logfileincreasitivity}"
        EndGroup

        BeginGroupIfDefined "migrationThresholdReached"
         Append "We will migrate to ${storagecode}"
        EndGroup
  }

alerts:
  storageboolean::
    "Storageboolean is true"
  migrationThresholdReached::
    "Migrationtreshold ${migrationtreshold} is not reached."
  logfileincreasitivityReached::
    "Log file ${storagefile} is longer than the policy ${logfileincreasitivity}."
  migrationThresholdReached.logfileincreasitivityReached.storageboolean::
    "Everything is set to execute the storage action, migrating to ${storagecode}"
```

C.4 Execute Actions

C.5 Virtual Machine Migration

```
#!/usr/bin/perl
use warnings;
use strict;

my $timestamp = time();

# get the current ip address and netmask
my $ip_old = `/sbin/ifconfig eth0 | grep "inet addr:"`;
chomp $ip_old;
my $subnet_old = $ip_old;
$subnet_old =~ s/^.*inet addr:(\S+)\.(\S+)\.(\S+)\.(\S+) .*/$1\.$2\.$3/g;

# find the current location:
my ($n1, $n2, $n3) = split(/\./,$subnet_old);
my $current_code;
if($n1=="69"){
        $current_code = 3;
}
else{
        if($n1=="128" && $n2=="39" && $n3=="73"){
                $current_code = 1;
        }
    else{
        $current_code = 2;
        }
}

my %area_hash = ( "1" => "/root/migrate_to_oslo.mln",
                "2" => "/root/migrate_to_gjovik.mln",
                "3" => "/root/migrate_to_conneticut.mln");
my $file = $area_hash{$ARGV[0]};

if($current_code == $ARGV[0]) {
        print "The virtual machine is located where it should be.\n";
        exit 0;
}

system ("/root/mln client -h huldra.vlab.iu.hio.no -f $file -c upgrade");
sleep 20;

system ("killall dhclient");

my $bool = 1;
while ( $bool ){
        system ("dhclient");
        my $ip_new = `/sbin/ifconfig eth0 | grep "inet addr:"`;
        chomp($ip_new);

        my $subnet_new = $ip_new;
        $subnet_new =~ s/^.*inet addr:(\S+)\.(\S+)\.(\S+)\.(\S+) .*/$1\.$2\.$3/g;

        last if( $subnet_new ne $subnet_old);

        sleep 10;
}

```

```
57 open FILE, ">>current_location.txt" or warn "$!";
58 print FILE "$timestamp $ip_old\n";
59 close FILE;
60
61 system("ddclient --force");
```

C.6 Virtual Machine Backup

```
1  #!/usr/bin/perl
2  use warnings;
3  use strict;
4
5  #get the IP address
6  my $ip_old = `/sbin/ifconfig eth0 | grep "inet addr:"`;
7  chomp $ip_old;
8
9  #get the net mask:
10 my $subnet_old = $ip_old;
11 $subnet_old =~ s/^.*inet addr:(\S+)\.(\S+)\.(\S+)\.(\S+) .*/$1\.$2\.$3/g;
12
13 my ($n1, $n2, $n3) = split(/\./,$subnet_old);
14 my $old_code;
15
16 #find the server's current location:
17 if($n1 == "69"){
18         $old_code = "003";
19 }
20 else{
21         if($n1 == "128" && $n2 == "39" && $n3 == "73"){
22                 $old_code = "001";
23         }
24         else{
25                 $old_code = "002";
26         }
27 }
28
29 #define the project file should be used:
30 my %old_hash = ( "001" => "/root/backup_in_oslo.mln",
31                  "002" => "/root/backup_in_gjovik.mln",
32                  "003" => "/root/backup_in_conneticut.mln" );
33 my $old_disk = $old_hash{$old_code};
34
35 #execute the local backup action
36 system("/root/mln client -h huldra.vlab.iu.hio.no
37                  -f $old_disk -c upgrade -s 5 -l lux1");
38 sleep 3;
39
40 while( not system("mount /dev/hdc /mnt") == 0 ){
41         sleep 5;
42 }
43
44 #copy the files that want to be baked up
45 system("cp -vr /var/www /mnt/www");
46 system("cp -vr /root /mnt/mln");
47
48 #umount the hard drive
49 system("umount /mnt");
50
```

```
51 #find the project file should be used:
52 my %oldcode_hash = ( "001" => "/root/migrate_to_oslo.mln",
53                  "002" => "/root/migrate_to_gjovik.mln",
54                  "003" => "/root/migrate_to_conneticut.mln" );
55 my $old_location = $oldcode_hash{$old_code};
56
57 #detach the local hard drive from the the virtual machine
58 system("/root/mln client -h huldra.vlab.iu.hio.no
59                  -f $old_location -c upgrade -s 5 -l lux1");
60
61 my $time_stamp = time();
62
63 open FILE, ">>/var/www/action/executed_time.txt" or die $!;
64 print FILE "$time_stamp \n";
65 close FILE;
```

Appendix D

Measurement and Analysis

D.1 Filter the Web Service Log File

```
1  #!/usr/bin/perl
2  use warnings;
3  use strict;
4  use File::ReadBackwards;
5
6  #define the different time period:
7  my $now = time();
8  my ($sec,$min,$hour,$mday,$mon,$year,$wday,$yday,$isdst) = localtime($now);
9  my $diff = $hour * 60 * 60 + $min * 60 + $sec;
10 my $start_time = $now - $diff;
11 my $end_time = $start_time - 3600*24*2;
12
13 #read backwards of the log file:
14 my $log_file = "/var/www/monitor/logfile/client_track.txt";
15 my $analyze = File::ReadBackwards->new($log_file) or die "can't read file: $!\n";
16
17 my %hash = ();
18 my $line;
19
20 #use hash to counte the total requests per 5 minutes
21 while ( defined($line = $analyze->readline) )
22 {
23         my ($user_ip, $timestamp, $country) = split(" ",$line);
24
25         if($timestamp !~/^\d+$/){next;}
26
27         if($end_time <= $timestamp && $timestamp <= $start_time){
28                 my $chunk_no = int(($timestamp - $end_time)/300);
29                 if($hash{ $chunk_no }){
30                         $hash{ $chunk_no }++;
31                 }
32                 else{
33                         $hash{ $chunk_no } = 1;
34                 }
35         }
36
37         else{
38                 if($timestamp < $end_time){
39                         last;
40                 }
```

```
41          }
42  }
43
44  open FILE, ">total_hits.txt" or die $!;
45
46  #sort the data in hash, and print them to a logfile
47  for my $key(sort {$a <=> $b} keys %hash){
48          my $value = $hash{$key};
49          my $time_stamp = $key * 300 + $end_time;
50          print FILE "$time_stamp $value\n";
51  }
52
53  close FILE;
```

D.2 Using Round Trip Time to Measure Virtual Machine Migration

```
1   #!/usr/bin/perl
2   use warnings;
3   use strict;
4
5   for(;;){
6     my $timestamp_start = time();
7
8     my $ping = `ping -w 3 -f -c 3 roamer.dynalias.org | grep 'rtt'`;
9     chomp $ping;
10
11    my $time = 0;
12    my $mdev = 0;
13
14    if ($ping){
15          $ping =~ /\d+\.\d+\/(\d+\.\d+)\/\d+\.\d+\/(\d+\.\d+) ms/;
16          $time = $1;
17          $mdev = $2;
18    }
19
20    open FILE, ">>rtt_log.txt" or die $!;
21    print FILE "$timestamp_start  $time  $mdev\n";
22    close FILE;
23
24    my $timestamp_end = time();
25    my $diff = 10 - ($timestamp_end - $timestamp_start);
26    sleep $diff;
27  }
```

D.3 Synchronize the Time Stamp in Log File

```
1   #!/usr/bin/perl
2   use warnings;
3   use strict;
4
5   #get the current IP address
6   my $ip = `/sbin/ifconfig eth0 | grep "inet addr:"`;
7   chomp $ip;
```

```
8
9  #get the current net mask
10 my $subnet = $ip;
11 $subnet =~ s/^.*inet addr:(\S+)\.(\S+)\.(\S+)\.(\S+) .*/$1\.$2\.$3/g;
12
13 #find the current location:
14 my ($n1, $n2, $n3) = split(/\./,$subnet);
15 my $current_code;
16 if($n1=="69"){
17         $current_code = 3;
18 }
19 else{
20         if($n1=="128" && $n2=="39" && $n3=="73"){
21                 $current_code = 1;
22         }
23     else{
24         $current_code = 2;
25         }
26 }
27
28 if ($current_code == 2){
29         my $line;
30         my $log_file = "/var/www/logfile/client_track.txt";
31
32         open FILE, ">/var/www/logfile/client_track.txt" or die $!;
33         open (CHECKBOOK, $log_file) || die "couldn't open the file!";
34
35         while ( $line = <CHECKBOOK>){
36                 my ($ip_address, $timestamp, $data) = split(" ",$line);
37                 if($timestamp < 1210000000){
38                         $timestamp+=2756849;
39         }
40                 print FILE "$ip_address $timestamp $data\n";
41         }
42                 close FILE;
43 }
44
45 else{
46         my $line;
47         my $log_file = "/var/www/logfile/client_track.txt";
48
49         open FILE, ">/var/www/logfile/client_track.txt" or die $!;
50         open (CHECKBOOK, $log_file) || die "couldn't open the file!";
51         while ( $line = <CHECKBOOK>){
52         my ($ip_address, $timestamp, $data) = split(" ",$line);
53                 if($timestamp > 1210000000){
54                         $timestamp-=2756849;
55                 }
56                 print FILE "$ip_address $timestamp $data\n";
57         }
58         close FILE;
59 }
```