UNIVERSITY OF OSLO
Department of Informatics

Virtual Machine Initiated Operations Logic for Resource Management

Oslo University College

Master thesis

Nii Apleh Lartey

May 18, 2009

Virtual Machine Initiated Operations Logic for Resource Management

Nii Apleh Lartey

2009-05-18

Abstract

This thesis takes a look at an alternative approach to resource management using virtualization. The paper makes an attempt at looking at some administrative issues that arise due to current designs of virtualization management especially in multi-tenant virtual hosting environments and possibility of taking advantage of different kinds of offers from different providers through automated operations logic driven management.

The alternative approach investigated is to hand decision making of operations logic over to the virtual machine as opposed to the virtual infrastructure itself. Being able to do this would mean that management policy can be maintained when transition between different sites is made.

Acknowledgements

My profound gratitude goes to my supervisor, Kyrre Matthias Begnum for his support, patience, firmness and ingenuity in shaping and guiding the outcome of this paper. I am grateful for the time taken to go through design issues and technical challenges with me.

I am Most grateful unto God, for the gift of life. I would like to thank my parents, Mr and Mrs Edna Lartey for their love, support and preparation through nurturing given to me over the years. I appreciate also, Ms Jumoke Lafenwa, for your love, affection and encouragement through this period.

My appreciation goes to all my lecturers as well to my classmates, for different kinds of help offered me through this period and entire study program.

Contents

List of Figures

List of Tables

Listings

Chapter 1

Introduction

1.1 Motivation

Large computer installations and data centers around the world have found an increasing need to manage resources more prudently in order to keep their cost of operation low. The role of the system administrator today, has gone beyond keeping systems up and maintaining state, to include this goal of efficient use of resources and cost saving. Saving power and an ability to dynamically scale up or down the architecture and function of computer systems are very desirable features in this pursuit.

The path towards more efficient usage of resources has lead to an increased popularity and increase in patronage of virtualization technologies. Virtualization, loosely explained, refers to the implementation of computer hardware resources through software. It allows for computer installations to be expanded with regards to function and design without necessarily acquiring more hardware. Operators and managers of datacenters and computer installations in various organizations have found the usefulness of virtualization in being able to consolidate several computer instances (by creating virtual machines) into fewer physical systems. Consolidation allows the system administrator to stick to the traditional practice of running each service as a separate installation even with fewer physical machines.

A common approach of virtualization technologies towards an optimal resource usage is to have a virtual infrastructure that has a virtual machine management software capable of making changes to the residing virtual resources dynamically in response to predetermined factors which can usually be altered within certain limits to suite one's environment and need. Implementation of operations logic [1] in virtual machines today, is conventionally dependent on the intelligence of the virtualization management software of the virtual infrastructure on which they run.

The idea of having all the decision-making and implementation of operations logic reside with the virtual infrastructure may not be desirable in all cases. There are issues of administration coupled with this paradigm; this is perhaps more apparent in a multi-tenant virtual hosting facility where owners may want their virtual machines to be more autonomous in this regard . Theses administrative challenges are perhaps more important when there are clients who are tenants to multiple multi-tenant virtual hosting providers.

1.2 Problem Statement

What are the effects of shifting some operations logic responsibilities to the virtual machines as opposed to the conventional case of total responsibility lying with the virtualization management software?

Effects: Effects as used in the problem statement refers to what changes in terms of ease of management, dependability and predictability would be caused. In addition to changes, the effects include any benefits would be gained or lost with this alternative approach.

Shifting responsibilities:Shifting the monitoring and decision-making responsibility from the infrastructure and virtual machine monitor to the individual virtual machines. The responsibility of altering configuration and setup such as migration of virtual machines still lies with the infrastructure however. The virtual machine would assume the responsibility of processing monitored information and requesting the virtual machine monitor or virtual machine management software to implement those changes.

Operations logic: Operations logic refers to programmable behaviour of individual virtual machines based on pre-defined and independent policies. This gives the idea and enablement of programing the behaviour of the system rather than maintaining state.

Conventional case: In the problem statement, the most common approach, conventional case refers to what is taken to be the most common approach of dynamically managing virtual resources, which is having the operations logic programmed into and implemented by the virtualization software.

Virtual machine: Virtual machine, as used here is software implementation of a computer, which is capable of self-management.

Virtualization management software: This refers to the software that manages the virtual machines and controls the hypervisor. It is the interface to which the system administrator is able to make changes to the configuration of the virtual infrastructure.

Having an architecture as proposed in this document would enable one to get a step closer to having a policy-based management. Policy-based management offers the possibility to reconfigure systems for the desired quality of service [2].

1.3 Approach

Quite a few possible alternatives to approaching to this problem exist. Among these alternatives is to implement an experimental setup solely by modeling and simulations. Using simulation tools would cost less in terms of setup time and resources as compared to some other approaches. It would also be more easily reproducible, especially to researchers and those in academia. Simulation tools would also simplify the experimental setup because one would have more control over values and parameters that affect the results and thus there would be more control over the total experiment. This could mean one may be able to perform more accurate and in-depth analysis on the results obtained. Performing this research through simulations and modeling however, is not the first choice because there would be some obstacles which can be avoided otherwise. The use of simulation and its success may not necessarily mean a successful implementation since simulation would be based on a model which would have to be developed without much information. In addition, the process and effects may be more difficult to communicate to the community of system administrators to which this information would perhaps be more beneficial. A later project that uses simulation could be undertaken in the future because there would be more information on the expected behaviour of such a system.

Yet another alternative is by literature review. The challenge that arises with this method is the lack of literature and study on the topic. Any conclusions that could be arrived at would be pretty much based on speculations and allegations.

In order to get a very realistic environment, especially with regards to user traffic, it would have been perhaps more appropriate to use a production environment to investigate the consequences. Such an environment would have probably been the best but then there is an increased risk to the production systems. If things were to go wrong, which is a possibility in an experiment of this nature, the service or services been offered by this environment could fail or be negatively affected.

Among the many alternatives available, however, the approach chosen to research into this idea is to perform an experiment by implementing possible sample policies in different scenarios and using that as a case study to obtain results and discuss the implication. An extensive study of current developments in the field of virtualization and power saving technologies will be made, especially those that use the approach being labeled as conventional here, that is, having an infrastructure that does all the monitoring and decision making for the virtual setup.

A set of basic operations logics would be designed and implemented. Behavioural policies would be defined that would describe how the setup is to evolve in response to differing factors. In building the experimental setup, it would be necessary to have scripts that monitor several parameters from within the virtual machines. The parameters measured are those relating to services offered by the virtual machine; they would therefore differ depending on what one is interested in and the kind of service running and should reflect the performance of the service running. These parameters would have thresholds that trigger several actions in accordance to their behavioural policy. Traffic to these services would be simulated in order to trigger behavioural reactions.

1.4 Thesis Outline

This paper would be outlined in the following manner: The introduction gives a brief overview of the topic and makes the problem statement. Chapter 2 gives more information and background knowledge on virtualization and related subjects and terminologies used in this paper. Chapter 3 contains the methodology, design and implementation of concept. Chapter 4 is the results section and contains the results obtained from the implementation setup. In Chapter 5, the results presented in Chapter 4 are discussed. Chapter 6 is the concluding chapter which talks about the findings of this thesis in relation to questions raised and the motivation for this topic. Possible future work and development of this concept is also addressed.

Chapter 2

Background

This chapter gives a brief introduction on several technologies and terminologies around virtualization as well as concepts and application software which are relevant to the proof of concept which would follow in later chapters.

Traditionally, system administrators have been concerned with maintaining the state of running machines and making sure they do not go down. With the primary aim of keeping systems up, system administrators have introduced methods and terms such as over-provisioning [3] and the like which are usually rather expensive to maintain over time. In today's world, system administration has come to mean more than just maintaining state and keeping systems up. Computer systems support businesses and datacenters are run as a business as well, as such, there is a factor of economics in the big picture. It has become more desirable and even necessary to have systems evolve in response to predetermined factors, thus the system administrator has become more concerned with programming behavior rather than maintaining states.

2.1 Operations logic

Operations are tasks performed in datacenters by human operators as a way to run and manage systems. Operations logic is thus the software implementation of operations tasks [1]. The aim of operations logic is to automate operations tasks. Being able to correctly implement operations logic would result in an improvement in recovery rates and performance by reducing human involvement in the process. Operations tasks involve protocols that are laid down to be performed under certain conditions. Common operations tasks identified by [4]include:

- Provisioning: how many instances of a particular type of server and where
- Deployment: installation of disk images, and instantiation of machines
- Interconnection: creation of networks
- Monitoring: measurement and observation of specific properties for failure or overload
- Evolution: response to events and versioning

2.2 Virtualization

Virtualization was introduced by International Business Machine (IBM) in the 1960s with the aim of partitioning mainframe hardware to allow better utilization of hardware and help their clients leverage their investments made. Partitioning hardware allowed IBM's mainframes to run multiple applications and processes simultaneously (multitasking). Over time, the inexpensive X86 architecture became more popular and accessible and also grew in it's processing power causing it to be used as a platform by businesses to run their services on. The desire and need to virtualize in the X86 was not compelling to begin with. Over time, the increase in the processing power and cost in implementation of large networked architectures and applications coupled with inefficient use of resources, especially computer processing unit (CPU) utilization of the X86 once again led to the desire to virtualize to obtain a similar leverage that was made possible in the mainframes which had been virtualized in the 1960s. Attempts to virtualize the X86 architecture began.

Operating systems intended to run on X86 CPUs are designed to run directly on physical hosts. Their architecture and mode of operation assumes total control of the hardware resources therefore their performance is optimal when they run on bare-metal. The X86 architecture consists of four levels of privileges; Ring 0, Ring 1, Ring 2 and Ring 3. These levels are made available to the operating system and application software to enable management of hardware resources. Ring 0 is the most privileged level and the level at which the operating system functions. Introducing a virtualization layer meant that the operating system had to function at a less privileged level that it was intended to.

In 1998 a great leap in the history of virtualization was made; the X86 platform was virtualized. The solution was a combination of binary translation and direct execution on the processor that allowed multiple operating systems (guest operating systems) to run in full isolation on the same physical computer. Today, manufacturers of CPU, provide virtualization support, often called hardware-assisted virtualization on at least some of the processors they manufacture. Hardware-assisted virtualization adds additional capabilities to enable the virtualization software to better virtualize the CPU.

In this paper, the term virtualization would refer specifically to server or system virtualization which is a method of running multiple independent virtual operating systems on a single physical computer. Each instance of operating system together with its virtual hardware can run as though it were a physical machine. A virtual machine is thus, a collection of virtual hardware that supports the installation of a supported guest operating system. The virtualization of systems is made possible by virtualization of computer processing unit (CPU), memory, device and input-output (I/O).

2.2.1 Types of server virtualization

There are generally three types of server virtualization. These are operating system virtualization, hardware virtualization and para-virtualization. The type chosen for use may vary depending on preference, requirements and even a budget.

2.3 Virtualization Architecture

Virtualization is made possible through software. The virtualization software is responsible for partitioning the physical resources available and making those resources accessible to the running instances of operating systems (virtual machines) [5]. The virtualization software is often referred to as the hypervisor. In setting up a virtual machine, different resources, available to the hypervisor, are allocated. The typical things that are allocated are disk, CPU, Memory (RAM), network (network interface card). It is to be noted that typically only the number of CPUs are specified and thus the CPU specifications of the host would greatly affect the performance of the virtual machine; meaning two similar virtual machines with similar specifications on different hosts of differing CPU speeds would perform differently.

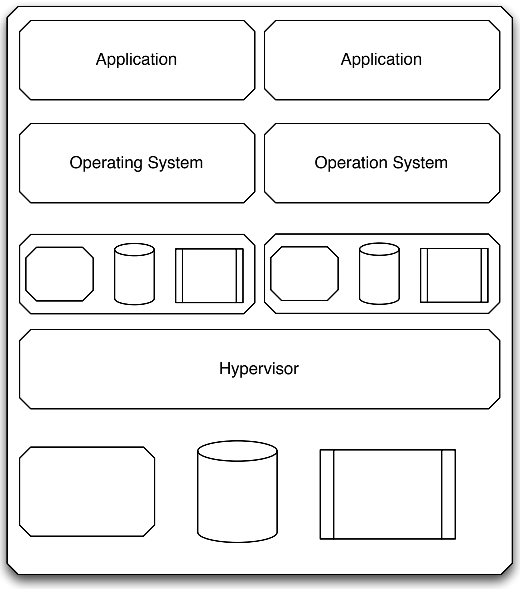


Figure 2.1: General virtualization architecture

Operating system virtualization

Operating system virtualization runs on top of an existing host operating system. It is sometimes referred to as container virtualization. A virtualization application, running on the host provides a set of libraries that gives applications the illusion that they are running on dedicated operating systems. The application can only interact with those applications running within its own virtual operating system. This type of virtualization is advantageous for implementing similar operating system functionalities on a single host machine. There are some major drawbacks however, in the choice of guest operating systems to run. Typically, one accesses the same functions as that of the host and so one is limited to that operating system.

Full hardware virtualization

Hardware virtualization makes use of a hypervisor which abstracts the hardware layer of the host on which guest operating systems are running. The virtual hardware environment is often referred to as virtual machine monitor (VMM). The virtual machine monitor offers a consistent environment irrespective of the host on which it resides, thus a possibility of migration is gained. Another advantage of this type of virtualization is that, different kinds of guest operating systems can be supported. A diagrammatic representation of full hardware virtualization is shown in figure 2.2

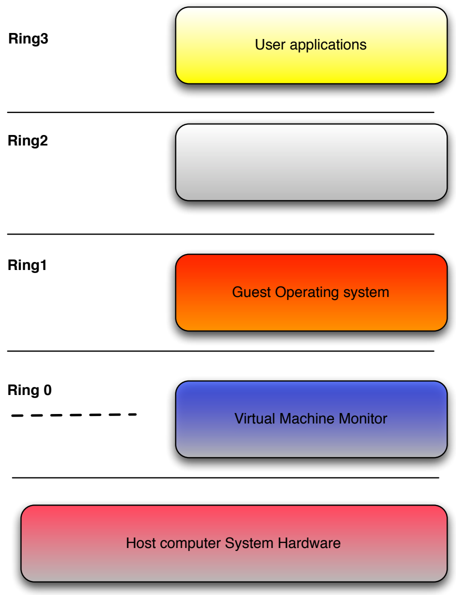


Figure 2.2: Full hardware virtualization

The presentation of a consistent environment means that translation between the virtual resources and system resources must be made by the hypervisor, this can be an issue for making use of devices for which the hypervisor has no device drivers. Hardware virtualization has a low performance because of the layers of translation done by the hypervisor.

Para-virtualization

In this type of virtualization the hardware is not abstracted but access to the underlying hardware resources by the guest operating systems is controlled by the virtualization software. In the implementation of para-virtualization, the guest operating system is aware of its state as a virtual machine. The operating system is thus modified to take advantage of features available; there is therefore a lower virtualization overhead.

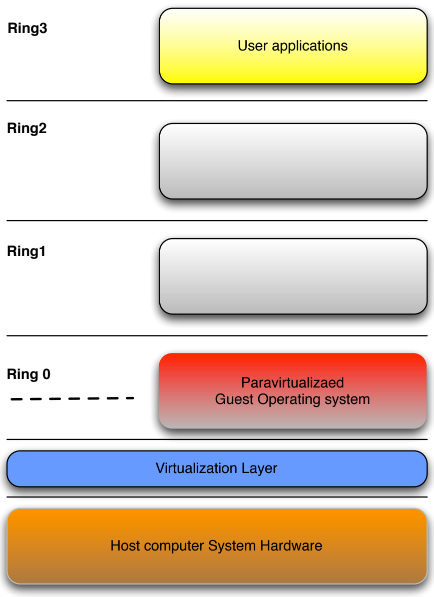


Figure 2.3: Paravirtualization architecture

2.3.1 Hardware-assisted virtualization

Hardware-assisted virtualization is a rather recent form of virtualization that stems from the efforts of hardware manufacturers to improve performance of hardware running virtualization software. This technique of virtualization depends on special feature provided by hardware manufacturers that allows the virtual machine monitor to run in privileged below Ring 0. This type of virtualization is still not very developed and rather costly because of the specialized hardware involved. It is however faster in performance than the others mentioned. A diagrammatic representation of this is shown in figure 2.4

Different hardware manufacturers implement virtualization support differently and this leads to interoperability issues when transferring virtual machines from a host with CPUs from one manufacturer to another host with CPUs from a different manufacturer.

2.3.2 Benefits and Trade-offs of virtualization

Among the many benefits of virtualization, perhaps, two stand out to make this technology really popular; cost saving and convenience. The idea of being able to run more servers than the number of physical machines that on has is an attractive one to stakeholders of businesses. Savings are made by buying less machines than would otherwise have been purchased. For the stakeholder, this means one needs less space that would otherwise have been the case and less cooling as well which is one of the major costs of datacenters. In a typical datacenter, machines run at about 15% utilization some 90% of the time. An underutilized machine takes up the same space and although less electricity, depending on the power saving features of the machine, increasing the load on a single machine is still less costly that powering up a new one. Less power usage means a greener environment as well. Being able to virtualize saves space, in turn saves money.

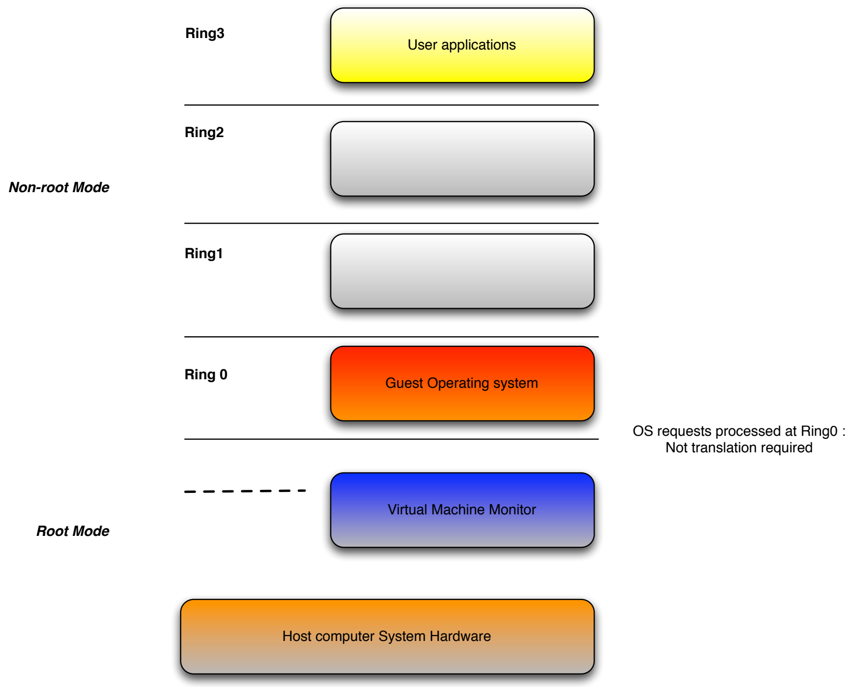


Figure 2.4: Hardware-assisted virtualization

In some cases, datacenters just do not have enough space for the expansion they desire and extending a datacenter is both costly and time consuming. In addition to infrastructural and power costs, virtualization allows saving on the cost of technical staffing. Less physical machines means less people are needed to maintain them.

For the system administrator, the thought that whole servers are regarded as a group of files allows for possibilities and portability which hitherto, were not possible. It means that cloning of an entire system could be done just by duplicating the files that represent it; these are usually some configuration and description files and a harddisk file. Virtualization decouples a running virtual machine from the underlying hardware by abstraction. This makes system upgrades very convenient and simple to do and even eliminates that in some cases. As an example, one can simply clone or migrate a virtual machine from an old system to a newer one without the need to update hardware dependent software and drivers.

Despite the many benefits that virtualization brings, there still are some down sides. Having less physical machines handle more load means that the machines and supporting hardware become more important. Points of failure become more important and critical such that the failure of on machine for example could cause multiple machines to go down which otherwise would not have been so. This is more so in the case of hardware-assisted virtualization. More points of failures are introduced as more machines depend on less parts to operate. When virtualization is done, a compromise is made on the performance of the physical machine. Despite claims of bare metal performance by some vendors with regards to their products, it is obvious that having more layers, however thin in-between software applications and the underlying resources would mean more delay in access those resources, thus a slower performance is expected. The hypervisor, however is designed to be a very thin layer and this extra delay and reduction in performance is made very small and thus insignificant for many cases.

2.3.3 Popular Virtualization Options

There are many server virtualization products and vendors available for use toady. Perhaps the two most widely used are those by VMware and Xen.

VMware

VMware is a company that was founded in 1998 to develop software for virtualization products. VMware makes different kinds of products for virtualization targeted to different kinds of people and for different uses. Among the virtualization products of VMware are VMware server and VMware ESX. Different virtualization technologies and combinations of these are used in different product offerings.

Xen

Xen is an open-source supported virtualization software originally created by the university of Cambridge Computer Laboratory. Xen serves as the backbone for many other virtualization products such as Citrix xenserver enterprise. It is included as a package in most major distributions of linux. Xen works in para-virtualization mode. The Xen hypervisor is responsible for scheduling CPU access to the residing guest operating systems. There first instance of guests is called “domain 0” (usually called dom0). The dom0 has direct access to the physical resources and serves as an interface for management.

2.4 Multi-tenant hosting environments

A multi-tenant hosting environment is one in which multiple independent clients are served [6]. The clients are served based on contractual agreements and they share certain resources that are owned by the hosting provider. This type of service is increasingly common especially for medium-sized companies that need affordable information technology services. Having multiple tenants means that the policies applied to these different services or systems are independent of each other and most likely varied. Administering policies from the hypervisor level is complex. Most virtualization management software are not designed for multiple administrators and as such granting administrative rights to theses clients without compromising on the integrity of the system in difficult thus most multi-tenant virtual hosting providers end up not granting their clients this access; thus they miss out on some of the advantages that come with virtualization and have to manage the virtual machines as if they were physical machines.

Sharing resources means that there is competition for resources among all these services. It is to the advantage of both the client and the provider that these resources be well managed. For the client, controlling the resources that are used and being charged for that is in the best interest of cost savings and thus more desirable.

It is not uncommon to find several business and institutions that have multiple multi-tenant hosting providers today. There are many reasons for this phenomenon, different price offerings and packages perhaps being the most important. This situation rises for temporary sessions during transition periods from one hosting provider to another as well. Figure 2.5 shows this sort of situation in a pictorial form.

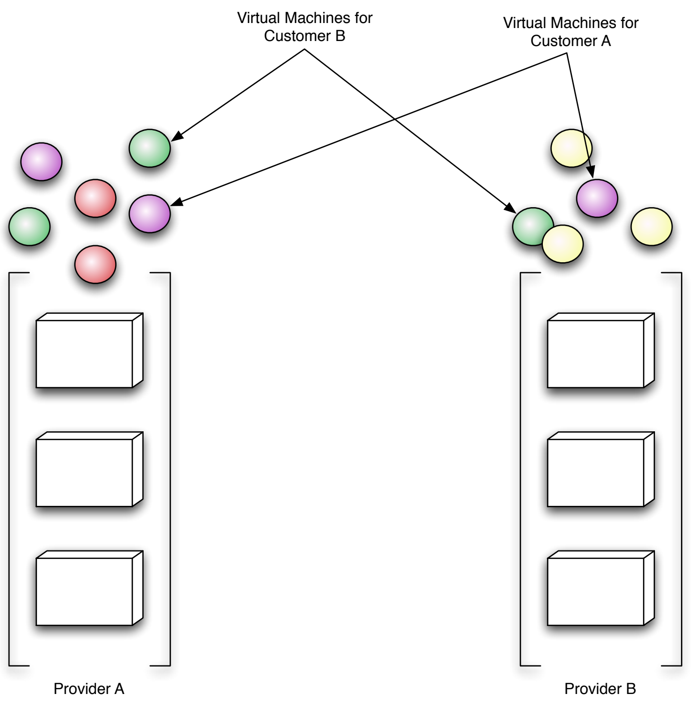

Virtual Machines for Customer B
Virtual Machines for Customer A
Provider A
Provider B

Figure 2.5: multi-tenant environments

2.5 Shared Storage Solutions

It is common to have a cluster of hosts composing the physical structure for virtualization. A cluster is desired for obvious reasons of redundancy and increased resource availability. To make use of many features that are available to clusters of hosts, it is common to have shared storage solutions for these clusters. Many different kinds of shared storage solutions exist but this document would not discuss them in any detail. It is worth mentioning that the type of of storage solution as well as it's configuration (RAID etc.) affects performance of the virtual machines

2.6 Cloud computing

Heading in the direction of rendering information technology as a service, many trends and approaches to technical challenges have cropped up. Among such trends in cloud computing. Cloud services are in their simplest form, IT services that are offered through the internet. They are available in different forms and implementation. Neal Leavitt [7] identifies four major types of cloud services; internet-based services, infrastructure as a service(IaaS), platform as a service(PaaS) and software as a service(SaaS). In this paper, the point of interest is IaaS (Infrastructure as a service). This is basically the delivery of a full computer over the internet. There has been heavy investments into infrastructure and the push for cloud computing by many large companies. To the system administrator, cloud computing is not very different from any other multi-tenant hosting environment. Different cloud computing providers just offer different ways of setting up computers, services and different pricing matrices. It is the different kinds of options and pricing that makes it attractive for business.

There are some advantages to cloud computing which makes it a appealing platform. According to [8], an experiment conducted in which a specific software application (montage) and the pegasus WS, a cloud computing platform, was used, a workflow with short runtimes provide good computer time performance but sometimes did suffer from resource scheduling delays and wide-area communication. There are some concerns especially on the security of cloud computing. In addition to the security concerns of the virtual infrastructure or cloud itself, access to the cloud is basically through the internet and that has some security and performance (delay) concerns as well.

Despite the concerns of cloud computing, many businesses see its viability and use especially for specific purposes and periods of time. As and example, a company located in Norway with a multi-tenant hosting provider that has a cap on network and of course limited bandwidth but needs to download large amounts of weather data from the united states at specific times may want to setup this server in a cloud located in the the United States for closer proximity and to maintain current costs with its hosting provider by maintaining the bandwidth cap limits. The cloud makes it possible to extend ones infrastructure over a large geographical region without concern for elaborate contract negotiations and high initial costs of equipment.

2.6.1 Amazon Web Services

Amazon web services is an example of a cloud computing services that is available for use and has become increasingly popular[9]. Like many of such implementations of cloud computing, Amazon web services is designed with with idea of proving infrastructure as a service. One only needs to sign up and then one can start deploying services as needed. There are different basic offers of hosts available for setup with different kinds of templates from different kinds of operating systems as well. Five basic kinds of machines are offered :standard, small, large, xlarge, high CPU:medium and high CPU: xlarge; these have different performance offerings and prices and a made to be suitable for different kinds of services and their requirements. The pricing scheme of Amazon web services, is to charge clients for hosts that are powered on. Other costs are incurred by acquiring additional disks, IPs and additional services. The basic scheme of being charged for periods during which hosts are up brings with it several possibilities for business that need extra servers running for short or specific periods of time.

2.7 Manage Large networks (MLN)

Manage Large Networks (MLN) is an administration tool for building and running virtual machines. MLN can be used to manage virtual machines that run on Xen, VMWare Server and User-Mode Linux [10]. MLN eases laborious tasks such as creating multiple instances of virtual machines and managing them as groups. It uses the concept of "project" which is a group of virtual machines. This concept makes it useful to allow administration of groups of virtual machines[11]. MLN has many desired concepts and functions that makes it a preferred choice in this experiment. The limited time available for the execution of this project means that everything cannot be built from ground up. Using MLN running as a client, it is possible to send commands for changes to be made on the running virtual machines and to startup other virtual machines if desired.

Creating and Upgrading projects

To create a project ín MLN, the description of the project is stated through the mln configuration language. The configuration language allows one to specify things such network settings, Random Access Memory (RAM), and the host on which virtual machines would reside. A sample of an MLN project as given in the mln manual[11] is shown below.

```
        global {
        project simple_network
        }
        switch lan {
        }
        host startfish {
        network eth0 {
        switch lan
                address 10.0.0.1
                netmask 255.255.255.0
        }
        }
        host catfish{
        network eth0{
                switch lan
                        address 10.0.0.2
                        netmask 255.255.255.0
        }
}
```

Listing 2.1: Example code for mln

The sample project defines two hosts and sets their ip addresses on the eth0 interface to static addresses defined. Both hosts are connected to the same virtual network swicth. To create a project, the configuration file is built using the mln build command.

```
mln build -f project-file.mln
```

Listing 2.2: building an mln project

It is possible using mln, to upgrade a running project. What this means is that, a the setup of the virtual machine is changed to a newly provided one and booted. Moving a virtual machine from one host to another is possible. This can be done whiles the virtual machine is still in use and thus is can be used to implement live migration, a feature discussed later in this text.

```
mln upgrade -S -f new-project-file.mln
```

Listing 2.3: upgrading an mln project

There are other useful commands for doing many other system administrative tasks and getting information about the state of running virtual machines that one would want to have. In addition to being able to manage virtual machines through mln commands on an mln server, mln enables the use of mln client commands.

2.8 Virtualization and resource management: The state of the art

There are many features that are being introduced into virtualization to contribute to better management of resources. One of such methods, which is a typical example of the popular approach is VMware's Distributed Resource Scheduler (DRS). DRS works by continuously monitoring resource utilization within the virtual infrastructure and intelligently aligning these resources according to the needs of business. According to [12], DRS enables one to

- Dynamically allocate IT resources to the highest priority applications. Create rules and policies to prioritize how resources are allocated to virtual machines.
- Give IT autonomy to business organizations. Provide dedicated IT infrastructure to business units while still achieving higher hardware utilization through resource pooling.
- Empower business units to build and manage virtual machines within their resource pool while giving central IT control over hardware resources.

As part of the drive to cut down costs of running datacenters, Vmware has come up with what they call Distributed Power Management (DPM). What DPM does is to continuously monitor resource requirements and power consumption across a virtual cluster and to consolidate workload thereby freeing up some virtual hosts and then putting the freed up hosts into standby mode in order to save power. When system requirements increase, the virtual machines are redistributed after powering on the host(s) in standby. This feature is currently classified as an experimental feature but does reflect the direction and mindset of such developments to save power; making the virtual infrastructure decide and implement operations logic.

2.9 Live Migration

Live migration is a feature of virtualization that allows an instance of a running operating system, to be transferred unto a different physical host [13]. Live migration offers many benefits by introducing flexibility into the infrastructure. A requirement in a virtual infrastructure setup that allows live migration is to have the physical hosts between which the migration is going to be made, have a shared storage between them. Having shared storage means that the filesystem of the virtual machine remains at its location. The session and the other system resources it is using however, are transferred to the destination host. As indicated in the section on types of virtualization, live migration is not possible when doing operating system virtualization.

Live migration is one of the features of flexibility that virtualization brings. It enables maintenance on physical hosts and even decommission and commissioning of such whilst avoiding downtime as far as services are concerned. It is possible to simply migrate a virtual machine running a service from an old host that needs to be decommissioned to a new one without those using services provided by the virtual machine or even those logged on through a terminal session noticing. This feature thus contributes to increased uptime as compared to having a physical infrastructure, for which swapping to a new server would require several hours of setup of a new machine, configuration of services and most likely some downtime in order to do the switch.

As mentioned earlier, the performance of a virtual machine is affected by the performance of the host on which it resides. This important fact means that, with live migration it is possible to change the performance of a virtual machine. Migrating a virtual machine to a host that has more CPU speed and is less crowded for example would improve the performance of the virtual machine somewhat. Live migration can thus be used as a quick way to improve the performance of a virtual machine.

Live migration in a hardware assisted environment is possible but works best if the physical hosts have similar virtualization features on their CPUs. This ensures that the virtual machines would work the same way and perform in a a like manner after a migration.

2.10 Performance of virtual machines

A few factors affect the performance of a virtual machine. As indicated earlier, the resources (CPU) available to the hosts directly influences this. In addition to the technical specifications of the hosts, the number of virtual machines residing on the host play a role. Obviously, the more crowded a host is the more it would take to time-share between the virtual machines. A virtual machine, all things being equal, would perform better on a host that has less virtual machines residing on it. It is thus a challenge to find the optimal number of virtual machines per host and still make use of as much resources as possible.

In addition to the sheer number of virtual machines on a host, the amount and type of load on the other residing virtual machines affects what resources are available to the virtual machine in question. If a neighboring virtual machine is utilizing a lot of bandwidth on the network for example, the bandwidth available to others is reduced. This is pretty much the same as having several physical machines sharing the same hub or internet connection.

There are a number of well known issues that accompany virtualization. Time drift is one of such. It is a challenge when several virtual machines are sharing CPU time to keep these systems in sync with the system click. Generally, virtual machines tend to drift off over a period of time especially when there is high load on them and during migration. A few approaches exist to curtail this challenge. Aside a few application and vendor specific solutions, network time protocol (ntp) client is usually used to get over this hurdle.

Chapter 3

Methodology

This chapter describes the design of the model used and the implementation of the experiments including justification for choices of approaches and options made as well as discussing some of the expected results.

3.1 Objectives

The objectives hoped to be attained by the experiment are stated in a clear manner here. These objectives are in line with the overall purpose of this paper and questions raised in the introductory chapter about virtual machine management towards a better resource management. The experiment is designed to mimic the administration of virtual machines within a multi-tenant virtual hosting environment. It assumes scenarios where tenants would want to implement especially migration policies on multi-tenant hosting environments with cost saving and resource management in mind and without much intervention from the hosting providers.

- Design sample policies that would run in a mult-tenant environment
- Design a framework that would be able to run policies by using constraint-based configuration files
- Perform quality of service driven management
- Implement and analyze policies with framework

In addition to testing the possibility of having decision-making ability or policy for that matter, be implemented on the virtual machine, the experiment aims at making it possible to alter policy only based on setting defined constraints. These would be accessible through configuration files. Thus, management and policy alteration would be simple to do an that would mean that the method can be scalable. System administrators are used to configuration management through files and thus have many familiar and reliable tools exist which can be used to conveniently implement such a system on a large scale.

3.2 Design of Experiment

The experiment conducted consists of a number of different scenarios which would be described in a more detailed manner later. There are, however, some assumptions that run through all the scenarios. Each scenario reflects a set of possible combinations of policies. The policies define different strategies that are used to manage the virtual machines. The main feature that would be used for implementation of operations logic throughout the experiment is live-migration. This is because live migrating would ensure continuos service availability as well as adjusted performance as desired and described by the policy. For this purpose there is a shared storage accessible to all the physical hosts being used.

It is assumed that, the environment in which the virtual machines of interest run, is multi-tenant. This assumption has some implications on the scenarios that are implemented later. Having a multi-tenant environment means that, the environment could very easily have a wide dispersion of policies which are compartmentalized due to differing services and clients or owners of the virtual machines. As indicated in the section 2.4 “Multi-tenant virtual hosting environments” on page 11, there is also competition for resources in such an environment. For the sake of simplicity, all the virtual machines would be the same in specifications except for policy being implemented and the service they run.

3.3 Definition of terms

Virtual Machine

For the purpose of this experiment, the term virtual machine refers to a virtual machine with a particular set of properties. These properties are

- a virtual machine running a service
- a virtual machine capable of performing monitoring of itself relative to its service
- a virtual machine capable of making decisions based on a defined policy which addresses the desired service level of the the tenant
- a virtual machine capable of sending requests for implementation of desired operations logic to a virtualization management application

Service

The service being run on the Virtual Machines are chosen to have certain desirable qualities. The services for which this approach would be suited are service that are network-based or that are dependable on other generated traffic or user activity. That means that they are accessed across a network and as such its patronage generates network traffic. The network traffic also, should be varying and should have an unknown pattern. An unknown pattern means that the pattern is not fully predictable by both the tenant and the infrastructure. The unpredictability of the service is the justification for monitoring.

If the pattern of behavior was predictable, scheduling alone would have been sufficient for implementation.

Strategy

Strategy in this context refers to the possible options in decision making that can be taken based upon monitoring information that is captured and processed.

Policy

The policy is responsible for the kind of operations logic and the conditions under which they are implemented. The policy is specific and individual to each virtual machine and thus many forms may be implemented by a single tenant that has many virtual machines. It dictates the behavior of the virtual machine.

3.3.1 Performance

To be able to know if a virtual machine has enough resources to conveniently offer the services it is meant for, a choice is made to monitor the service in question. This choice is made because the experience of the users (people of other services) is what is regarded as paramount. The performance and how it is defined and monitored depends upon the services being offered by the virtual machine and the interest of the policy being implemented. It is important, however, that this performance indicator be such that, it reflects the effects of actions taken to adjust the performance by the policy.

Provider, Customer/tenant, Migration

As mentioned in the section 2.4 "Multi-tenant virtual hosting environments" on page 11 in the background chapter, with the conventional approach to virtual machine management, an administrative issue is introduced because of rights and permissions management and different offerings of virtual machine management approaches and interfaces by different hosting providers. This is depicted in figure 3.1. With the approach being proposed in

this paper, the variety introduced by conventional design would be done away with. An approach in which policy is implemented by changing constraints and strategies when desired locally is easier administratively. To make things even simple, a configuration management application can be used to manage and maintain these configuration files that define policy. Figure 3.2 shows the administrative implications of this approach.

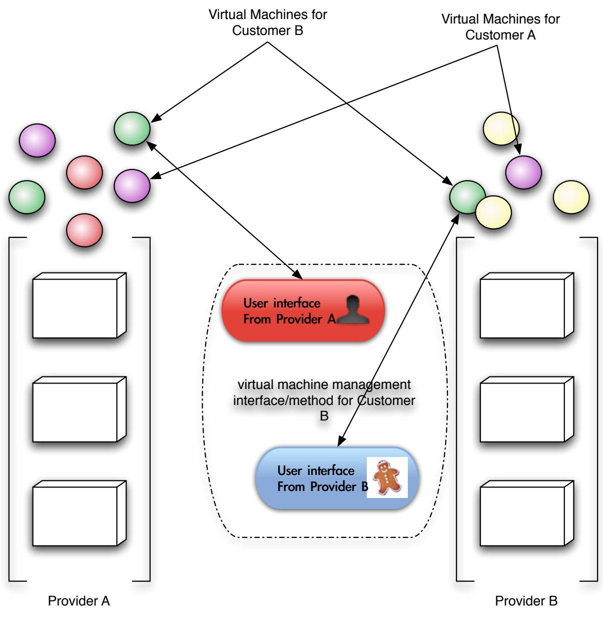

Virtual Machines for Customer B
Virtual Machines for Customer A
User interface From Provider A
virtual machine management interface/method for Customer B
User interface From Provider B
Provider A
Provider B

Figure 3.1: Management paradigm in multi-tenant virtual hosting environments

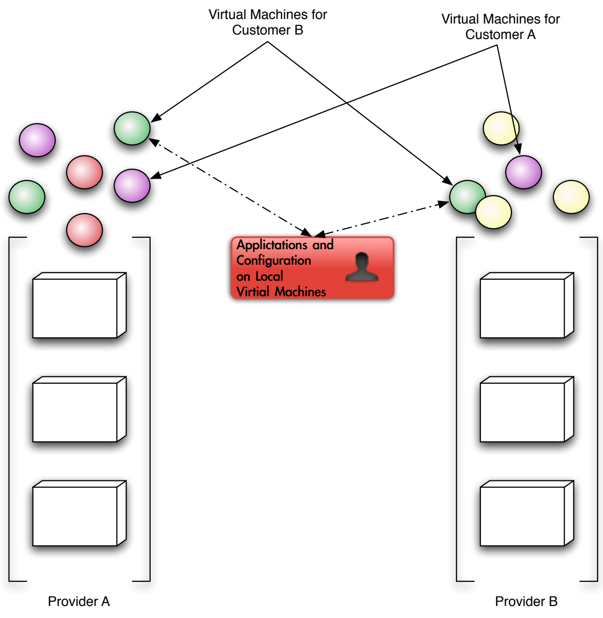

Virtual Machines for Customer B
Virtual Machines for Customer A
Applictations and Configuration on Local Virtial Machines
Provider A
Provider B

Figure 3.2: virtual machine based management

3.4 Policy Design and Overview

Four policies are defined upon which different scenarios would be created for different experimental setups. The goal of the policies are to make requests for migration as a mechanism to affect the performance of the virtual machine to which the policy is applied. The fourth policy, however, works slightly differently; it makes use of a resources across a different hosting provider. Since a shared storage is necessary for live migration, that is not done but rather, a clone of the running virtual machine is made on the different hosting provider, and jobs sent to the clone for processing. The policies grow in complexity from policy 1 through policy 3. There is an assumption of a setup that has several hosts that are viewed as though they were in an array. When a request from migration is made, it simply means, the virtual machine is requesting to to shifted to the next host in this array. Policy 2 introduces the term base. The base is a special host which the policy considers to be outside the array. This means it would never migrate to the base unless it specifically requests to be migrated there. The Base is this special host that would house the virtual machines that do not require so many resources or would house them at the time they do not require them. The idea of a base is a cost incentive one. Lower costs and full use of resources are the incentives. The term "move to base" would be used to refer to the special case of a migration to base.

In all policies, there is a comparison of current performance values to previous performance values. This current performance value is compared to a specified threshold value and decisions made based on the outcome of this comparison. The comparison of the current the specified value is done by taking the last n recorded values and seeing if they all hold true for the given comparison. A comparison is made in order to be able to specify a delay before the implementation of the operations logic desired. Reasons for the delay are further explained later in section 3.5 on page 33

3.4.1 Hosts as arrays

The concept of viewing hosts as arrays is an important one in the implementation of the chosen policies. Viewing each cluster of hosts as an array allows prioritization of these hosts in a simple way which can very easily be implemented programatically through already established methods of sorting. It allows for the possibility of dynamically re-prioritizing hosts to reflect resource usage. Arrays makes it convenient to extend and scale policies over a wider domain when necessary. The array is basically a set of hosts that are in a cluster and have a shared storage among them. This means that, it is possible to perform live migration between nodes (hosts) of an array. The properties of an array makes it rather convenient to set boundaries on hosts to which virtual machines can migrate to simpler by changing the range and some sorting.

An array would typically represent a single hosting provider, data center or site and could as well represent different clusters of hosts owned by the same provider. The cluster is assumed to have be on the same network or subnet and have shared storage. Because of

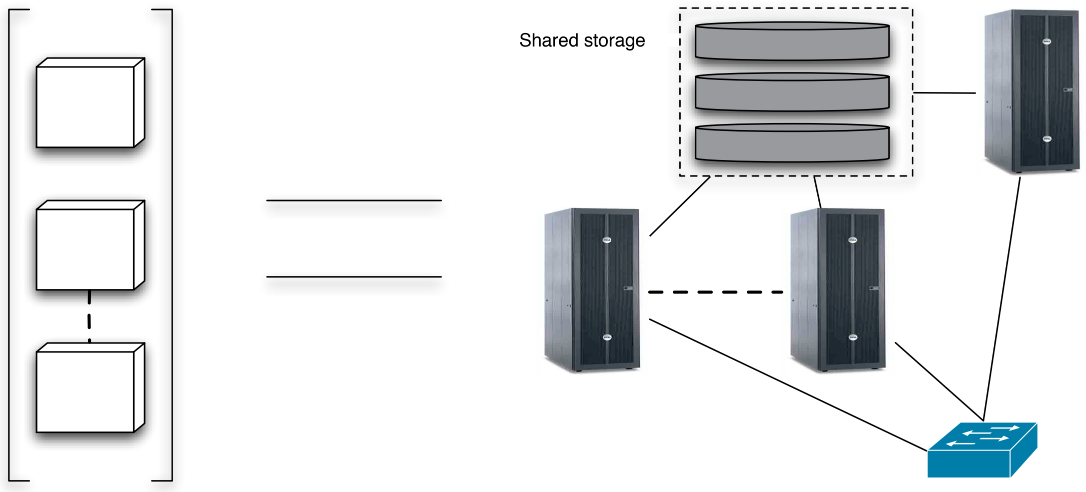


Figure 3.3: An array: A cluster of hosts with shared storage and network

shared storage and network, live migration is possible within an array. Figure 3.3 is a pictorial representation of an array; it shows the components of an array.

3.4.2 Policy 1: Threshold-aware

In policy 1, the virtual machine monitors performance of its service and does a comparison with previous values obtained. The comparison is to find out how long the value of the performance indicator (V) has been in a given range. V is a performance indicator that increases as performance decreases or vice versa. The exact behaviour of the performance indicator depends on the service and the particular indicator being measured. If the value in question decrease as performance decreases then it means lower values of v are desired. The period (t) spans over a number of previous values, and is determined by the policy. This would affect how quickly or otherwise, a virtual machine reacts to a change in performance. A value X of performance, which is predetermined by the policy, acts as the threshold value to which v (current value of performance indicator) is compared and the outcome of which triggers a request for migration or otherwise. It is the aim of policy 1, to relocate the virtual machine to a host that keeps the performance indicator below the given value, X.

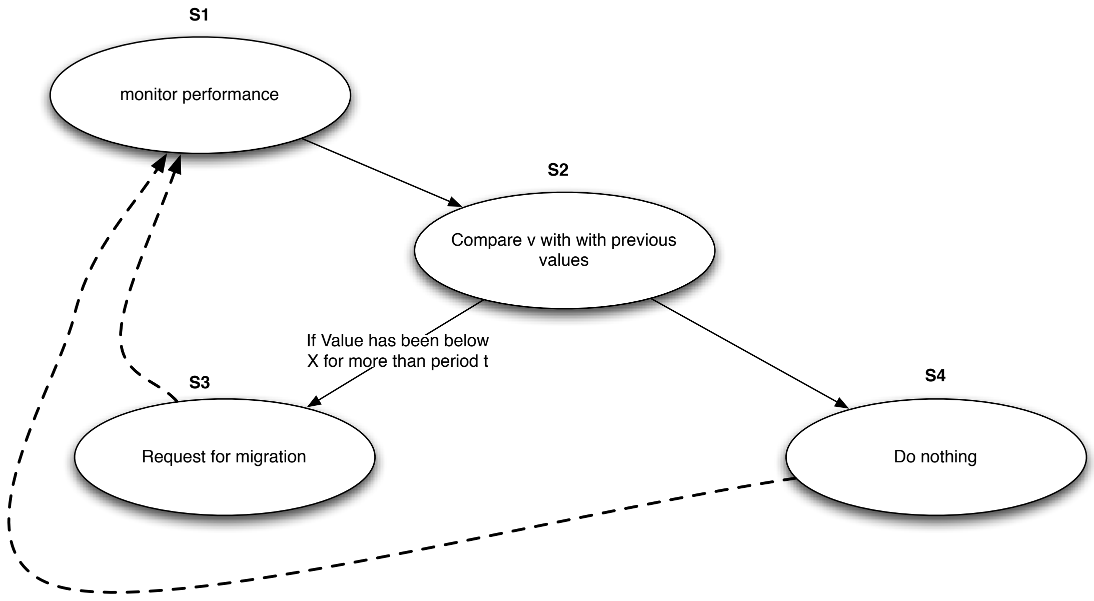

S1
monitor performance
S2
Compare v with with previous values
If Value has been below X for more than period t
S3
Request for migration
S4
Do nothing

Figure 3.4: policy 1 - Single threshold-aware policy (migration request based on single threshold value)

```
read configuration file
X = threshold
t = delay
while service running
        {
        monitor performance (v)
        if v >= X
                do counter +1
        else
                do counter = 0
        endif

        if counter > t
                do request_for_migration
                do sleep
        }
```

Listing 3.1: Policy 1:threshold-aware

3.4.3 Policy 2: Base-aware

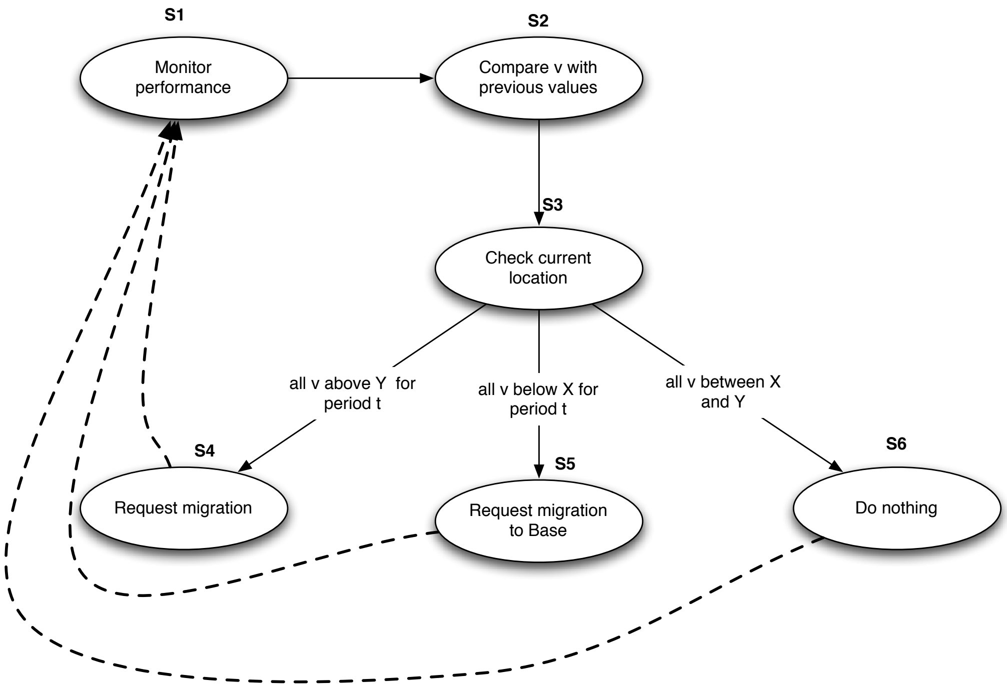

S1
Monitor performance
S2
Compare v with previous values
S3
Check current location
all v above Y for period t
all v below X for period t
all v between X and Y
S4
Request migration
S5
Request migration to Base
S6
Do nothing

Figure 3.5: policy 2 - "Base-aware" policy with multiple threshold values

In policy 2, the virtual machine monitors the performance of it's services, obtaining a current value (v) as in the policy 1, The current value (V) is compared with occurrences that occurred over the period t as defined by the policy in the same manner as explained the the threshold-aware policy. The policy defines two threshold values of the performance indicator, X and Y; these values define a range of values which reflect the desired performance level for the service. X is the lower threshold and Y the higher. If the current value (V) is in-between X and Y, then, the current location is considered to be optimal and therefore desired. If however, the current value (V) goes below X, it is assumed that, the is too much extra resource and the virtual machine is over-provisioned, therefore the virtual machine requests a move to base if it is not already located at the base. On obtaining V to be above Y, the virtual machine requests a migration to a different location other than the base. The amount of time (t) for which a virtual machine should wait could differ for both kinds of thresholds. Thus two new values, low_t and high_t are used in the explanation of the policy through pseudocode.

```
read configuration file
X = high_threshold
Y = low_threshold
high_t = delay_for_high_threshold
low_t = delay_for_low_threshold
while service running
        {
```

```
        monitor performance (v)
        if v >= X
                do high_counter +1
                do low_counter = 0
        else
                do high_counter = 0
        endif

        if v <= Y
                do low_counter +1
                do high_counter = 0
        else
                do low_counter = 0
        endif


        if low_counter > low_t
                do request_for_migration_to_base
                do sleep
        if high_counter > high_t
                do request_for_migration
                do sleep
        }
```

Listing 3.2: Policy 2: Base-aware

3.4.4 Policy 3: schedule-aware

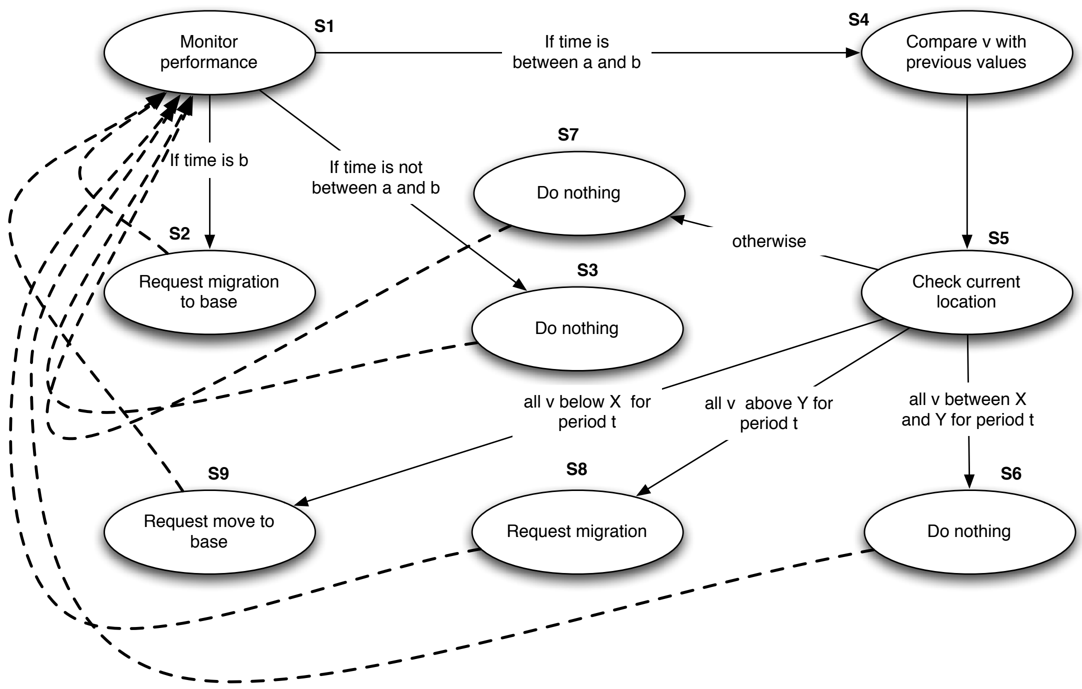

S1
Monitor performance
If time is between a and b
S4
Compare v with previous values
If time is b
If time is not between a and b
S7
Do nothing
S2
Request migration to base
otherwise
S5
Check current location
S3
Do nothing
all v below X for period t
all v above Y for period t
all v between X and Y for period t
S9
Request move to base
S8
Request migration
S6
Do nothing

Figure 3.6: policy 3 - “schedule-aware”, multiple threshold value policy

Policy 3 is a schedule-aware policy that aims at providing a certain level of performance during specified hours of the day (working hours). When the working hours are over, the virtual machine makes a request to be migrated to base and stays there without any monitoring until the start of another working period. At times outside the working hours, no requests are made for migration. Monitoring of service can be stopped to reduce the processing units taken up on the virtual machine. During the working hours of the virtual machine, the current value (V) is compared with the predefined thresholds (lower threshold X and higher threshold Y). The current location is considered a desired one if v is found to be between X and Y. In cases where the current value (V) is found to be below X, a request to be moved to base is made. On the other hand, if the current value, v is found to be above Y, a request for migration instead. This policy behaves the same way as the base-aware policy (policy 2) during its working hours.

```
read configuration file
X = high_threshold
Y = low_threshold
high_t = delay_for_high_threshold
low_t = delay_for_low_threshold
while time is between a and b
        {
        monitor performance (v)
        if v >= X
```

```
                do high_counter +1
                do low_counter = 0
        else
                do high_counter = 0
        endif

        if v <= Y
                do low_counter +1
                do high_counter = 0
        else
                do low_counter = 0
        endif


        if low_counter > low_t
                do request_for_migration_to_base
                do sleep
        if high_counter > high_t
                do request_for_migration
                do sleep
        }
while time is not betweeen a and b
        {
        do sleep
        }
```

Listing 3.3: Policy 3:Schedule-aware

3.4.5 Policy 4: Site-Aware

Policy 4 is a site-aware policy. The idea behind policy 4 is to make use of different pricing schemes and offers of different kinds of multi-tenant virtual hosting environments by being able to make use of resources between the two as the need arises. Like what most businesses would be interested in, this sample policy is designed to make use of the cheapest provider as much as possible. When a queue needs to be process which meets a certain specified category, such as a high CPU demand and would perhaps take too much tim to process locally, the virtual machine migrates to a site which can better handle the task. The idea behind is that the local or home array is cheap but does not have as much resources needed to complete the task in the desired time.

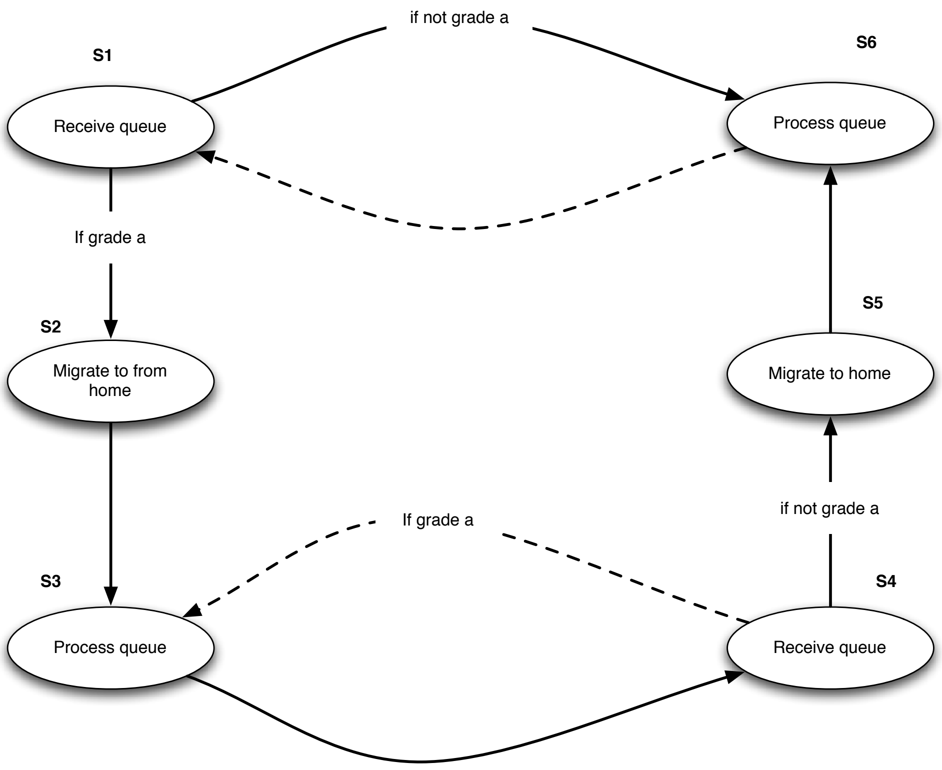


Figure 3.7: policy 4 - site-aware policy

```
read configuration file
Z = threshold
a = big_job


while service is running
        {
        receive entire queue
        if grade not a
        do check location
                if not in home
                {
                        do complete current job
                        do migrate away from home
                }
                if in home
                {
                        do continue processing jobs
                        do complete queue
                        do receive new queue
                }
        if grade a
        do check location
        if not in home
                {
                        do continue processing jobs
                        do complete queue
                        do receive new queue
                }
        if in home
                {
                        do migrate to home
                }

        }
```

Listing 3.4: Policy 4:site aware

3.5 Policy Discourse

The aim of managing resources is to reduce wastefulness and essentially to reduce costs. Clearly, some of the policies are more complex than others and their cost saving capacities differ as well.

In the threshold-aware policy (policy 1), it is clear that cost savings is not as high a priority as in the rest of the policies since there is no inference of different pricing schemes or priority levels in its design and presentation. Keeping the performance of the running service above a specified level is the only concern here. This policy aims at proofing the concept of affecting system performance and in effect service performance by live-migrating. The virtual machine would just keep migrating away from the host that does not have enough resources to support it. It is also the case that the threshold-aware policy could cause the virtual machine to go into a state of continuous migration because it is not aware of any levels of resource availability on the hosts and thus would not know when the virtual machine is residing on the best host available to in the array. It could easily migrate to a worse place than its current location even if the location in question happens to have more resources than the next but monitoring detects a fall in the performance of the running service. This policy does not take cost into account and has the primary aim of making an attempt to maintain a certain level of performance as far as the service in question is concerned.

The base-aware policy (policy 2) is a significant improvement to the threshold-aware policy. The concept of the base introduces a new possibility and advantage of cost saving. It is possible here as well, for the virtual machine to go into a state of continuous migration. The advantage of having two thresholds is to define a range of desired performance. This range servers as the constraints that bound the level at which one desires to operate. When the performance is above a given threshold, the policy seeks to utilize less resources in order to save cost because that performance level is luxurious and way above what is required. Presumably, one would be paying different prices for keeping hosts up in the two different arrays; the base and the "normal" array.

In comparing the cost incurred by virtual machines in implementing these sample policies, we consider the cost of renting location on a host to be C. C_t is the total cost incurred by the tenant in keeping the virtual machine up. C_b is the cost incurred by the client in keeping the virtual machine in the base or base array. C_a is the cost incurred by the client in keeping the virtual machine running in the array. The total cost incurred would be a sum of the costs of incurred at both locations. Thus

$$C_b < C_a \leq C_t \tag{3.1}$$

$$C_t = \sum C_b + \sum C_a \tag{3.2}$$

In the schedule-aware policy (policy 3), the strongest push for cost saving is made. In addition to the advantages of the base-aware policy, the schedule-aware policy introduces a schedule for which on can specify mandatory states. Thus, a combination of dynamic service driven behaviour and static, mandatory time driven states are made. The virtual machine is forced to migrate to the base host during specific periods irrespective of performance. In this setup, monitoring still continues during that period in order to keep track of what happens. In a system where one is rather confident and finds it unnecessary, monitoring could be stopped to further reduce the resource usage during this period. In comparison to the base-aware policy, there is a period of time that one can be certain that there is cost saving. Maintaining the same definitions for C_b and C_a already made and defining two new variables C_tb and C_sb, with C_sb as the total cost incurred by running a virtual machine on the base during its scheduled period and C_tb being the total cost of running the virtual machine in the base, we get the following equation

$$C_{tb} = \sum C_b + \sum C_{sb} \tag{3.3}$$

$$C_t = \sum C_{tb} + \sum C_a \tag{3.4}$$

Policy 4, the site-aware policy, demonstrates how managing virtual machines with this from within the virtual machines can bring ease in managing virtual machines across different sites and even hosting providers. This more clearly shows the advantage depicted in 3.2. The policy, the policy in its implementation would make use of a cloud computing platform. The Amazon web services would be used as a platform to handle the processing of certain kinds of loads. Transferring jobs to different “sites” may be desirable and advantageous for certain environments. As spelt out in the Background chapter, the performance of virtual machines are affected by the load of other virtual machines residing on the host as well. Having policies which prevent heavy loads from being run on specified hosts or arrays would mean better performance for all the virtual machines in question. In addition to managing the load and thus overall performance of the hosts, one may want to send particular loads to other sites because of time. The processing time used for processor intensive tasks would be shorter on virtual machines that resides on a host with higher CPU speed and cores, all things being equal. This kind of policy cannot be implemented using the “usual” method of virtual machine management. In addition to allowing some form of control of virtual machines on different arrays (sites), one gets to manage these in the same way and thus one does not have to learn the usage of different kinds application as one is used to.

Policy 4 demonstrates how virtual machines can be made to implement a policy that makes use of different hosting providers or sites. It is used for processing both deterministic and non-deterministic job queues. Various queuing policies can be used to process job queues [1], and depending on what exactly one wants, one may choose one over the other. In this policy a specialist queue in used. Jobs are tagged with priority numbers which form the basis for discrimination or decision making. This policy allows a virtual

[1]More information on standard ways of processing queues can be found in [14]

machine to migrate between sites. Because of the lack of a shared storage been shared between the hosts, a downtime is needed for migration because live migration would not work. Justification for the implementation of such a policy would be because the second provider is perhaps better with a certain kind of resource for which we need large amounts to process certain kinds of jobs such as CPU or bandwidth. One may want however, to run on the first site because it is cheaper.

This kind of policy can work in example case of a build server or graphics rendering server that works on jobs of different magnitudes. There would be certain lengths of queues containing certain magnitudes of jobs for which it would make sense to have some down time and get more CPU power to complete the tasks in good time.

If we assume time spent processing a queue to be the factor for determining where a server should be located, then the time spent in both migrating and processing at the faster array should be far less than that spent in the slower array. Taking the home array to be the slower, with the time spent on processing a particular queue being T_{PH}, the time spent migrating from the home array to the faster array being T_M and the time spent processing the same queue on the faster array being T_{PA}, we come up with equation 3.5 as the incentive for migration away from the home array.

$$\boxed{T_{PH} >> T_M + T_{PA} \qquad (3.5)}$$

T_{PH} and T_{PA} can be calculated in each receipt of queue using information about the queue, and with well know queue processing models. The time taken to migrate, T_M, is proportional to the size of the virtual machine as well as inversely proportional to the effective bandwidth between the two arrays.

3.6 Expected Results

The results expected consist mainly of logs from scripts for monitoring and migration requests as well as from mln. These logs would show the behaviour of the virtual machines under the conditions under which they are placed. The scripts are designed to be very chatty and as such create log files of every interesting activity. Logs of both monitoring and requests would be kept on the virtual machines. In the analysis of the results, plots of performance indicators would be compared with logs of movement. The logs generated from the scripts are inscribed with time-stamps to make reconciliation possible.
In addition to the status of individual virtual machines, the number of virtual machines at different times and how long they reside on a particular host is also of interest. This is perhaps more important in policies that include the concept of the base. This would be reflective of what costs savings have been achieved. A log of movements would be kept. It is expected that there would be high migration activities when load is high and little or no activity when load subsides.

3.7 Challenges

Some challenges arise in the design of experiment. Deciding to monitor the services from within the virtual machine is impacted on by the time drift phenomenon of virtual machines discussed in section 2.10.

Finding the exact location of a virtual machine from within the virtual machine is also a challenge. The is no sure way of doing that and some assumptions of location may have to be made to begin with. This may require, for the sake of convenience that virtual machines would have to be deployed on specific hosts so as to know what the initial location is.

3.8 Environment

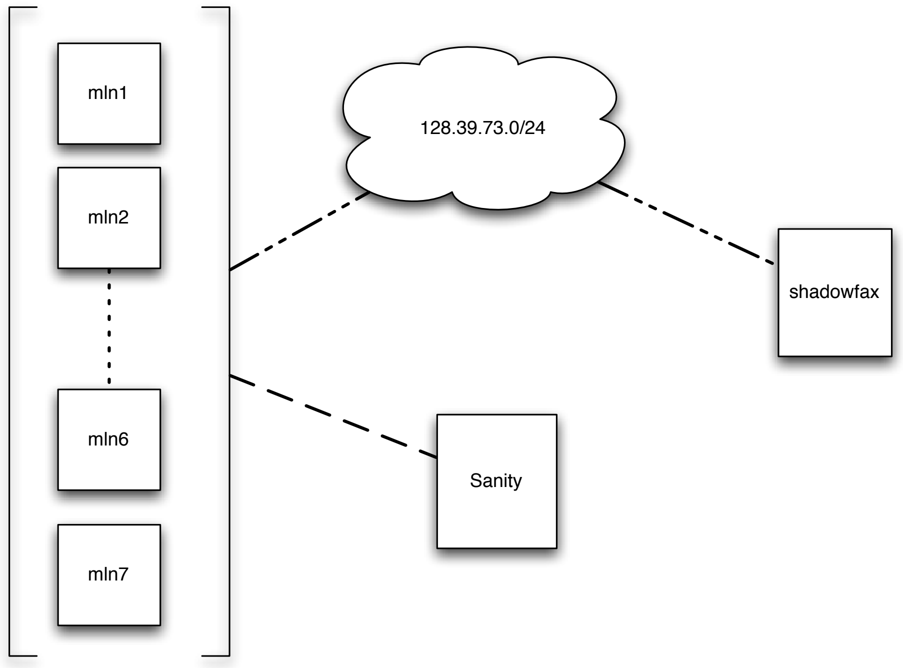


Figure 3.8: Experimental Environment

The environment used consists of a cluster of 7 machines which serve as hosts for the virtual machines (mln1, mln2...mln7). MLN is installed on shadowfax and that serves as the management machine in this environment. Shared storage is provided by sanity which serves as the SAN (storage area network) server through iscsi.

Software	version	Role
mln	ver 1.0.2	virtualization management program
perl	v5.8.8	compiler for scripts
mysql	Ver 14.12	database server
apache	Apache/2.2.3	web server

Table 3.1: Software Environment

Hostname	CPU	Memory
mln1	2 x 2.0 Ghz	8.0 GB
mln2	2 x 2.6 Ghz	8.0 GB
mln3	2 x 2.6 Ghz	8.0 GB
mln4	2 x 3.0 Ghz	8.0 GB
mln5	2 x 3.0 Ghz	8.0 GB
mln6	2 x 3.0 Ghz	8.0 GB

Table 3.2: Hardware Environment

A table containing the list of software used, their versions and roles played in the experiment is shown in 3.1. The particular software used was chosen because they are widely used, free or open source.

Specifications of the hardware used is shown in 3.2.

3.9 Process

3.9.1 Preliminary Tests

Before the start of the experiments, certain tests were made to prove some assumptions made. One of such preliminary tests was to compare processing times of a simple but processor intensive job between different offers of the Amazon web services by simple commands. No complicated tools were used for this purpose but the results are considered valid for the purposes of this experiment. Although the system time remained pretty much the same, the real time used to execute the job was rather very different and with a reverse proportion to the CPU specifications of the virtual machine.

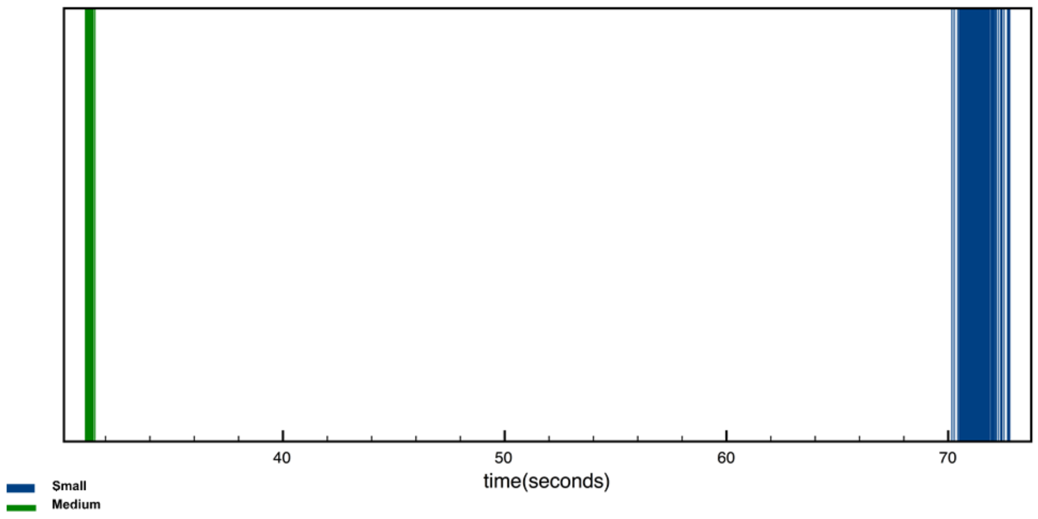


Figure 3.9: Real time taken for small and medium virtual machines to process 500 iterations of an exponential computation

Figure 3.9 shows plot of real time taken to do 500 iterations of an exponential calculations (CPU intensive). A table containing the mean, median and range of real, system and user time are shown in table 3.3. The specification of the virtual machines are given in table 3.4. Amazon uses its own convention for indicating CPU speed, the EC2 compute units.

It can be seen from both the table of distribution and the plot of time spent that, the large instance takes a far less time to complete CPU intensive processes. The time different times taken by different instances of the same virtual machine to process the same task can be a strong incentive to change instance types when certain tasks are at hand and time is more important that cost.

Time	**Small**			**large**		
distribution	**mean**	**median**	**range**	**mean**	**median**	**range**
real	71.087114	71.0185	70.167 - 72.774	31.207246	31.1975	31.093 - 31.523
user	30.71354	30.7	30.25 - 31.04	31.11438	31.1	31.04 - 31.41
system	0.0246199999999998	0.02	0 - 0.09	0.0186799999999999	0.02	0 - 0.06

Table 3.3: Times taken for computation on small and large virtual machines

Type	CPU	Memory
small	1 EC2 Compute Unit	1.7 GB
large	4 EC2 Compute Units	1.7 GB

Table 3.4: Specification of small and large instances of virtual machines

3.9.2 Choosing services

To keep the experiment simple within the time frame given, only two popular services were chosen to be used in implementation. These were web service and database service. These were chosen because they are probably the most common forms of service run by business that are made available to customers. Although in most cases, the database service is usually a back-end to the the web, they were made separate for the sake of simplicity. The choice of a database server was mysql server. This is because it is popular, easy to setup and free. Apache was used as the webserver for similar reasons to mysql server.

3.9.3 Choosing performance Indicators

The choice of a performance indicator for the database was rather straight forward. Mysql has a parameter call slow-queries which is an indication of the response performance of the database server. It is that the better the performance of the server, the less the slow-queries.
The choice of a performance indicator for web-service was more unclear. After going through a number of tests, the idle workers value, which can be read when the server-status feature is enabled is used. The assumption here is that, one would have more idle workers if processes are handled more quickly.

MySQL Monitoring

Monitoring the performance of the database server (MySQL in this instance) was been fairly straightforward. According to the criteria for selecting a performance indictor, already discussed. The slow queries feature of mysql was chosen. Querying for a status of mysql returns among other parameters, the total number of slow queries executed on the running instance. It makes it possible for one to be able to retrieve this value intermittently and find the difference, thus obtaining a rate at which slow queries occur. According to the Reference Manual for MySQL, the value returned for slow queries in the status command are the total number of queries that took longer than long_query_time, a value set in the the configuration; seconds to complete. It is apparent that if the conditions under which the database is running improves, these queries would not take as long as they do and thus this value would drop. This value, the difference, would this be low under bad conditions and higher under worse conditions. Slow queries thus meets the requirement spelt out as necessary for the choice of a performance indicator.

Apache Monitoring

Monitoring the performance the webserver was done by enabling the server-status functionality of apache. This gives among other values, the idle workers. The idle workers was chosen as a the performance indicator because it gives an idea of how much pressure the webserver is under. If the server is responding quickly to requests, there would be more idle workers most of the time. If however, there load is high on the server, the idle workers would decrease. This value would thus be high under good conditions and lower under worse conditions.

3.10 Keeping track

It is necessary to keep track of changes in configuration done automatically in response to threshold set for values being monitored. To do this a lot of logging is necessary. For convenience and scalability, a central logging system was considered. It would be too much work to keep track of different logs on many different systems as they grow in number. Syslog central logging was considered and implemented but discarded after a few tries because it was not considered ideal for the purpose of this experiment. It might however be ideal in a real life scenario where a log analysis program maybe present and not as much verbosity on the logs may be needed for day-to-day running. In the place of syslog, a custom server/client script using sockets was used. The server-side script was designed to have an output to a file thus it was possible to have logs for different runs and scenarios separated.

Listings 3.5 and listings 3.6, show the perl scripts that were developed for this purpose, the former being the server-side script and the latter, the client side. The client-side script ensure that each message sent was preceded by a timpe-stamp and the hostname of the client sending the information. It is, however, important to not that the logs reflect the requests made for live migration to be effected and not necessarily live migration itself since the virtual machines only initiate migration and not effect it.

```
#!/usr/bin/perl

my $logfile = $ARGV[0];
my $port = 31000;
use IO::Socket;


my $sock = new IO::Socket::INET(
          LocalPort => $port,
          Proto     => 'tcp',
          Listen    => SOMAXCONN,
          Reuse     => 1);

$sock or ( warn "no socket: $!\n" and return);
STDOUT->autoflush(1);
while (my $new_sock = $sock->accept()) {
     my $date = `date`;
     my $time = time();
     chomp $date;
     my $line = <$new_sock>;
     chomp $line;
     my $line = "$time $date $line";
     print "line: $line\n";
     system("echo $line >> $logfile");
}
```

Listing 3.5: server-side central logging script for central logging

```
#!/usr/bin/perl

my $message = $ARGV[0];
my $port = 31000;
my $hostname = `cat /etc/hostname`;
chomp $hostname;
use IO::Socket;

$sock = new IO::Socket::INET(PeerAddr => "mainsyslog.dyndns.org",
      PeerPort => $port,
      Proto    => 'tcp') or die "Failed to open connection: $!\n";


my $message = "$hostname: $message";
print "sending: $message\n";
print $sock $message . "\n";
close ($sock);
```

Listing 3.6: client-side central logging script

Tracking the location of a virtual machine in policy 4 is much simpler than that in the preceding policies. Because the virtual machine is expected to cross arrays, the network on different arrays is expected to be different and thus, that can be used as an indication of location. To be able to refer to the virtual machine by name, a dynamic dns client service, dyndns is used. Ddclient, a client software for updating dns contents is used to implement this.

3.11 framework Design

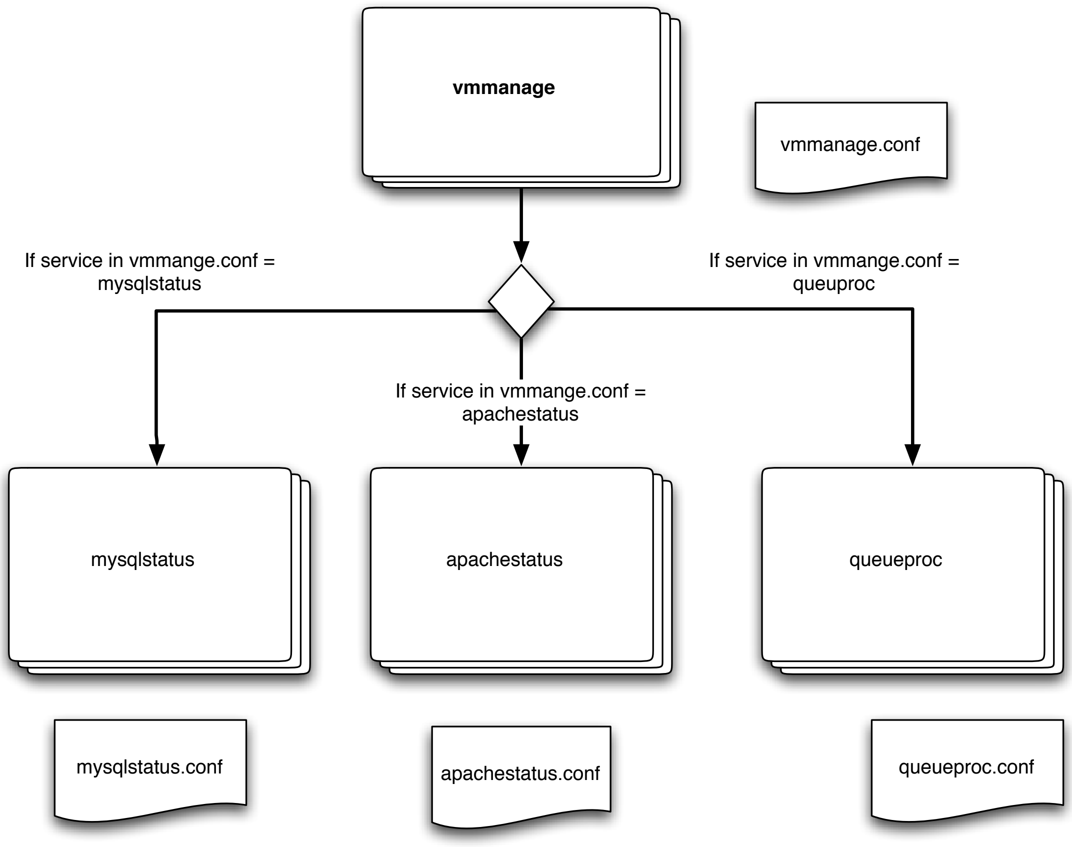

vmmanage
vmmanage.conf
If service in vmmange.conf = mysqlstatus
If service in vmmange.conf = queuproc
If service in vmmange.conf = apachestatus
mysqlstatus
apachestatus
queueproc
mysqlstatus.conf
apachestatus.conf
queueproc.conf

Figure 3.10: Framework Design

Figure 3.10 shows a pictorial view of the general design of implementation scripts used in the scenarios. vmmanage is the main script that is started when the virtual machine starts. It is stated by initd/initrc scripts so it runs as a daemon. vmmanage serves as the program through which monitoring scripts request for changes. Based on the content of the vmmanage.conf file, a specific policy is run. This makes the design extensible, allowing the addition of different kinds of policies as one pleases.

3.12 Implementation

The Four sample policies were implemented with perl programming. An generic architecture for implementation was designed with allows extensibility for other kinds of servers apart from those chosen (apache and MySQL) or even different performance indicators to be used conveniently. The monitoring and decision making was detached from the implementation of logic. A configuration file specifying constraints for decision making were used. This make change of constraints to reflect policy changes simple to do. All the scripts were developed on one system and several parts of it tested. The initial virtual machine used was save as a template in mln. This was in line with the initial desire to have the same specifications for all the virtual machines.

The mln file for building the virtual machines is shown in listings 3.7. The project name changed for the building of each virtual machine. This was created to match the hostname of the virtual machine; it was nii1 for building nii1.dyndns.org and so on. The template keyword is used to point to the initial file system from which the project is to be built from; this was the same for all the virtual machines. A dyndns plugin was used to automatically update dns records on the completion of building and startup of the virtual machines.

```
global {
        $proj = ??????
        project $[proj]
        project_password ********
}

host $[proj] {
     xen
     iscsi sanserver
     template nii.ext3
     network eth0 {
              address dhcp
     }
     memory 512M
     service_host mln1.vlab.iu.hio.no
     dyndns {
             hostname $[proj].dyndns.org
             user ******
             password  ******
     }
}
```

Listing 3.7: mln file used in creating virtual machines for scenarios

The linux kernel version used on all the virtual machines was Linux 2.6.18-6-xen-686. The version of operating system used is Debian GNU/Linux 4.0

3.12.1 Scenario 1

The first scenario, scenario 1, is a simple test of constraints in monitoring values serving as a trigger for change in location of virtual machine. This scenario makes use of policy 1 alone but runs two services, a mysql database server on nii1.dyndns.org and a webserver, nii2.dyndns.org. The purpose of this scenarios is to serve as a proof of concept of all the other scenarios since policy 1 (the threshold-aware) is the most basic concept and present in all the other policies and to test policy constraint manipulation through simple configuration file.

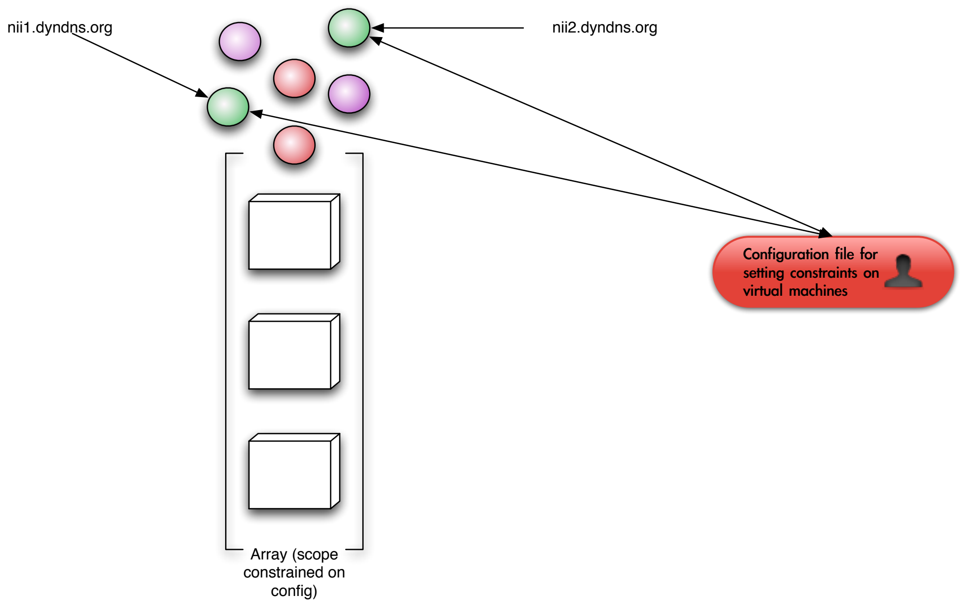

nii1.dyndns.org
nii2.dyndns.org
Configuration file for setting constraints on virtual machines
Array (scope constrained on config)

Figure 3.11: Scenario1

In this scenario, only two servers are made to run; a mysql database server and an apache web server. The two servers have hostnames nii1.dyndns.org and nii2.dyndns.org respectively. The web server is receiving requests from a script to serve a page that is composed purely of text and 664830 bytes in size.

A database server and web server are used to show that this would work with different kinds of services. The important thing is choosing the correct performance indicator which is already discussed in 3.9.3

3.12.2 Scenario 2

In the second scenario policy two and three, the base-aware and the schedule-aware policy are implemented. To simplify the experiment, only database servers are used for demonstration. The virtual machine running policy2 is nii3.dyndns.org and policy3 is implemented on nii5.dyndns.org.

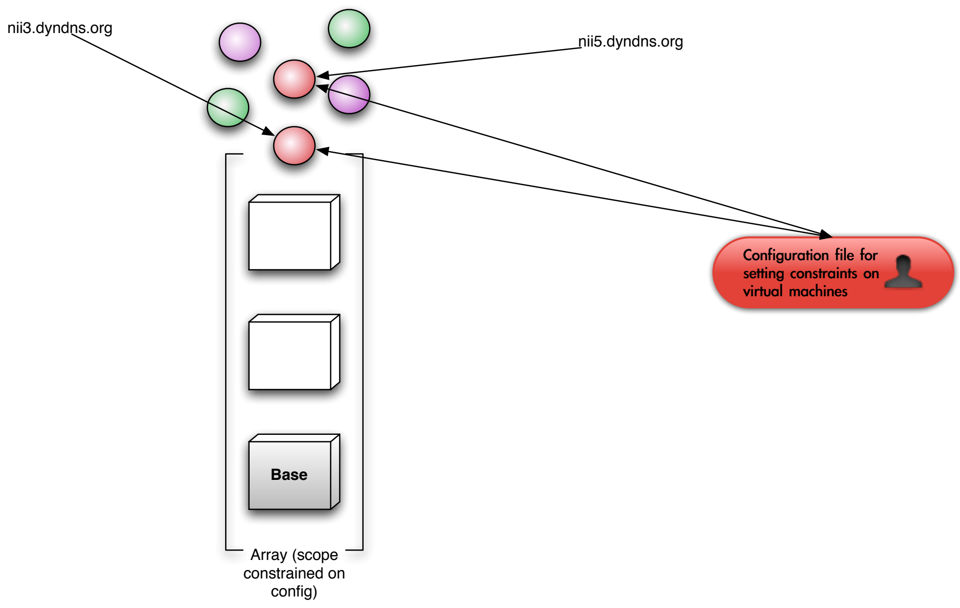

nii3.dyndns.org
nii5.dyndns.org
Configuration file for setting constraints on virtual machines
Base
Array (scope constrained on config)

Figure 3.12: Scenario2

nii3.dyndns.org is configured to run in a much similar was as nii1.dyndns.org except that it has two different thresholds, a low threshold and a high threshold. It has as part of it scope arrays within which to migrate to, a base host. The base is part of the array. nii5.dyndns.org has in addition to the parameters of nii3.dyndns.org, a working period. Migration is allowed only during this working period otherwise the virtual machine should relocate to the base host. These thresholds are defined in a configuration file, thus policy can be dictated purely through the configuration file. The configuration file for nii5.dyndns.org is shown in listings 3.8.

```
#configuration file for mysqlstatus - vmmanage

 daemon_interval=60
 threshold=50
 tarry=3
 lowthreshold=1
 lowtarry=20
 workstart=108
 workstop=170
```

Listing 3.8: configuration file for mysqlstatus for nii5.dyndns.org

3.12.3 Scenario 3

The third scenario is an implementation of policy 4. This scenario is the most interesting for this concept. It is demonstrates the benefit that arises from the paradigm being investigated in this paper; managing resources across different platforms in a similar manner and with the same interface by setting constraints and triggers for migration. The second array used is the amazon web services cloud. This implementation makes use of some of the advantages of cloud computing and the way it is designed to be have a flexible billing scheme.
A number of processes had to be setup to make it possible to migrate the virtual machine to the amazon cloud. Most of these steps are specific to the amazon cloud and mln. Details of making a virtual machine ready for the amazon cloud are described in great detail in [11]. The steps include storing certificates and user IDs for security and identification purposes. Dyndns played an important role in being able to reach the server irrespective of which array it was located in.

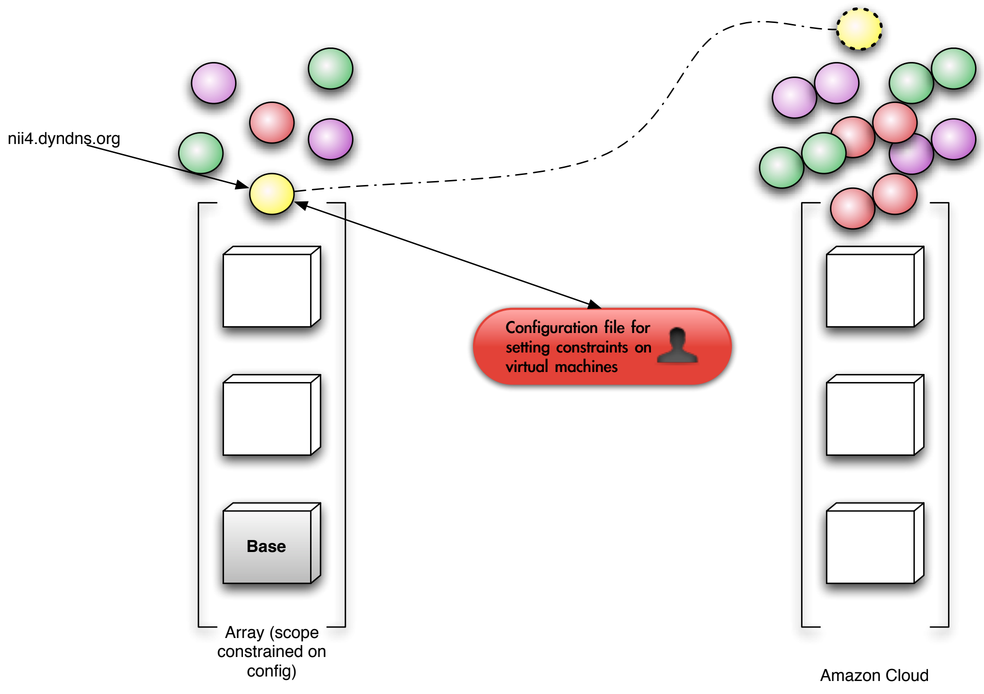

nii4.dyndns.org
Configuration file for setting constraints on virtual machines
Base
Array (scope constrained on config)
Amazon Cloud

Figure 3.13: Scenario3

nii4.dyndns.org is setup to read a list of jobs from a webpage and process them. Jobs are registered or claimed as a local queue and then processed locally on the system. The acquisition of the jobs in the queue had to be made by pulling from the list on the page. If jobs are pushed directly to the server, some jobs would be lost because of the downtime needed to do migration. The policy idea behind the policy for triggering motion is that certain jobs take a very long time to complete and migrating to a different site although more costly would ensure that the jobs are completed in a much shorter time. For the sake of demonstration, tags were manually added to these jobs to identify them. A threshold was set by a configuration file to dictate the number of "big jobs", which show jobs with a particular tag, that satisfy the condition for migration which is described in equation 3.5. The jobs used do not necessarily satisfy that condition and the output is not sent to any location since that was not the objective of the experiment. The configuration file used for setting policy boundaries is shown in listing 3.9. The configuration file means that the service would check the main queue for any jobs every 120 seconds (2 minutes) when there is not local queue being processed. It set the "big jobs" to be those of type grade a and causes the virtual machine to migrate when there are 2 or more jobs of this grade. The compete program for making this decision and processing is shown in listing D.1 in appendix D

```
#configuration file for queue processing - vmmanage
daemon_interval=120
threshold=2
bigjobs=grade a
```

Listing 3.9: configuration file for queue processing: queueproc.conf

For this policy to work in this implementation, output of the jobs must not be stored locally, especially when the virtual machine is away from the home array. The reason for not storing the jobs is that, the amazon web services does not offer persistent storage on virtual machines. If anything is stored on the disk of the virtual machine, it would be lost when it is shutdown on its way from the amazon cloud back to the home array. Changes made to the virtual machine when it is on the local array, however, are persistent and remain permanent even after migration.

Location of the virtual machine is acquired by acquiring its broadcast address, this is compared with the value of the broadcast address of the home array which is specified in a configuration file. A request to "move from home" or otherwise is made based on the location and values set triggering such action. The section of code responsible for this is shown in listings 3.10.

```
if ( $action eq "move_to_home" )
        {
                if ( $myHomeBroadcast = getCurrentBroadcast() )
                {
                `$logstatus "Already in home array"`;
                }
                else
                {
                my $newloc = $home;
                `$logstatus "Now using $hostname.$newloc.mln --moving from home"`;
                `mln client -f /root/.vmmanage/$hostname.$newloc.mln -S -c upgrade\
                                -h huldra.vlab.iu.hio.no`;
                }
        }
```

Listing 3.10: nii4.dyndns.org code decision code to request for move to home

In the implementation, a queue was represented by a simple text file which was readable via a web url. Access to change the filename to represent a receipt of the main queue into the local queue is granted through passwordless ssh. Renaming the file (queue) from jobs to nii4jobs represented this receipt. nii3.dyndns.org was used as the web server hosting the queues. A sample of the queue processed by the server is shown in listing 3.11. The page consists of the category or grade of job and a calculation that needs to be done. This calculation represents the job to be processed. The server iterates through the list, a single line at a time and performs the calculations (processes the jobs). An exhaustion of the list represents a completion of the local queue and the file is deleted at that point. A new local queue is then obtained and the process continues as before. If there is no new queue (an empty main queue), the process pauses for an interval and rechecks. Rechecking continues until a queue is present.

```
grade b = 123^123
grade b = 344^12
grade b = 12^2
grade b = 2^1
grade a = 1^2
:
:
```

Listing 3.11: Sample page to represent queue: taken from http://nii3.dyndns.org/nii4jobs

When the server comes up after a migration, it is important for it to check its local queue before continuing to receive a new queue from the main. One limitation that occurred because of the non-persistent nature of storage in the amazon cloud was the storage of log files. Because all changes to the file system done when the server is away from home all logging was sent to mainsyslog.dyndns.org.

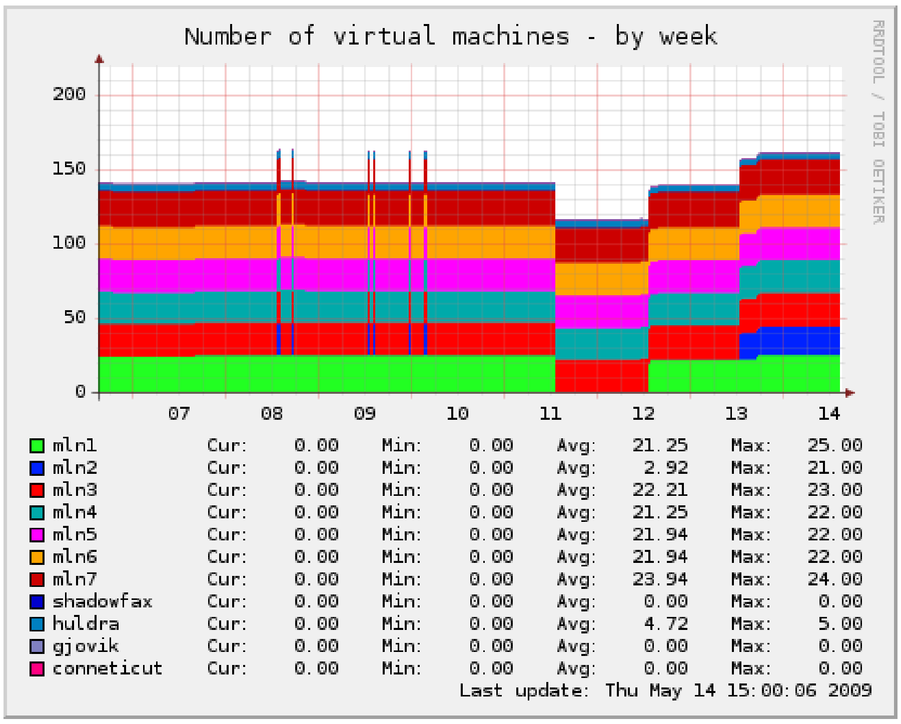
Chapter 4

Results

In this chapter, the results obtained from the various scenarios run are presented. A presentation of logs and graphs are made here.

4.1 Environment

Figure 4.1 gives an overview of the virtual infrastructure used in the experimental setup. The graph shows the number of virtual machines residing on the various hosts in the array.



🟩	mln1	Cur:	0.00	Min:	0.00	Avg:	21.25	Max:	25.00
🟦	mln2	Cur:	0.00	Min:	0.00	Avg:	2.92	Max:	21.00
🟥	mln3	Cur:	0.00	Min:	0.00	Avg:	22.21	Max:	23.00
🟦	mln4	Cur:	0.00	Min:	0.00	Avg:	21.25	Max:	22.00
🟪	mln5	Cur:	0.00	Min:	0.00	Avg:	21.94	Max:	22.00
🟧	mln6	Cur:	0.00	Min:	0.00	Avg:	21.94	Max:	22.00
🟥	mln7	Cur:	0.00	Min:	0.00	Avg:	23.94	Max:	24.00
🟦	shadowfax	Cur:	0.00	Min:	0.00	Avg:	0.00	Max:	0.00
🟦	huldra	Cur:	0.00	Min:	0.00	Avg:	4.72	Max:	5.00
🟪	gjovik	Cur:	0.00	Min:	0.00	Avg:	0.00	Max:	0.00
🟥	conneticut	Cur:	0.00	Min:	0.00	Avg:	0.00	Max:	0.00

Last update: Thu May 14 15:00:06 2009

Figure 4.1: Overview of hosts in array

4.2 Scenario 1

A graph of the number of slow queries on nii1.dyndns.org is displayed in 4.2. The graph shows a cycle which is rather sinusoidal as expected.

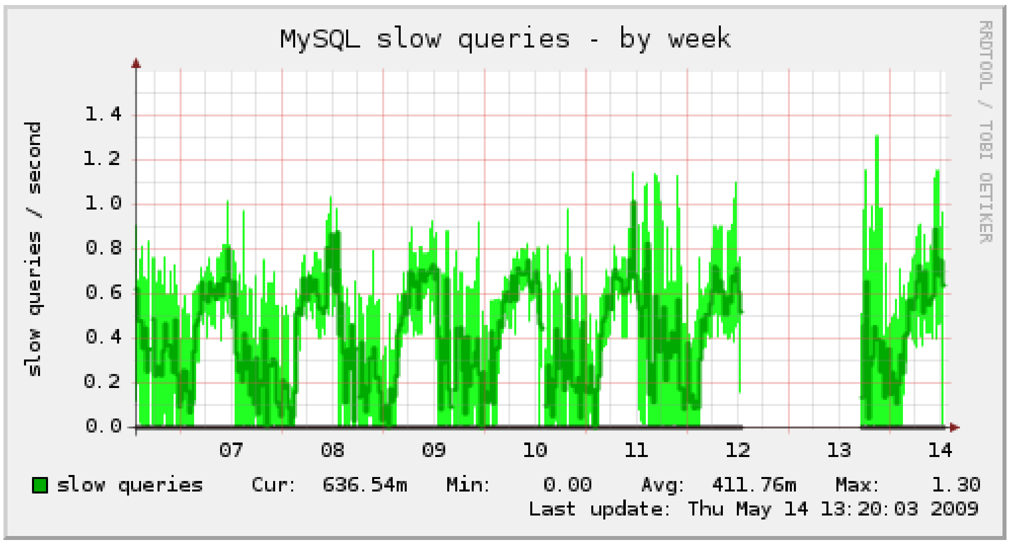


Figure 4.2: Slow queries for a week on nii1.dyndns.org

Listings 4.1 shows logs of requests of migration that originated from nii1.dydns.org. The logs show a time-stamp, first is syslog format and then in human readable form, the hostname from which the request or information is originating from, and the information being sent from the host.

```
1241613863 Wed May 6 14:44:23 CEST 2009 nii1: Requesting migration of VM: just moving
1241613863 Wed May 6 14:44:23 CEST 2009 nii1: Now using nii1.1.mln
1241614090 Wed May 6 14:48:10 CEST 2009 nii1: Requesting migration of VM: just moving
1241614090 Wed May 6 14:48:10 CEST 2009 nii1: Now using nii1.2.mln
1241614315 Wed May 6 14:51:55 CEST 2009 nii1: Requesting migration of VM: just moving
1241614315 Wed May 6 14:51:55 CEST 2009 nii1: Now using nii1.3.mln
1241614545 Wed May 6 14:55:45 CEST 2009 nii1: Requesting migration of VM: just moving
1241614545 Wed May 6 14:55:45 CEST 2009 nii1: Now using nii1.4.mln
1241614771 Wed May 6 14:59:31 CEST 2009 nii1: Requesting migration of VM: just moving
1241614771 Wed May 6 14:59:31 CEST 2009 nii1: Now using nii1.5.mln
1241615290 Wed May 6 15:08:10 CEST 2009 nii1: Requesting migration of VM: just moving
1241615290 Wed May 6 15:08:10 CEST 2009 nii1: Now using nii1.1.mln
1241615516 Wed May 6 15:11:56 CEST 2009 nii1: Requesting migration of VM: just moving
1241615516 Wed May 6 15:11:56 CEST 2009 nii1: Now using nii1.2.mln
1241615765 Wed May 6 15:16:05 CEST 2009 nii1: Requesting migration of VM: just moving
1241615765 Wed May 6 15:16:05 CEST 2009 nii1: Now using nii1.3.mln
1241615990 Wed May 6 15:19:50 CEST 2009 nii1: Requesting migration of VM: just moving
1241615991 Wed May 6 15:19:51 CEST 2009 nii1: Now using nii1.4.mln
1241616218 Wed May 6 15:23:38 CEST 2009 nii1: Requesting migration of VM: just moving
1241616218 Wed May 6 15:23:38 CEST 2009 nii1: Now using nii1.5.mln
1241616443 Wed May 6 15:27:23 CEST 2009 nii1: Requesting migration of VM: just moving
.
.
.
.
.
```

Listing 4.1: nii1.dyndns.org logs of requests for migration (on mainsyslog.dyndns.org)

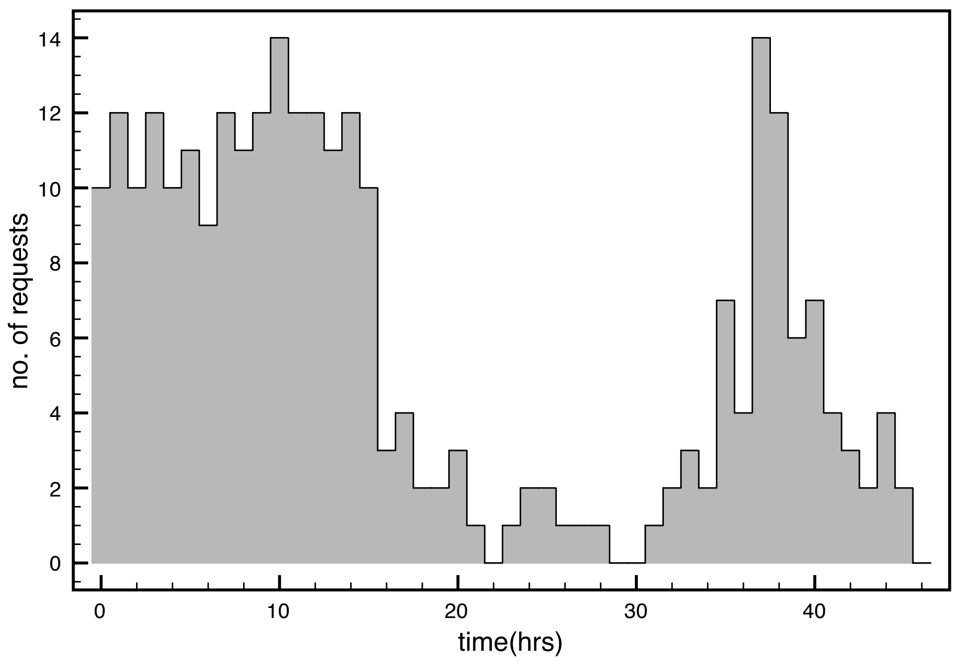


Figure 4.3: Activity graph of nii1: showing requests for migration per hour over 48 hours

Figure 4.3 shows the frequency of requests per hour over a period of 48 hours. Within this period the threshold value was changed from 20 to 50. This change was made in the 16th hour.

Listing 4.2 shows the parts of logs created by the decision making script and monitoring of mysql. Every cycle of monitoring consists of 4 lines. The first shows the current recorded value of slow queries, the second is a time recording, third the count of slow queries above the threshold and the last line in the cycle shows the current threshold as indicted in the configuration file for the script.

```
:
:
:
slow rate:      0
time:           Wed May 13 17:16:07 CEST 2009
count:          0
threshold:      50
**------------------------------------**
slow rate:      0
time:           Wed May 13 17:17:07 CEST 2009
count:          0
threshold:      50
**------------------------------------**
slow rate:      0
time:           Wed May 13 17:18:07 CEST 2009
count:          0
threshold:      50
**------------------------------------**
slow rate:      0
time:           Wed May 13 17:19:07 CEST 2009
count:          0
threshold:      50
**------------------------------------**
```

```
slow rate:       0
time:            Wed May 13 17:20:07 CEST 2009
count:           0
threshold:       50
**--------------------------------**
slow rate:       0
time:            Wed May 13 17:21:07 CEST 2009
count:           0
threshold:       50
**--------------------------------**
slow rate:       0
time:            Wed May 13 17:22:07 CEST 2009
count:           0
threshold:       50
**--------------------------------**
slow rate:       0
time:            Wed May 13 17:23:07 CEST 2009
count:           0
threshold:       50
**--------------------------------**
slow rate:       0
time:            Wed May 13 17:24:07 CEST 2009
count:           0
threshold:       50
**--------------------------------**
```

Listing 4.2: content of vmmanage.log on nii1.dyndns.org

4.3 Scenario 2

Parts of logs with interesting events obtained in scenario 2 are shown here.

As in listings 4.2 of scenario 1, listings 4.3 contains a number of lines for each cycle of monitoring. Unlike that for scenario 1, this contains 6 lines. In addition to the slow rate, time, count and threshold, which now represents the upper threshold, the value of the lower threshold is show as well and the count of values below the low threshold.

```
:
:
slow rate:       0
time:            Wed May 13 18:00:57 CEST 2009
count:           0
threshold:       50
lowcount:            1
lowthreshold:        1
**-----------------------------------**
slow rate:       0
time:            Wed May 13 18:01:57 CEST 2009
count:           0
threshold:       50
lowcount:            2
lowthreshold:        1
**-----------------------------------**
slow rate:       0
time:            Wed May 13 18:02:57 CEST 2009
count:           0
threshold:       50
lowcount:            3
lowthreshold:        1
**-----------------------------------**
slow rate:       0
time:            Wed May 13 18:03:57 CEST 2009
count:           0
threshold:       50
lowcount:            4
lowthreshold:        1
**-----------------------------------**
slow rate:       0
time:            Wed May 13 18:04:57 CEST 2009
count:           0
threshold:       50
lowcount:            5
lowthreshold:        1
**-----------------------------------**
slow rate:       0
time:            Wed May 13 18:06:02 CEST 2009
count:           0
threshold:       50
lowcount:            6
lowthreshold:        1
**-----------------------------------**
slow rate:       76
time:            Wed May 13 18:07:02 CEST 2009
count:           1
threshold:       50
lowcount:            0
lowthreshold:        1
**-----------------------------------**
:
:
```

Listing 4.3: content of vmmanage.log on nii3.dyndns.org

Figure 4.4 shows the number of requests for migration made from nii3.dyndns.org per hour.

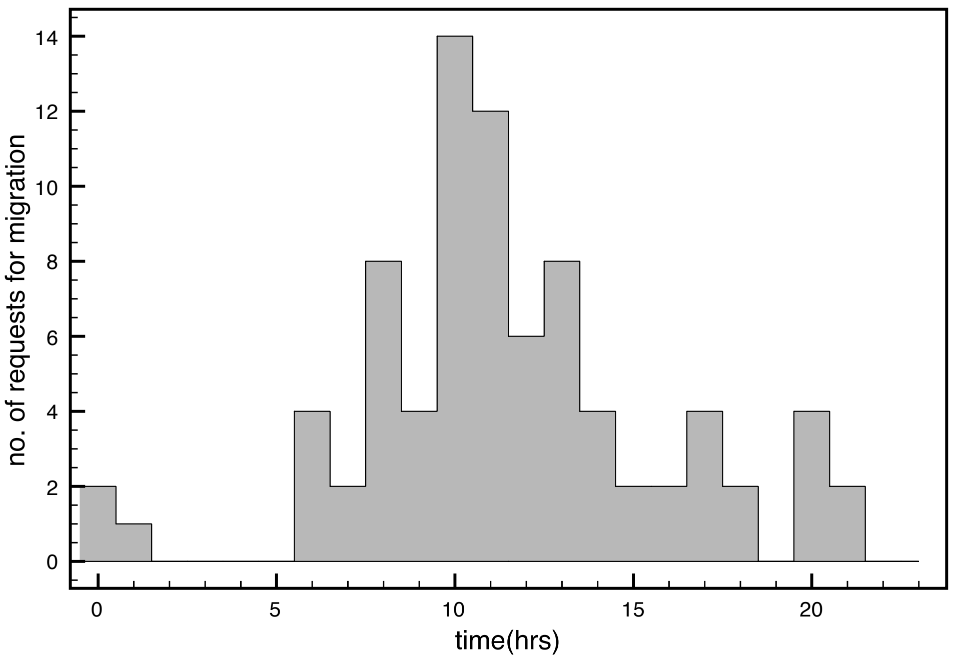


Figure 4.4: Activity graph of nii3: showing requests for migration per hour

Listings 4.4 shows logs of requests of migration that originated from nii3.dydns.org. The logs show a time-stamp, first is syslog format and then in human readable form, the hostname from which the request or information is originating from, and the information being sent from the host.

```
1242230527 Wed May 13 18:02:07 CEST 2009 nii3: Starting VMManage
1242232480 Wed May 13 18:34:40 CEST 2009 nii3: Requesting migration of VM: just moving
1242232480 Wed May 13 18:34:40 CEST 2009 nii3: Now using nii3.2.mln
1242244517 Wed May 13 21:55:17 CEST 2009 nii3: Requesting migration of VM: just moving
1242244517 Wed May 13 21:55:17 CEST 2009 nii3: Now using nii3.3.mln
1242248094 Wed May 13 22:54:54 CEST 2009 nii3: Requesting migration of VM: just moving
1242248094 Wed May 13 22:54:54 CEST 2009 nii3: Now using nii3.4.mln
1242248416 Wed May 13 23:00:16 CEST 2009 nii3: Requesting migration of VM: just moving
1242248416 Wed May 13 23:00:16 CEST 2009 nii3: Now using nii3.1.mln
1242249851 Wed May 13 23:24:11 CEST 2009 nii3: Now using nii3..mln
1242253388 Thu May 14 00:23:08 CEST 2009 nii3: Now using nii3..mln
1242255539 Thu May 14 00:58:59 CEST 2009 nii3: Now using nii3..mln
1242257634 Thu May 14 01:33:54 CEST 2009 nii3: Now using nii3..mln
1242271476 Thu May 14 05:24:36 CEST 2009 nii3: Requesting migration of VM: just moving
1242271476 Thu May 14 05:24:36 CEST 2009 nii3: Now using nii3.2.mln
1242271772 Thu May 14 05:29:32 CEST 2009 nii3: Requesting migration of VM: just moving
1242271772 Thu May 14 05:29:32 CEST 2009 nii3: Now using nii3.3.mln
1242272368 Thu May 14 05:39:28 CEST 2009 nii3: Requesting migration of VM: just moving
1242272368 Thu May 14 05:39:28 CEST 2009 nii3: Now using nii3.4.mln
1242278697 Thu May 14 07:24:57 CEST 2009 nii3: Requesting migration of VM: just moving
1242278697 Thu May 14 07:24:57 CEST 2009 nii3: Now using nii3.1.mln
1242280196 Thu May 14 07:49:56 CEST 2009 nii3: Requesting migration of VM: just moving
1242280196 Thu May 14 07:49:56 CEST 2009 nii3: Now using nii3.2.mln
1242280498 Thu May 14 07:54:58 CEST 2009 nii3: Requesting migration of VM: just moving
1242280498 Thu May 14 07:54:58 CEST 2009 nii3: Now using nii3.3.mln
1242280788 Thu May 14 07:59:48 CEST 2009 nii3: Requesting migration of VM: just moving
```

```
1242280788 Thu May 14 07:59:48 CEST 2009 nii3: Now using nii3.4.mln
1242282869 Thu May 14 08:34:29 CEST 2009 nii3: Requesting migration of VM: just moving
1242282869 Thu May 14 08:34:29 CEST 2009 nii3: Now using nii3.1.mln
1242285586 Thu May 14 09:19:46 CEST 2009 nii3: Requesting migration of VM: just moving
1242285586 Thu May 14 09:19:46 CEST 2009 nii3: Now using nii3.2.mln
1242285877 Thu May 14 09:24:37 CEST 2009 nii3: Requesting migration of VM: just moving
1242285877 Thu May 14 09:24:37 CEST 2009 nii3: Now using nii3.3.mln
1242286184 Thu May 14 09:29:44 CEST 2009 nii3: Requesting migration of VM: just moving
1242286184 Thu May 14 09:29:44 CEST 2009 nii3: Now using nii3.4.mln
1242287406 Thu May 14 09:50:06 CEST 2009 nii3: Requesting migration of VM: just moving
1242287406 Thu May 14 09:50:06 CEST 2009 nii3: Now using nii3.1.mln
1242288890 Thu May 14 10:14:50 CEST 2009 nii3: Requesting migration of VM: just moving
1242288890 Thu May 14 10:14:50 CEST 2009 nii3: Now using nii3.2.mln
1242289191 Thu May 14 10:19:51 CEST 2009 nii3: Requesting migration of VM: just moving
1242289191 Thu May 14 10:19:51 CEST 2009 nii3: Now using nii3.3.mln
1242290407 Thu May 14 10:40:07 CEST 2009 nii3: Requesting migration of VM: just moving
1242290407 Thu May 14 10:40:07 CEST 2009 nii3: Now using nii3.4.mln
1242291002 Thu May 14 10:50:02 CEST 2009 nii3: Requesting migration of VM: just moving
1242291002 Thu May 14 10:50:02 CEST 2009 nii3: Now using nii3.1.mln
1242291234 Thu May 14 10:53:54 CEST 2009 nii3: Requesting migration of VM: just moving
1242291234 Thu May 14 10:53:54 CEST 2009 nii3: Now using nii3.2.mln
```

Listing 4.4: nii3.dyndns.org logs of requests for migration (on mainsyslog.dyndns.org)

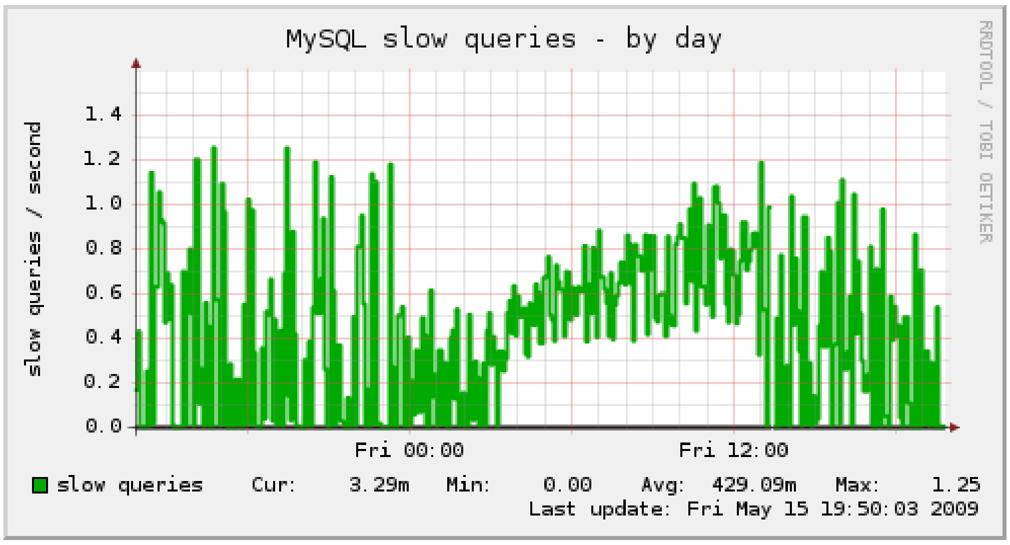


Figure 4.5: Slow queries for a day on nii3.dyndns.org

Figure **??** shows a histogram the requests for migration made from nii5.dyndns.org each hour. The working hours of nii5.dyndns.org was between 9:00 and 16:00. It can be observed that there is no activity within this period.

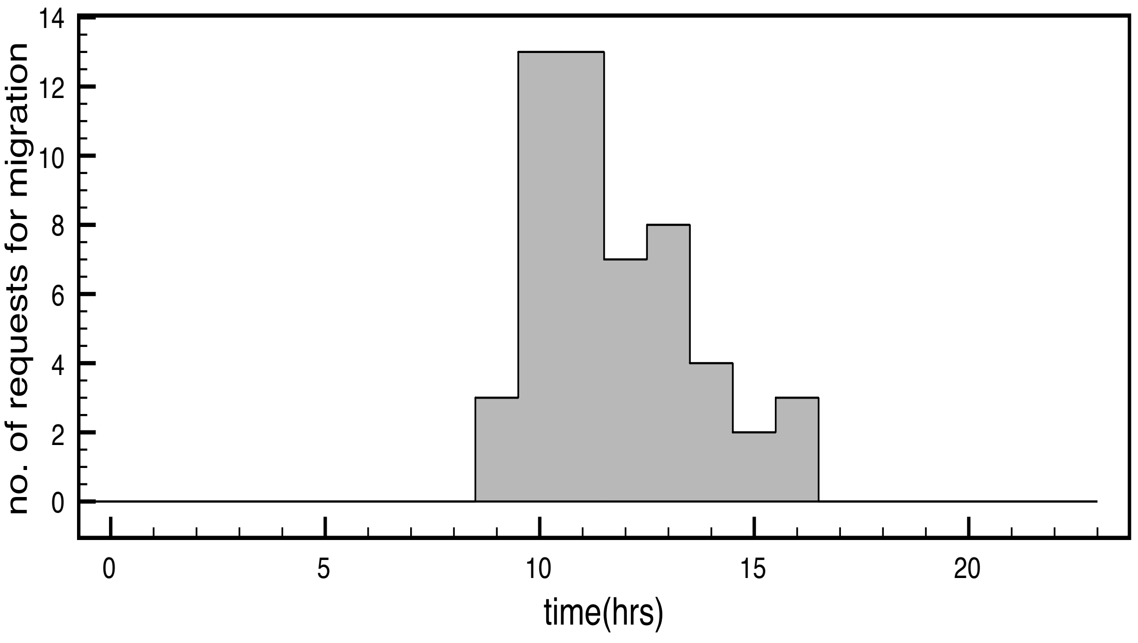


Figure 4.6: Activity graph of nii5: showing requests for migration per hour

4.4 Scenario 3

Logs obtained from scenario 3 are shown in this section.

```
:
:
1242584629 Sun May 17 20:23:49 CEST 2009 nii4: Already out of home array
1242584629 Sun May 17 20:23:49 CEST 2009 nii4: Would process: away from home now
1242584696 Sun May 17 20:24:56 CEST 2009 nii4: Already out of home array
1242584696 Sun May 17 20:24:56 CEST 2009 nii4: Would process: away from home now
1242584870 Sun May 17 20:27:50 CEST 2009 nii4: Now using nii4.1.mln --moving home
1242584870 Sun May 17 20:27:50 CEST 2009 nii4: Would process queue after move to home
1242585444 Sun May 17 20:37:24 CEST 2009 nii4: Starting VMManage
1242585449 Sun May 17 20:37:29 CEST 2009 nii4: Already in home array
1242585449 Sun May 17 20:37:29 CEST 2009 nii4: Would process: at home now
1242588940 Sun May 17 21:35:40 CEST 2009 nii4: Already in home array
1242588940 Sun May 17 21:35:40 CEST 2009 nii4: Would process: at home now
1242589046 Sun May 17 21:37:26 CEST 2009 nii4: Already in home array
1242589046 Sun May 17 21:37:26 CEST 2009 nii4: Would process: at home now
1242589412 Sun May 17 21:43:32 CEST 2009 nii4: Already in home array
1242589412 Sun May 17 21:43:32 CEST 2009 nii4: Would process: at home now
1242590118 Sun May 17 21:55:18 CEST 2009 nii4: Now using nii4.2.mln --moving from home
1242590119 Sun May 17 21:55:19 CEST 2009 nii4: Would process queue after move from home
1242591443 Sun May 17 22:17:23 CEST 2009 nii4: Starting VMManage
1242591446 Sun May 17 22:17:26 CEST 2009 nii4: Already out of home array
1242591446 Sun May 17 22:17:26 CEST 2009 nii4: Would process: away from home now
1242592852 Sun May 17 22:40:52 CEST 2009 nii4: Now using nii4.1.mln --moving home
1242592853 Sun May 17 22:40:53 CEST 2009 nii4: Would process queue after move to home
1242593061 Sun May 17 22:44:21 CEST 2009 nii4: Starting VMManage
1242593061 Sun May 17 22:44:21 CEST 2009 nii4: Already in home array
1242593062 Sun May 17 22:44:22 CEST 2009 nii4: Would process: at home now
:
:
```

Listing 4.5: nii4.dyndns.org logs (on mainsyslog.dyndns.org)

Listings 4.5 shows part of the logs sent to mainsyslog.dyndns.org from nii4.dyndns.org during the run of scenario 3 whilst processing different kinds of queues to trigger different kinds of behaviours. The section of log shows the successful migration of the server to and from the home array. The log contains a time stamp and a message of the action being performed or to be performed.
During the period in question the jobs file, accessible through http://nii3.dyndns.org/jobs, was updated with new content to trigger various reactions. Every occurrence of a line containing the text “Would process queue after migration” signifies a point where migration has been requested. The phrase “Starting VMManage” indicates a service start and that follows after the system has successfully migrated and booted up. It is therefore possible to see how long it took to do a migration between the arrays.

```
Filesystem            1K-blocks       Used  Available  Use% Mounted on
/dev/hda1               4959726    1015417    3739509   22% /
tmpfs                     65608          0      65608    0% /lib/init/rw
udev                      10240         16      10224    1% /dev
```

Listing 4.6: df command on nii4.dyndns.org)

Listings 4.6 shows the result if the df command when run in nii4.dyndns.org. This shows the size of the virtual machine. The time taken to migrate out of the home array is related to the size of the virtual machine.

Chapter 5

Discussion

This chapter makes a discussion and analysis of the results presented in chapter 4. Implications of choices made and possible improvements are discussed.

5.1 Review of Approach

In making a comparison between this approach and other possible approaches that have been mentioned in section 1.3 as possible options that could have been used, one realizes the lack of control in triggering interesting conditions and replaying them in the environment. This would have been more suited if simulation was used. However, the experiment and its unpredictability reflect what happens in real systems. Some possibilities could have easily been overlooked if simulation was used. Not using simulation meant it was not possible to play back scenarios of interest, however. That would have been an advantage to have in order to be able to explicitly say if certain behaviours were just random or otherwise.

5.2 Missing Parts

In the implementation of scenario 2 and scenario 3, it became clear that virtual machines were not necessarily migrating to a host with more resources available as the policy was meant to do. This was because of the lack of a framework that would do re-prioritization of hosts to reflect the current state of host resource availability. This is one major advantage that having the decision making being made on the infrastructure level has over making decision from within the virtual machine. What is missing in this implementation is a framework that re-orders the array to reflect the changing conditions of available resources that occur on the hosts.

The implementation was on purpose chosen to be service determined or oriented. This has to it some advantages as well as some downsides. The monitoring of services for the maintenance of a desired quality of service (QoS) is what is ultimately desired by any service provider. When using conventional approaches, the overall performance of the virtual machine is what is typically used to trigger operations logic. It is possible for memory for example, on a virtual machine to go high and trigger a migration of whatever operations logic is defined without the service being offered by the virtual machine is performing as desired.

Monitoring service from within the virtual machine itself has some disadvantages. It does not necessarily reflect the experience of the users of the service since it does not take network performance into consideration. However, there is an advantage to that in helping system administrators troubleshoot issues with services. If external monitoring indicates a drop in performance and local monitoring indicates otherwise, one gets to eliminate certain parts of the system just by comparison.

5.3 Scenario 1

Scenario 1 proved that, it is a challenge to choose and tune performance indicators more so with some services more than others. Whereas a lot of time was spent on tuning and adjusting thresholds, this would have already been defined in the policy specification based on prior knowledge. Scenario 1 offered a few interesting moments that showed possible cases which had been discussed earlier and others which were not condsidered.

For the most part, the behaviour of the virtual machines were as expected. The machines migrated when load was too high on them and thus when performance low. This was the basic idea behind the policy design as design was based purely on a single threshold which was to prevent a performance from going below a specified value.

The hypothetical case of a virtual machine going into an endless migration loop was achieved in the former part of scenario 1. This was predominantly due to bad threshold values set in the configuration file. It thus shows that in addition to incomplete knowledge of the infrastructure, bad threshold values can cause such a situation to occur.

A moment which reflects a flaw in implementation and is linked to discussion made earlier occurred in scenario 1. Mln server stopped running on mln1 and so the servers could not migrate although they were sending information to mainsyslog.dyndns.org that they were requesting for migration. On the 12th of the month, mln1 was restarted and thus the servers went off. There is no way to confirm whether migration was successful or not by looking at the log files; they only reflect requests made to the mln server for live migration to be made. The only way of confirming migration from this design is to get access to the hosts in the array. In a multi-tenant environment however, one does not have access of control over these.

Although this situation could not have been prevented in conventional systems unless high availability services are running, it brings to the fore a hypothetical case which had been overlooked. What happens if the hosting provider wants to take down a host for maintenance or replacement. As mentioned in section 2.3.2, one of the benefits of having virtualization is to be able to take down hosts without taking down services, on the other hand, Each host becomes more important because its failure can cause many services to fail. It is therefore necessary that the programs implementing policy be able to receive some kind of information about such changes in infrastructure. In the absence of this the flexility gained through virtualization on the part of the hosting provider is taken away.

Figure 4.3 shows how activity and thus behaviour can be influenced or controlled by simply changing values in the configuration file. The period prior to changing the value of the threshold value in the configuration file from 20 to 50 had a lot of activity (requests per hour) which caused the virtual machine to go close to a state of an endless loop almost all the time.

5.4 Scenario 2

In scenario 2, having two threshold values and defining a base cause a very different behaviour from that observed in scenario 1. There were some similarities in the behaviour of the two virtual machines when they were both active. When it was not “working hours” for nii5.dyndns.org however, there was activity (migration) with nii3.dyndns.org.

Comparing the graphs from figure 4.5 to those from figure 4.4, one sees that there is a rather direct correlation between the requests for migration made by nii3.dyndns.org and the graph of slow queries obtained through munin. This was the intended behaviour of the policy implementation, in the absence of a host with enough available resources to provide the virtual machine with the resources needed to support the load.

During the peak hours migration requests went as high as 14 requests per hour. This meant the virtual machine did enter a state of endless migration during this period. As mentioned earlier, this is what happens if no host with adequate resources are found.

The histogram in figure 4.6, confirms the inactivity of nii5.dyndns.org at times which where out of its working hours(working hours was set between 9:00 and 16:00). At this point, the virtual machine is expected to migrate to the base to reduce costs and stay there until the next working hours start.

As with scenario 1, it was not possible to confirm actual migration from the graphs obtained on the overall status of the array because the array had so many virtual machines running and a migration of a few virtual machines did not cause any significant or noticeable change in the graphs. We assumed for the purpose of this experiment that each request for migration was successful.

5.5 Scenario 3

Running scenario 3, showed results of success for cross array migration in response to policy. The scenario did run as anticipated. The logs from mainsyslog.dydns.org in section 4.4 prove this. An ssh connection was made to the server to verify this as well.

The logs from listings 4.5, indicate that it takes that, the time taken to migrate T_M is sometimes up to 22 minutes as seen by looking at line 19 and line 20 (from 21:55:19 to 22:17:23). The migration from the amazon cloud to back to the array, however, takes a far shorter time than that. The time taken to migrate back back to the local array can be seen to be quite varied; it took 9:26 minutes for the first instance in the section of log shown on line 8 and line 9 (from 20:27:50 to 20:37:24) and in the second instance took only 3:32 minutes shown in line 24 and line 25 (from 22:40:53 to 22:44:21). Although the time spent to migrate back to the home array is not discussed in in any great detail prior to this point, it is important that this time should not be too much else it would mean only an accumulation of an unreceived queue whilst in the migration process.

In the setup process of this scenario (scenario 3), it was necessary to configure firewall rules for the amazon cloud. In migrating from array to array or site to site, there are concerns of security. Security concerns rise from network concerns due to change in network locations and firewall rules set and used by different sites. In addition to security concerns because of the network, there may be concerns arising from the virtual infrastructure or platform on which the virtual machine runs.

The thought of cross array migration makes even more profound, the need to set constraints and boundaries as to policy implementation to ensure that the location of virtual machine can be known and its behaviour controlled especially when things go wrong such as network connections break. Failover measures must be put in place and more intelligent systems established to know if it is even possible to implement such an action at a give point in time. How does a virtual machine know if a particular array is not reachable before it migrates? How does a virtual machine recover from a failure during migration. These are all questions that must be answered with by developing more intelligent frameworks.

5.6 General

Having virtual machine-side policy implementation programs, which interact both ways with a infrastructural setup would be a more preferred way to allowing decision making to be made with more information. Tracking location of virtual machines would also be more certain and simpler especially when the same array is being used. Scenario 2 and scenario 3, are cases that can be setup by conventional architectures of virtualization management if they are allowed to support service monitoring in addition system monitoring.

This approach of making decisions from the virtual machones, brings to the fore, some possibilities of of multi-platform utilization whilst still maintaining a very similar front to the system administrator. The benefit of being able to dictate and influence policy and thus behaviour through configuration files is one that would be very easy to manage since there exist already tested methods of configuration management and tools.

Chapter 6

Conclusion

With the aim of appealing to a system administration audience, this thesis has demonstrated by implementation, an alternative approach to virtual machine management and thus resource usage optimization with the aim of breaking the tradition of over-provisioning. The implementation has made use of principles of policy driven management by making the virtual machine a more autonomous and automated entity, to effect operations logic. Whilst certain kinds of operations logic functions can be implemented from the virtual machine side, it is clear that by itself, it is not enough for total implementation of desired behaviour and policies of especially those of availability. Having a operations logic being initiated or decided by the virtual machine itself, however, is an appealing concept that offers a wide range of opportunities. It brings to it the advantage of being able to transfer policy from one site to another because it is contained in the virtual machine itself. The findings of this paper also proves that it is possible, given more time and resources, to develop a more automated operations logic driven management for virtual machines that can work across multiple sites.

Looking back at the question asked in the problem statement, "What are the effects of shifting some operations logic responsibilities to the virtual machines as opposed to the conventional case of total responsibility lying with the virtualization management software?", the findings of this thesis show that information on the current state of the overall virtual infrastructure is not accessible when this is done, however, the benefit of tying policy to virtual machines irrespective of their location, an enormous advantage in reducing costs of operations, is gained. Although operations logic was limited to just migration and live migration, the success of the experiments indicate that this can successfully be controlled from the virtual machine side.

Future Work

Perhaps, a combined approach where infrastructural side decisions for availability are made and a virtual machine side for quality of service and cost savings are made would be the way forward. This would allow clients to control the costs and quality of service as well as allow hosting providers to maintain system availability. Creating Application programming interfaces (APIs) that allow programs running from within the virtual machine to send queries to gain information such as current location would be great.

With developments of towards creating a standardized virtual machine format (OVF), it would perhaps become possible one day to develop applications that would enable one to utilize virtualization technologies from different vendors seamlessly .An implementation of this nature would demand a rather huge amount of thinking and design since the nature of policies are very large and varied and unique to different installations. A framework which allows the description of operations logic and thus behaviour in a declarative manner would probably be the way forward.

Bibliography

[1] Andy Gordon and Karthkeyan Bhargavan. Getting operations logic right. *The rise and the rise of the declarative datacentre*, 2008.

[2] Marinos Charalambides, Paris Flegkas, George Pavlou, Arosha K Bandara, Emil C Lupu, Alessandra Russo, Naranker Dulay, Morris Sloman, and Javier Rubio-Loyola. Policy conflict analysis for quality of service management. Technical report, University of Surrey and Imperial College London and Universtat de Catalunya, 2005.

[3] Mark Burgess. *Principles of Network and System administration.* Wiley J and Sons, second edition edition, 2004.

[4] Paul Anderson. System configuration, volume 14 of short topics in system administration. Technical report, SAGE, 2006.

[5] Kyrre Matthias Begnum. *Towards autonomic management in system administration.* PhD thesis, Faculty of Mathematics and Natural Sciences, University of Oslo, 2008.

[6] Burt Kaliski. Multi-tenant cloud computing: From cruise liners to container ships. *Third AsiaPacific Trusted Infrastructure Technologies Conference*, 2008.

[7] Neal Leavitt. Is cloud computing really ready for prime time? *IEEE technology news*, 2009.

[8] Christina Hoffa, Gaurang Mehta, Timothy Freeman, Ewa Deelman, Kate Keahey, Bruce Berriman, and John Good. On the use of cloud computing for scientific workflows. *Fourth IEEE International Conference on eScience*, 2008.

[9] *Amazon Web Services: Overview of Security Processes*, 2008.

[10] Mln - manage large networks. http://mln.sourceforge.net/.

[11] Kyrre Begnum and John Sechrest. *MLN Manual version 1.0.1*, 2009.

[12] Distributed resource scheduler for dynamic resource scheduling of server resources - vmware. http://www.vmware.com/products/vi/vc/drs.html.

[13] Christopher Clark, Keir Fraser, Steven Hand, Jacob Gorm Hansen, Eric Jul, Christian Limpach, Ian Pratt, and Andrew Warfield. Live migration of virtual machines. In *Proceedings of the 2nd Symposium on Networked Systems Design and Implementation (NSDI '05).* University of Cambridge Computer Laboratory and Department of Compter Science, University of Copenhagen, Denmark, 2005.

[14] Mark Burgess. *Analytical network and system adminstration.* John Wiley and Sons, Ltd, 2004.

Appendix A

Common Scripts and Configuration Files

```
#! /bin/sh
#
# init script for the vm management program: vmmanage

PATH=/sbin:/bin:/usr/sbin:/usr/bin
DAEMON="/root/.vmmanage/vmmanage"
NAME="vmmanage"
DESC="vm managememt utility"

# Don't run if missing
test -f $DAEMON || exit 0

# Evaluate the config for the Debian scripts
run_daemon=true

if [ -f "/root/.vmmanage/vmmanage.conf" ]; then
  . /root/.vmmanage/vmmanage.conf
  echo "loaded conf"
fi

OPTIONS=""

set -e

echo "yipee"
case "$1" in
  start)
        if [ "$run_daemon" != "true" ]; then
                echo "run_daemon not true"
                exit 0
        fi
        echo -n "Starting $DESC: "
        start-stop-daemon --start --quiet --background \
                --make-pidfile --pidfile /var/run/$NAME.pid --exec $DAEMON
        echo "$NAME."
        ;;

  stop)
        if [ "$run_daemon" != "true" ]; then
                exit 0
        fi
        echo -n "Stopping $DESC: "
        start-stop-daemon --stop  \
                --pidfile /var/run/$NAME.pid --oknodo
        echo "$NAME."
        ;;
  restart)
        if [ $run_daemon != "true" ]; then
                exit 0
        fi
        echo -n "Restarting $DESC: "
        $0 stop >/dev/null
        $0 start >/dev/null
        echo "$NAME."
        ;;

  *)
        N=/etc/init.d/$NAME
        echo "Usage: $N {start|stop|restart}" >&2
        exit 1
        ;;
esac

exit 0
```

Listing A.1: Initd script to daemonize vmmanage

```
#!/usr/bin/perl
#!/usr/local/bin/perl
######################################################################
#
# vmmanage - a Perl client for autonomic management of VMs
#
# Author: Nii Apleh Lartey
######################################################################

require 5.004;
#use strict;
no strict "vars";

my $locationfile = "/var/run/vmmanage_location";
my $logstatus = "/root/.vmmanage/logstatus.pl";
my $action = $ARGV[0];
my $empty = "";
my $hostname = `cat /etc/hostname`;
chomp $hostname;
#read from configuration file

my %config;
open my $config, '<', '/root/.vmmanage/vmmanage.conf' or die $!;
while(<$config>)
{
        chomp;                  # no newline
        s/#.*//;                # no comments
        s/^\s+//;               # no leading white
        s/\s+$//;               # no trailing white
        next unless length;     # anything left?
        my ($var, $value) = split(/\s*=\s*/, $_, 2);
        no strict "refs";
        $$var = $value;
}

if ( $action eq $empty )
        {
                #$action = "empty";
                print "starting service\n";
                `$logstatus "Starting VMManage"`;
                ` /root/.vmmanage/$service > /var/log/vmmanage/vmmanage.log 2> /var/log/vmmanage/vmm
        }


if ( $action eq "move" )
        {
                my $newloc = getNextLocation();
                `$logstatus "Requesting migration of VM: just moving"`;
                `$logstatus "Now using $hostname.$newloc.mln "`;
                `mln client -c upgrade -f /root/.vmmanage/$hostname.$newloc.mln -h shadowfax.vlab.iu.
        }



if ( $action eq "move_up" )
        {
                if ( getUpLocation() > getCurrentLocation() ){
                my $newloc = getNextLocation(); #location up
                `$logstatus "Requesting migration of VM: moving up"`;
                `$logstatus "Now using $hostname.$newloc.mln "`;
                `mln client -c upgrade -f /root/.vmmanage/$hostname.$newloc.mln -h shadowfax.vlab.iu.
                }else{
                `$logstatus "VM already at best location"`;
                }
        }

if ( $action eq "move_down" )
        {
                if ( getDownLocation() < getCurrentLocation() ){
                my $newloc = getDownLocation(); #location down
                `$logstatus "Requesting migration of VM: moving down"`;
                `$logstatus "Now using $hostname.$newloc.mln "`;
                `mln client -c upgrade -f /root/.vmmanage/$hostname.$newloc.mln -h shadowfax.vlab.iu.
                }else{
                `$logstatus "VM already at least location"`;
                }
        }

if ( $action eq "move_to_base" )
        {
                if ( $mybase = getCurrentLocation() ){
                `$logstatus "Already at base"`;
                }else{
                my $newloc = $mybase;
                `$logstatus "Now using $hostname.$newloc.mln --moving to Base"`;
                `mln client -c upgrade -f /root/.vmmanage/$hostname.$newloc.mln -h shadowfax.vlab.iu.
                }
        }

if ( $action eq "move_from_home" )
        {
                if ( $myHomeBroadcast = getCurrentBroadcast() ){
                my $newloc = $away;
               `$logstatus "Now using $hostname.$newloc.mln --moving from home"`;
               ` mln client -f /root/.vmmanage/$hostname.$newloc.mln -S -c upgrade -h huldra.vlab.iu.
                }
                else
                {
                `$logstatus "Already out of home array"`;
                print "no\n";
                }
        }

if ( $action eq "move_to_home" )
        {
                if ( $myHomeBroadcast = getCurrentBroadcast() )
                {
                `$logstatus "Already in home array"`;
                print "no\n";
                }
                else
                {
                my $newloc = $home;
                `$logstatus "Now using $hostname.$newloc.mln --moving from home"`;
                `mln client -f /root/.vmmanage/$hostname.$newloc.mln -S -c upgrade -h huldra.vlab.iu.
                }
        }
```

```
sub getNextLocation {
    my $range_start = $min_range;
    my $range_stop = $max_range;

    # get the current value:
    open(LOC,"$locationfile");
    my $location = <LOC>;
    chomp $location;
    close(LOC);
    print "current location: $location\n";
    if ( not $location ){
        $location = $range_start + 1;
    } else {
        $location++;
    }

    if ( $location > $range_stop ){
        $location = $range_start;
    }

    open(LOC,">$locationfile");
    print LOC "$location\n";
    close LOC;
    return $location;
}

sub getUpLocation {
    my $range_start = $min_range;
    my $range_stop = $max_range;

    # get the current value:
    open(LOC,"$locationfile");
    my $location = <LOC>;
    chomp $location;
    close(LOC);
    print "current location: $location\n";
    if ( not $location ){
        $location = $range_start + 1;
    } else {
        $location++;
    }

    if ( $location > $range_stop ){
        #$location = $range_start;
    }

    open(LOC,">$locationfile");
    print LOC "$location\n";
    close LOC;
    return $location;
}

sub getDownLocation
{
    my $range_start = $min_range;
    my $range_stop = $max_range;

    # get the current value:
    open(LOC,"$locationfile");
    my $location = <LOC>;
    chomp $location;
    close(LOC);
    print "current location: $location\n";
    if ( not $location ){
        $location = $range_start;
    } else {
        $location--;
    }
    if ( $location > $range_stop ){
        #$location = $range_start;
    }

    open(LOC,">$locationfile");
    print LOC "$location\n";
    close LOC;
    return $location;
}

sub getCurrentLocation
{
    my $range_start = $min_range;
    my $range_stop = $max_range;

    # get the current value:
    open(LOC,"$locationfile");
    my $location = <LOC>;
    chomp $location;
    close(LOC);
    print "current location: $location\n";
    close LOC;
    return $location;
}

sub getCurrentBroadcast
{
        my $string = `ifconfig | grep Bcast | awk '{print \$3;}'`;
        my $bcast =  substr $string, 6;
       # print "current broadcst address: $bcast\n";
        return $bcast;
}
```

Listing A.2: Main vmmanage script

Appendix B

Scripts Used in scenario 1

```
#!/usr/bin/perl
#!/usr/local/bin/perl
####################################################################
#
# mysqlstatus - a Perl client for monitoring mysql slow queries on VMs
#
# Author: Nii Apleh Lartey
####################################################################

require 5.004;
strict;
no strict "vars";
no strict "refs";

my $current_slow = 0;
my $slow_rate = 0;
my $previous_slow = 0;
my $count = 0;
my $logstatus = "/root/.vmmanage/logstatus.pl";
my $vmmanage = "/root/.vmmanage/vmmanage";


#read from configuration file

my %config;
open my $config, '<', '/root/.vmmanage/mysqlstatus.conf' or die $!;
while(<$config>)
{
        chomp;                    # no newline
        s/#.*//;                  # no comments
        s/^\s+//;                 # no leading white
        s/\s+$//;                 # no trailing white
        next unless length;       # anything left?
        my ($var, $value) = split(/\s*=\s*/, $_, 2);
        no strict "refs";
        $$var = $value;
}


while (1)
{

        $current_slow =`mysqladmin status | awk '{print \$9;}'`;

        $slow_rate = $current_slow - $previous_slow;

        if ($slow_rate == $current_slow)
        {
                $slow_rate = 0;
        }
        print "slow rate:        $slow_rate\n";
        $mydate=`date`;
        print "time:             $mydate";

#Count the number of times indicator is above threshold
        if ($slow_rate > $threshold)
        {
                ++$count;
        }
        else
        {
                $count = 0;
        }

        print "count:            $count\n";
        print "threshold:        $threshold\n";
        print "**-------------------------------**\n";
        $previous_slow = $current_slow;

        if ( $count == $tarry )
        {
                `$vmmanage move`;
                $count = 0;
                print "sleeping for 60 secs\n";
                sleep 45;
        }


        sleep $daemon_interval;
}
```

Listing B.1: Mysqlstatus script for nii1.dyndns.org

Appendix C

Scripts Used in scenario 2

```
#!/usr/bin/perl
#!/usr/local/bin/perl
#####################################################################
#
# mysqlstatus - a Perl client for monitoring mysql slow queries on VMs
#
# Author: Nii Apleh Lartey
#####################################################################

require 5.004;
strict;
no strict "vars";
no strict "refs";

my $current_slow = 0;
my $slow_rate = 0;
my $previous_slow = 0;
my $count = 0;
my $logstatus = "/root/.vmmanage/logstatus.pl";
my $vmmanage = "/root/.vmmanage/vmmanage";


#read from configuration file

my %config;
open my $config, '<', '/root/.vmmanage/mysqlstatus.conf' or die $!;
while(<$config>)
{
        chomp;                  # no newline
        s/#.*//;                # no comments
        s/^\s+//;               # no leading white
        s/\s+$//;               # no trailing white
        next unless length;     # anything left?
        my ($var, $value) = split(/\s*=\s*/, $_, 2);
        no strict "refs";
        $$var = $value;
}


while (1)
{

        $current_slow =`mysqladmin status | awk '{print \$9;}'`;

        $slow_rate = $current_slow - $previous_slow;

        if ($slow_rate == $current_slow)
        {
                $slow_rate = 0;
        }
        print "slow rate:       $slow_rate\n";
        $mydate=`date`;
        print "time:            $mydate";

#Count the number of times indicator is above threshold
        if ($slow_rate > $threshold)
        {
                ++$count;
        }
        else
        {
                $count = 0;
        }

        print "count:           $count\n";
        print "threshold:       $threshold\n";
        $previous_slow = $current_slow;

        if ( $count == $tarry )
        {
                `$vmmanage move`;
                $count = 0;
                print "sleeping for 60 secs\n";
                sleep 45;
        }

#Count the munber of time indicator is below lowthreshold
        if ($slow_rate < $lowthreshold)
        {
                ++$lowcount;
        }
        else
        {
                $lowcount = 0;
        }

        print "lowcount:            $lowcount\n";
        print "lowthreshold:        $lowthreshold\n";
        print "**--------------------------------**\n";

        if ( $lowcount == $lowtarry )
        {
                `$vmmanage move_to_base`;
                $count = 0;
                print "sleeping for 60 secs\n";
                sleep 45;
        }


        sleep $daemon_interval;
}
```

Listing C.1: Mysqlstatus script for nii3.dyndns.org

```
#!/usr/bin/perl
#!/usr/local/bin/perl
########################################################################
#
# mysqlstatus - a Perl client for monitoring mysql slow queries on VMs
#
# Author: Nii Apleh Lartey
########################################################################

require 5.004;
strict;
no strict "vars";
no strict "refs";

my $current_slow = 0;
my $slow_rate = 0;
my $previous_slow = 0;
my $count = 0;
my $logstatus = "/root/.vmmanage/logstatus.pl";
my $vmmanage = "/root/.vmmanage/vmmanage";
my $status = "awake";

#read from configuration file

my %config;
open my $config, '<', '/root/.vmmanage/mysqlstatus.conf' or die $!;
while(<$config>)
{
        chomp;                  # no newline
        s/#.*//;                # no comments
        s/^\s+//;               # no leading white
        s/\s+$//;               # no trailing white
        next unless length;     # anything left?
        my ($var, $value) = split(/\s*=\s*/, $_, 2);
        no strict "refs";
        $$var = $value;
}



while (1)
{
         $status = "resting";

         my $time = time();
         my $now = int(( $time % 86400 ) / 300 );
         print "Now in the day: $now\n";

         if (($now >= $workstart) && ($now < $workstop))
         {
                $status = "awake";
         }

         if ($status eq 'resting')
         {
                print "Resting, will resume $workstart\n";
         }


        if ($status eq 'awake')
        {

        $current_slow =`mysqladmin status | awk '{print \$9;}'`;

        $slow_rate = $current_slow - $previous_slow;

        if ($slow_rate == $current_slow)
        {
                $slow_rate = 0;
        }

        print "slow rate:       $slow_rate\n";
        $mydate=`date`;
        print "time:            $mydate";

#Count the number of times indicator is above threshold
        if ($slow_rate > $threshold)
        {
               ++$count;
        }
        else
        {
                $count = 0;
        }
        print "count:           $count\n";
        print "threshold:       $threshold\n";
        print "rest will start  $reststart\n";
        $previous_slow = $current_slow;

        if ( $count == $tarry )
        {
                `$vmmanage move`;
                $count = 0;
                print "sleeping for 60 secs\n";
                sleep 45;
        }

#Count the munber of time indicator is below lowthreshold
        if ($slow_rate < $lowthreshold)
        {
                ++$lowcount;
        }
        else
        {
                $lowcount = 0;
        }

        print "lowcount:             $lowcount\n";
        print "lowthreshold:         $lowthreshold\n";
        print "**------------------------------**\n";

        if ( $lowcount == $lowtarry )
        {
                `$vmmanage move_to_base`;
                $count = 0;
                print "sleeping for 60 secs\n";
                sleep 45;
        }
        }

        sleep $daemon_interval;
}
```

Listing C.2: Mysqlstatus script for nii5.dyndns.org

Appendix D

Scripts Used in scenario 3

```
#!/usr/bin/perl
#!/usr/local/bin/perl
#######################################################################
#
# queueproc - a Perl script for processing queue base polices on VMs
#
# Author: Nii Apleh Lartey
#######################################################################

require 5.004;
strict;
no strict "vars";
no strict "refs";

#my $bigjobs = 'grade 1';
my $count = 0;
my $logstatus = "/root/.vmmanage/logstatus.pl";
my $vmmanage = "/root/.vmmanage/vmmanage";

#read from configuration file

my %config;
open my $config, '<', '/root/.vmmanage/queueproc.conf' or die $!;
while(<$config>)
{
        chomp;                  # no newline
        s/#.*//;                # no comments
        s/^\s+//;               # no leading white
        s/\s+$//;               # no trailing white
        next unless length;     # anything left?
        my ($var, $value) = split(/\s*=\s*/, $_, 2);
        no strict "refs";
        $$var = $value;
}



while (1)
{
#take jobs in queue.
if($localqueue == 'empty')
{
`ssh mroot\@nii3.dyndns.org mv /var/www/jobs /var/www/nii4jobs`;
}
#check what to do with jobs
open(STREAM,"wget -q -O - http://nii3.dyndns.org/nii4jobs |");

while ( $line = <STREAM> ){
        chomp $line; # remove newline at the end
          if ($line =~ /$bigjobs/)
          {
                $line =~ s/.* (\d+) $bigjobs .*/$1/g;
                ++$count;
        }

}

if($count >= $threshold)
{
        my $attempt = `$vmmanage move_from_home`;
        print "$attempt\n";
        if ($attempt =='no')
        {
        print "should process away from home now\n";
        `$logstatus "Would process: away from home"`;
        processQueue();
        $localqueue = 'empty';
        }
        else
        {
        print "Would process queue after migration away from home";
        `$logstatus "Would process queue after move away from home"`;
        }
}
if($count < $threshold)
{
        my $attempt = `$vmmanage move_to_home`;
        print "$attempt\n";
        if ($attempt =='no')
        {
        print "should process at home now\n";
        `$logstatus "Would process: at home now"`;
        processQueue();
        $localqueue = 'empty';
        }
        else
        {
        print "Would process queue after migration to home";
         `$logstatus "Would process queue after move to home"`;
        }
}

sub processQueue
{

open(STREAM," wget -q -O - http://nii3.dyndns.org/nii4jobs |");

while ( $line = <STREAM> ){
        chomp $line; # remove newline at the end
         if ($line =~ /grade/)
         {
                $calc = substr $line,10;
                print "equation: $calc\n";
                $result = `echo $calc | bc`;
                print $result;
         }

        }
        }

        sleep $daemon_interval;

}
```

Listing D.1: Script for processing queue information: queueproc

Appendix E

Other Scripts

```
#!/usr/bin/perl
use warnings;
use Math::NumberCruncher;
open(STREAM,"$ARGV[0]");
#$ref = Math::NumberCruncher->new();
while ( $line = <STREAM> ){
        chomp $line; # remove newline at the end
        #print "Line: $line\n";
        if ($line =~/$ARGV[1]/)
         {
                $line =~ s/.* (\d+) $ARGV[1] .*/$1/g;
                $mins = substr $line,4,2;
                #print "mins : $mins";
                $mins = 60*$mins;
                #print "$line\n";
                $sec = substr $line,7,6;
                $totalsec = $mins+$sec;
                #print "totalsec: $totalsec\n";
                $ARRAY[++$#ARRAY] = $totalsec;

         }

}

foreach $line (@ARRAY)
{
#       print "$line\n";
}
$mean = Math::NumberCruncher::Mean(\@ARRAY);
($high,$low) = Math::NumberCruncher::Range(\@ARRAY);
$median = Math::NumberCruncher::Median(\@ARRAY);


print "mean $ARGV[1] is: $mean\n";
print "median $ARGV[1] is: $median\n";
print "range of $ARGV[1] is $low - $high\n";
```

Listing E.1: Script for statistical distribution of time spent to process jobs