UNIVERSITY OF OSLO
Department of Informatics

Towards Automatic Management and Live Migration of Virtual Machines

Master Thesis

Ingard Mevåg
Oslo University College

May 23, 2007

Towards Automatic Management and Live Migration of Virtual Machines

Ingard Mevåg
Oslo University College

May 23, 2007

Abstract

This project will concider management of a Xen-based virtual infrastructure in several aspects. First, management of the infrastructure itself with configuration and provisioning of virtual machines, followed by a proof-of-concept model for dynamic resource sharing for the virtual machines using the newly introduced Xen-API for resource consumption monitoring. Experiments will document the feasibility of such a model, using both single, independent virtual machines and various size private networks connected by local bridge devices. Two applications, XeniNFO for resource consumption monitoring and XenGuard for load balancing of the virtual infrastructure using live migration, have been developed. XenGuard, which utilizes the features of XeniNFO, has been merged with the open source virtual machine management tool MLN (Manage Large Networks) and will be publicly available in a future release of MLN while XeniNFO has been contributed to XenSource and is publicly available from the Xen-Unstable repository.

Acknowledgements

First of all I would like to extend my utmost gratitude to my project advisor, Assoc. Professor Hårek Haugerud. Thank you for your time, effort, our numerous discussions about everything, related and unrelated and last but not least for keeping my spirits up.

Kyrre Begnum deserves equal gratitude. Thank you for being an ever inspiring person, coming up with the project idea, showing great interest in my progress and for general, much valued, conversations and input on just about everything regarding this project work. Also, much appreciated was your help and effort on rewriting and adopting MLN to suit this project's needs. I hope we will have the opportunity to extend the cooperation and make MLN-XenGuard an official release in the future.

Thanks to Alexander Andersen for fruitfull discussions, text editing aid, being a worth tabletennis opponent and for being my number 1 competitor in our struggle to achieve good grades throughout our time in the OUC Network and System Administration masters program. To Edson Ochoa for being my partner in crime for the last five year here at OUC as well as also being a worthy tabletennis opponent. To the rest of my classmates, thank you for many good times at "kroa" and other social events.

Thank you mom and dad, for motivating talks, free meals and proof-reading help.

Finally i would like to thank my friends, Asle Nødtvedt, Eskil Kristiansen, Alf Marius Foss Olsen, Yngve Tetlivold, Vegard Hamnes, Christian Fossmo and everyone else i have forgotten for believing in me, continuous support, keeping my spirits up and being good friends throughout the duration of this work.

Once again, Thank you all!

Oslo University College, May 2007

Ingard Mevåg

Contents

Chapter 1

Introduction

Virtual Machines, or guest-hosts, are individual hosts separated from the physical hardware of a PC or server by a virtual machine monitor. The virtual machine monitor (VMM) is a thin software layer acting as a mediator between the software and the hardware. The guest-hosts have no notion of the the actual physical hardware, but see only the virtual hardware provided by the VMM. Every guest host operates with the same abstracted hardware since the VMM provides an uniform view of the actual physical hardware independent of the vendor. This, in turn, yields numerous advantages. To mention a few:

- All software written for the physical hardware will work on any guest host - Programmers, software engineers and developers will not have to concider different hardware architectures. Legacy software solutions will be compatible while system administrators can update and renew operating systems and applications

- Server consolidation increases server utilization percentage and reduces server proliferation and equipment costs

- Server migration - Hardware upgrades and maintenance is easily dealt with. Migrating a server from one physical host to another is simplified to the extent of a few mouse clicks or commands and results in reduced maintenance time and costs and eases server management severely

- Backup and disaster recovery - Backups of whole virtual machines can be restored, and services brought back by simply booting up the backup with a known "good" state

- On-demand services - Virtual machines providing any services can be started and stopped anywhere in the virtual infrastructure when it is needed

- Resource distribution - With a virtual infrastructure, and a proper setup, the combined resources of all the physical servers can be looked at as one united pool of resources and can be re-allocated to suit changes in the virtual machines load and/or needs.

1.1 Motivation

The virtualization technology has been around for decades. The area of usage has changed somewhat back and forth over time as prices for hardware and floorspace have varied. Vendors have developed and implemented virtualization support in their products and software, both open and closed source, has been made available for more operating systems and with support for a greater range of hardware as well as offering more advanced solutions.

At present time, virtualization in general is getting increased attention by several groups within the information technology community. There are solutions for "home" usage such as running Microsoft Windows applications in a separate window on top of for instance a Linux or Macintosh OS X operating system. In system administration communities, products such as the VMware ESX server and the XenSource Xen hypervisor, are seeing increased production usage to host fully virtual infrastructures which is beneficial in terms of resource usage and sharing, redundancy, ease of management, cost and many more.

The resurrection of virtualization technology in the system administration aspect is what has motivated the author to start working in this field. Oslo University College (OUC) has been using virtual machines in various teaching scenarios regarding both system administration, networking and operating systems in general. The author has experienced a growing interest in this field both by playing with various VMware products at home and also through the virtualization lab at OUC. Several aspects of virtualization research has been conducted at OUC in recent time:

- A tool for virtual machine creation, together with a simple interface for change management[1][2]
- An evaluation of the live migration feature of Xen in an automated failover scenario[3]
- Benchmarks of virtualized cluster deployment and computation performance[4]

There are still, however, multiple, un- or partially explored challenges related to virtual infrastructures. The VMware product family has seen a lot of improvement over the years and they have a lot of features to offer through standalone, or different combinations of their products. VMware, however, is not using open source products, nor do they offer free licenses for research. They also have a user agreement license which restricts publications of any comparisons or benchmark including their products.

This makes the XenSource opensource Xen hypervisor a lot more attractive when conducting research on virtual infrastructures, hence it was the obvious choice for this project. Some particular challenges, which is the motivation behind this project, concerns the deployment and management of the infrastructure as well as providing sufficient resources for the virtual machines to operate as though they would have the physical hardware to themselves.

1.2 Challenges

To narrow down the scope of the research even further, these are the challenges this project aims to address:

- Configuration and change management for virtual- machines and infrastructure
  - Commercial products exist already - This project will focus on finding a proper open source solution that will allow for expansion easily
- Focus on open source
  - Support further development and introduction of new competitive features for the open source community tools
- Automated load balancing for a virtual infrastructure based on resource consumption statistics
  - Main focus - An extension to a management tool will have to be made from scratch

The latter of these challenges, as pointed out, is where this project has focused the most. The following sections will address how these challenges are met and how they are intended to be solved. Table 1.1 will explain some of the acronyms and conventions used in this thesis as well as in virtualization technology in general.

Expression	Meaning
VM	Virtual Machine
VMM	Virtual Machine Monitor
VI	Virtual Infrastructure
Physical host/node	A physical server in the virtual infrastructure
Guest (host)	A virtual machine
Hypervisor	Thin software layer which handles all interaction between virtual machines and hardware
Dom(ain)0	The control domain of a Xen server (also a virtual machine)
HVM guest/host	A virtual machine running on top of virtualization friendly hardware
PV guest/host	A virtual machine using the ParaVirtualization technology
LVM	Logical Volume Manager - Software used to ”partition” storage volumes
AoE	ATA over Ethernet - Software used to share storage volumes across the network
SAN	Storage Area Network - Technology for remote storage with redundancy
MLN	Manage Large Networks - Software used to configure, provision, monitor and manage the virtual machines
Service_host	A MLN term - The physical server on which the VM resides

Table 1.1: Acronyms and conventions used in this thesis and virtualization research in general

1.3 Configuration and change management for virtual-machines and infrastructure

Managing virtual machines and a virtual infrastructure can be done in many different ways. The Xen hypervisor has its own ”control domain”, which is, as all other guest hosts, a virtual machine running on top of the hypervisor software. This VM however, can control, configure and provision other VMs. From a localhost perspective, an administrator can log in on Dom0 as normally on a console and is presented with a regular Linux prompt. With root user access the administrator can control every aspect of the server, both regarding the Xen hypervisor's configuration and that of the virtual machines using Xensource built in tools. This might be a viable solution if the virtual infrastructures consist of a conveniently small number of servers. However, *n*

Xen servers will be n times the labour if all aspects of configuring the virtual infrastructure is to be manual labour, hence an infrastructure wide management tool is both convenient and necessary.

The following list of software (not complete), taken from the Xensource wiki[1] and the Xen-users mailinglist[2], are some of the most used and renowned opensource remote monitoring and management solutions:

- LibVirt
- Virtual Workspace Service
- XenMan
- DTC-Xen
- Enomalism
- openQRM
- Argo
- PHPmyXen
- MLN
- Virt-manager

There are other non-freeware solutions as well mentioned on the Xensource wiki, but this project is focusing on opensource, non cost, tools.

For the management of the testbed virtual infrastructure, this project choose to use MLN. This is an opensource tool developed partly at OUC and written in the Perl programming language. Both being an advantage for this project as the main developer could easily be reached, and was frequently consulted, for usefull discussions.

From an infrastructure management perspective, there are quite a few bonuses when changing from physical to virtual machines. MLN has the following key features which makes it a very usefull tool to manage and monitor the virtual infrastructure:

- Centralized management and distributed workload - Each Xen node in the infrastructure is running the MLN daemon which handles all VMs on each respective physical server. The management node communicates with these daemons over a Unix socket

[1]http://wiki.xensource.com/xenwiki/XenRemoteManagementInterfaces
[2]http://lists.xensource.com/xen-users

- Built in support for remote storage through AoE[3] and LVM[4]
- On the fly change management, (live) migration and re-provisioning

MLN, however, is not using any of Xen's built in remote management interfaces, but rely on the local daemons to communicate with the hypervisor through the tools provided locally. Namely the *xm* python program.This project found that the new Xen-API which was released as a preview for the Xen 3.0.4 release, holds more potential to dig out more detailed and accurate measurements and information. This monitoring of the virtual infrastructure is then in turn meant to be analyzed and provide an administrator with detailed information of the status of all physical and virtual hosts. The new Xen-API[5] is using XML-RPC for client/server communication and is meant to standardize the data model used internally. The API, which is scheduled for a v1.0 release in the upcoming Xen 3.0.5 release, is the new long-term maintained interface of Xen.

1.4 Open source

This project will maintain a strong focus on developing a solution to the challenge in section 1.5 with open source in mind. A open source management tool will be used, and modified, to suit the automated resource distribution mentioned. Commercial products exist, mainly from VMware, that focuses on dealing with this, but with a an extensive pricetag attached. Even Xen is including management of for instance remote storage in their XenServer and XenEnterprise editions of the Xen hypervisor, but no solution exist for automatic live migration of the virtual machines. Many strong factors recommends this sort of development:

- Other opensource products such as Linux itself and MySQL are doing very well and have other business strategies such as marketing support only
- Research with virtualization is rarely done with pure commercial products and a continued effort in developing a wider variety of solutions and features for Xen will help stimulate further progress and raise the interest in creating a better qualified competitor to the best commercial products
- Commercial products often have licenses which restricts the use of the software and limits further modifications to suit individual needs. Some

[3]http://www.coraid.com/documents/AoEDescription.pdf
[4]http://sources.redhat.com/lvm2/
[5]http://wiki.xensource.com/xenwiki/XenApi

times it even limits the rights to publish benchmarks and similar comparisons which in turn might have produced healthy competition and helped spread higher integrity information to the general public

Open source products allows for re-use, changes and further development, depending on which license is chosen by the author. The *XeniNFO* application was mentioned as an example to communicate with the Xen-API using the Perl programming language on the Xen Summit conference of April 2007, and was also contributed to Xensource, under the creative commons license[6], for possible inclusion in the next release for the same reason.

1.5 Automated load balancing for a virtual infrastructure based on resource consumption statistics

All virtual machines in any given infrastructure has its own set of defined static variables which tells how much resources they get in terms of virtual memory and diskspace. CPU resources however is dynamically allocated for the virtual machines depending on what CPU sharing algorithm the VMM is programmed with and the needs of the virtual machine at any given time. If a virtual machine, lets say it is running a webserver service, is peaking in usage at any point, the underlying infrastructure can only allow it to use as much CPU resources as is available on the physical host at that point in time. This will change depending on the usage level of the different virtual machines, but available resources will allways be limited by the restraints of the physical hardware which is shared among them. This project will strive to achieve an application that can live migrate virtual machines to other physical locations where more resources are available, i.e. where the physical resources are less utilized or more capable of serving higher load. This process will happen through resource utilization monitoring and calculating where virtual machines should be placed for the distribution of resources to be spread evenly across the infrastructure.

The previously mentioned Xen-API is a key feature of Xen for gathering resource consumption statistics, both from the physical and the virtual machines. During the course of work on this project an application called *XeniNFO* was created to gather these statistics on a continuous basis. This application was built around the Xen-unstable development release and as such was constantly improving along with new features being introduced in the development repository.

[6]http://creativecommons.org/

This application, and the Xen-API it communicates with, was at a later stage merged into the source code of MLN. The reasons for doing this was simply to not reinvent the wheel as both MLN and XeniNFO brought different features to the table. XeniNFO provided detailed resource usage numerics which could be analyzed and used to make decisions about possible re-location of one or more virtual machines in order to balance resource consumption across the virtual infrastructure. Further development on the automated live migration of domains was accompanied with the XeniNFO add-on to change name to *XenGuard*. The goal for this work was to create an application that could analyze resource utilization on the virtual infrastructure and, through live migration, distribute and balance the total load across all available physical servers.

The following is the inner workings of the XenGuard application in short form:

1. XeniNFO: Gather information about the virtual infrastructure, its virtual machines and the total resource consumption and distribution
2. XenGuard: Run logic for load balancing and decide what needs to be relocated where
3. XenGuard + MLN: Activate the live migration process to come to terms with what was decided and configured in step 2

1.6 Thesis outline

This report will go through the following chapters:

- Chapter 2 will discuss background information about virtualization in general
- Chapter 3 will give an introduction to tools and software used throughout this project work
- Chapter 4 will discuss the development process and functionality of the tools produces by this project
- Chapter 5 will explain all experiments and results
- Chapter 6 will discuss the whole work process, the results found in chapter 5 and end with a final conclusion

Some key words and statements have been selected to describe the work: *Virtual infrastructure management*, *Live migration*, *Live migration of complete networks*, *Xen*, *Open source*, *Load balancing*, *Autonomic management* and *Dynamic resource distribution*

Chapter 2

Background

2.1 The dawn of virtualization

Virtualization[5][6] has been a recurring topic of interest since the late 1960's. At first, VMMs were developed to better utilize the little, not to mention expensive, mainframe hardware and its resources so that multiple applications and servers could coexist on the same physical host. Present technology is heading somewhat in the same direction again, but now also considering redundancy and ease of management/downtime as benefits. In-between present time and the dawn of the VMMs however, the industry experienced a lack of interest when hardware-costs dropped and modern operating systems were introduced. The ideas of the 80's and 90's did not include saving floorspace by virtualization of multiple servers and applications in one physical server. The motivation for research in this field today is increasing dramatically as more hardware is developed with built in native virtualization support and more software solutions appear. The trends in modern system administration are gradually moving towards a virtualized infrastructure.

2.2 What is Virtualization

Virtualization[7] is when one, using a software layer, decouples hardware from software and creates a logical representation of the underlying physical hardware so that resources can be controlled, distributed and presented in ways that might be more suitable than the actual configuration. The virtualization is thereby not restricted to the actual physical configuration. For instance, in a multi processor setup, the CPUs could be represented as one single processing unit so that software developed to only utilize one CPU could in fact utilize all the processing power available.

A Virtual Machine[8] is a guest host and a virtual environment created by a

VMM which sees the presented abstraction as its original hardware. The VM operates with a notion of being a standalone host as the VMM provides total mediation of all interaction between the actual physical hardware and the VM itself resulting in the VM to be completely isolated. Or as Gerald J. Popek and Robert P. Goldberg [9] puts it: ″A virtual machine is taken to be an efficient, isolated duplicate of the real machine″.

Popek and Goldberg also characterize the VMM with the following characteristics: ″First, the VMM provides an environment for programs which is essentially identical with the original machine; second, the programs run in this environment show at worst only minor decrease in speed; and last, the VMM is in complete control of system resources″.

Popek and Goldberg describes three main properties when analyzing the environment created by a VMM:

1. Equivalence: a program running under the VMM should exhibit a behavior essentially identical to that demonstrated when running on the original machine directly.
2. Resource control: the VMM must be in complete control of the virtualized resources.
3. Efficiency: use the native hardware of the physical machine to as great a degree as possible.

Although stated more than 30 years ago, these properties are still valid. The VMMs of today are typically assumed to satisfy properties 1 and 2, and partially 3 depending on which virtualization technologies are used.

Different virtualization techniques will be discussed in the subsequent section.

2.2.1 Common techniques

There are several ways of presenting a virtual hardware layer, and the many Virtual Machine Monitors solve the problems in different ways. The approach depends on the underlying hardware. Support for full virtualization was never a part of the x86 architecture[10] which results in very sophisticated techniques having to be used in order to trap and virtualize the execution of certain instructions. Hardware such as the IBM System 370[11][12][13]/390/zSeries or Motorola MC68020[14] which allows a virtual machine to run native does not incur any conciderable overhead compared to the x86 architecture.

The x86, or IA-32, architecture does not meet the virtualization requirements that [9] Popek and Goldberg describe in their paper. This is because the VMM loses control of the total resource pool if a guest host tries to perform a privileged mode operation like disabling interrupts. However efficient virtualization is possible as the x86 architecture supports 4 different privilege levels. The x86 privilege levels are generally described as ring0 through ring3, ring0 being most privileged. The operating system usually runs all its code in ring0, while applications run in unprivileged mode - ring3. If one modifies the operating system to run in ring1 and the VMM runs in ring0, one would get a system that never relieves the VMM of its control because code that executes in ring1-3 is not allowed to do privileged instructions and thus has to go through the VMM. Modifications like these is what is used in the paravirtualization technology. A CPU architecture is virtualizable if it supports the basic VMM technique of direct execution, meaning execution of the virtual machine on the real machine while letting the VMM remain in control of the CPU. The most prevalent techniques for solving this problem is paravirtualization [15][16][17] and direct execution combined with fast binary translation [6][18].

2.2.1.1 Full virtualization

Full virtualization is achieved when the VMM simulates the complete underlying hardware so that unmodified operating systems and its applications can run on what seems like legacy hardware. The operating systems does not have to be designed for the actual underlying hardware, and will have no notion of its existence. However, in order to achieve full virtualization on architectures such as the x86, which doesn't support native virtualization, one has to translate portions of the original instruction set in order for the VMM to keep control of the CPU even when the virtual host for instance turns off interrupts. VMware is using this technique[6] in their products and what happens is that the VMM does translation of all the instructions on the fly which are not virtualizable into other equivalent virtualizable instructions. So the VMM keeps control of the CPU but turns off interrupts for the current virtual machine until told otherwise by the machine itself.

2.2.1.2 Paravirtualization

To avoid the drawbacks and overhead induced by the full virtualization technology another alternative known as paravirtualization has been introduced. The VMM presents an abstraction that is similar, but not identical to the underlying hardware. This approach is perhaps the most prevalent as it proclaims near native performance, but it does require modifications made to the operating system intended to run in a guest host. The VMM builder defines the virtual machine interface by replacing nonvirtualizable instructions in the original code with more efficient equivalent instructions which are easier to virtualize.

2.3 The main contributors

2.3.1 VMware

The VMware history[19] began in the late 1990's when researchers at Standford University started to look into the potential of virtual machines for using commodity OSs on parallel processing machines. These machines were hard to program and could not run any existing operating systems. With the use of virtual machines they discovered that they could make the complex underlying architecture look sufficiently similar to existing platforms to leverage current OSs. This project was the start of the ideas, and the people behind them that has now formed into VMware Inc.[1]

VMware offers a range of different products. VMware Workstation[20], VMware ESX server[21][17][22][18]and VMware Server which is built on the deprecated VMware GSX server and is now VMware's free alternative. The free edition, version 1.0.0 was released in October 2006, is available for the Microsoft Windows and the Linux platforms. It is currently, as of May 2007, at version 1.0.2 released in February 2007.

Allthough the different VMware products vary somewhat in features and implementation, they all share the general principle of execution. Code segments are scanned and instructions identified as non-virtualizable are either translated on the fly so that it jumps to the VMM or substituted with sequences of equivalent instructions which trap or safely perform the original operation. This results in a full virtualization system which allows for unmodified operating systems to work as if all instructions where executed directly on the hardware.

2.3.2 Xen

XenSource[2], as well as VMware, have both commercial and free, open source products. To compete with VMware, both in terms of management and support, they have the XenServer which only supports Windows virtualization and XenEnterprise which is XenSource's alternative to VMware ESX. The hypervisor, however, is the same in all of XenSource's products.

[1]http://www.vmware.com
[2]http://www.xensource.com

The Xen hypervisor[17] is an open-source project which utilizes the paravirtualization technology. They implement a small software layer, called the hypervisor, between the hardware and the operating system. With the paravirtualization technology it is required to patch, or modify, the guest operating system(s) that is to be used in order to go around the on the fly binary translation that induces a substantial overhead. When the operating system is modified it can use the hardware directly without leaving the hypervisor without its control of the total resourcepool.

The current release as of late May 2007, is Xen 3.1.0 which has had built in support for the AMD Pacifica and Intel Vanderpool hardware virtualization technologies since early 2006. The new technologies from Intel and AMD satisfies Goldberg and Popeks[9] requirements for virtualizable architecture and is thereby able to support unmodified operating systems, i.e. running Windows on top of Xen. As of the latest Xen release, 3.1.0, support for live migration of hardware virtualized domains is also available which makes Xen an even stronger competitor for VMware's enterprise ESX release.

2.4 Hardware development

2.4.1 IBM

IBM has since the release of their IBM System 370 mainframes[11][12][13][23] in June 1970 supported full native virtualization[24]. They have been pioneering the virtualization technology on the mainframe side ever since. The System 370 underwent many architectural improvements in its 20 year lifespan until it was replaced by the System 390[25] in the 1990s. The mainframeseries was renamed to XA/370 in the 1980s when they started supporting 32bit processing and more recently, in 2000, replaced by the new [26][27]IBM zSeries mainframes supporting 64bit processing.

2.4.2 Others

With the release of Intel's Vanderpool[28] and AMD's Pacifica technology, Xen is now able to support running unmodified operating systems and VMware may in the future no longer be needing their on the fly binary translator. What they both do is adding a new execution mode to the CPU so that the VMM can safely and transparently use direct execution for running virtual machines rather than making existing modes virtualizable. To improve performance the mode attempts to reduce both the traps needed to implement virtual machines and the time it takes to perform the traps.

2.5 Achievements

Back in the 60s and 70s the birth of the virtual machine monitor meant better utilization of large and expensive mainframe computers. The VMM provided a compelling way of multiplexing resources amongst different applications and the technology was very prosperous. Then, in the 80s and 90s commodity hardware became cheaper and use of the virtualization technology disappeared to the extent that hardware architectures no longer supported virtualization in the way that Goldberg and Popek had layed out. Nowadays researchers and academics are working continuously to better the virtualization technology. The drop in hardware costs had led to a proliferation of servers which all occupied floorspace and induced substantial management overhead. The complexity of modern operating systems which made them so powerful was also an origin for failures and vulnerabilities. Here is where the virtualization technology comes in. System administrators started resorting to one application per server in order to protect against break-ins and system crashes. This again led to an increase in hardware requirements, a higher equipment budget and more time spent on maintenance. Moving applications and services that once ran on separate hardware servers onto virtual machines and consolidating those machines onto one or a few physical servers reduces hardware costs and time spent on maintenance and increases the use percentage of the available resources justifying expensive server hardware.

The trend seems to be that from a VMM being a utility for multiplexing resources and multitasking, the future of this technology is more and more a solution for security, redundancy and reliability. Functions such as server migration have proven to be difficult to achieve on satisfactory levels and seem more suitable to implement on a virtual infrastructure. Innovative operating system solutions can be developed and deployed while keeping the existing software base.

2.5.1 Redundancy - Towards less downtime

Reliability and redundancy are key features in modern virtualization technology. Mission critical applications do not tolerate downtime and corporations can quickly loose money[29] if their availability is affected by either maintenance, hardware- or software failure or malicious activity. A virtual infrastructure can utilize live server migration to move running production servers to other network hosts in order to do for instance scheduled maintenance or replace faulty hardware. This makes the network setup very fault-tolerant and eliminates any maintenance or management downtime.

2.5.2 Reduced cost

Data center managers are faced with the challenge of limiting IT-spending and reducing server proliferation. The practise of dedicating one server to each service or application is costly and precious IT-resources are stretched thin procuring, provisioning and maintaining a growing number of under-utilized servers. Business continuity requires continuous server uptime and meeting this demand is costly in terms of redundancy, management and maintenance. The solution could be a virtual infrastructure.

With the ability to separate applications and services from the hardware and use all servers as a united pool of resources one can easily deploy or move any services to a system in that pool. This, in turn, makes administrators able to look at resources in a whole new way. Old and new equipment can be used to its full potential when sharing resources and no individual network component is a potential bottleneck if something should go wrong.

2.5.3 Maintenance

Server maintenance is the biggest factor in downtime at high availability sites. Studies[30] show that scheduled maintenance is the cause for 75-90% of all server downtime. Despite this fact , no practical approach exists to combat this problem. Most administrators typically deploy their servers on redundant hardware to protect against hardware failure, but this does not help if for instance a motherboard or a disk controller is failing, or any other part that is not possible to duplicate in a redundant server setup. Virtual machine technology deals with this challenge with just simply moving the virtual machine(s) from the physical host that needs maintenance onto other server hardware while the virtual machine is running and maintaining service availability. With the virtual machines running elsewhere there is no harm in doing a server shutdown in order to do maintenance quick and undisturbed. The same migration possibilities also applies for situations like equipment lease ends and server hardware upgrades.

There are also other approaches to minimize downtime when doing scheduled maintenance. One is using what D.E. Lowell et. al. named devirtualizable virtual machines[30], which is a solution for enabling virtualization only when doing maintenance and upgrades/updates. When virtualized the microvisor in their solution causes 5.6% CPU overhead and no overhead at all when devirtualized.

2.5.4 Server migration

Manual server migration is still common but with virtualization and the ease of server migration it offers, this is about to change. Administration of clusters and datacenters is made easy with the possibility of live server migration[31]. The complete separation of hardware and software increases fault tolerance in the network and simplifies load-balancing and server hardware maintenance. Load balancing is simplified when administrators can monitor either manually or automatically the complete resource pool of datacenters or clusters. If resources become scarce on one physical host one can simply migrate a running virtual machine to another physical host in the network that has more resources available. Another alternative is to instantiate a clone or generate a new virtual host from a template and bring it up on a host with more resources available and do loadbalancing between two or more virtual machines.

Xen has built in live server migration tools and the VMware Enterprise solution comes with a management product called VirtualCenter which handles what VMware calls VMotion[32]. Both Xen and VirtualCenter with VMotion lets administrators move virtual machines across the network of connected physical hosts in for instance a datacenter or a cluster while at the same time maintaining continuous service availability.

2.6 Overhead and performance

Is the overhead caused by the VMMs justifiable? Lowell et al.[30] claim that high availability sites would not sacrifice a 10-20% overhead at ”typical enterprise workloads”. They continue by stating that they believe enterprise customers will resist even paying a 10% performance penalty at times of peak load to let them perform infrequent maintenance without downtime.

Barham et al.[17], presents a series of different benchmarking results comparing both VMware’s ESX server and Xen to native performance. Their findings show that the paravirtualized approach used in Xen performs no less than at least 93% compared to native while the ESX server performed as low as only 13% on very I/O intensive benchmarks. The tests include benchmarking of the PostgreSQL databaseserver and the Apache HTTP server. They conclude: ”the performance of XenoLinux over Xen is practically equivalent to the performance of the baseline Linux system”.

Both Menon et al.[33] and Kiyanclar[18] present results in their respective studies that show Xen to perform within 5% of native performance on heavy I/O operation tasks. None of them offer quantitative results for the ESX server because of license restrictions on benchmarking. However Kiyanclar concludes with both The VMware ESX server and Xen to be attractive candidates for his On-Demand Secure Cluster Computing project and says ”While ESX server is noted in the Xen work as having better performance than the hosted VMware editions (VMware Workstation), the same notes that Xen still outperforms the higher-end VMware product.”

Little recent, as of 2007, information is available. Comparisons of for instance all virtualization technologies and how they use the newly introduced hardware which support virtualization, namely Intel’s VT and AMD’s V chips, could be very interesting. Instead we have to monitor what the makers of the software release themselves in various white papers and presentations. As the continued reading of this section will display, commercialization has a negative impact on both research progression and spreading of high integrity information. Subjective test results and other types of biased information is spread by developers and others in trying to promote their products. This is not very fruitful, neither for research nor cooperation on improving technology.

Technology, virtualization capable wise, is maturing fast after the virtualization trend has picked up its pace, and equally so is the software that runs on top of it. Comparisons and benchmark results introduced in section 2.6 are ageing quickly as improvements are made, new technology is available and more mature products are released to the public. VMware still does not allow anyone to publish performance benchmarks without their concent, but publish their own comparison benchmarks as they see fit since Xensource does not have any license difficulties regarding this. In late January 2007, VMware published a whitepaper entitled ”A Performance Comparison of Hypervisors”[34] which strongly suggest that the Xensource hypervisor, Xen 3.0.3 was used, is not ready for an enterprise production environment. The following is a quote taken from the conclusion of the mentioned paper:

> We found that VMware ESX Server is far better equipped to meet the demands of an enterprise datacenter than the Xen hypervisor. While Xen-based virtualization products have received much attention lately, customers should take a closer look at the enterprise readiness of those products. The series of tests conducted for this paper proves that VMware ESX Server delivers the production-ready performance and scalability needed to implement an efficient and responsive datacenter. Furthermore, we had no problems setting up and running virtual SMP and virtual machine scalability tests with the reliable and proven third-generation VMware ESX Server hypervisor. Despite several attempts, we were not successful in running similar tests with the Xen hypervisor.

The paper was also accompanied by numerous graphed measurements such as the one in figure 2.1. This graph, and the wording in the previous quote, implies a rather subjective comparison.

speaks for itself. It states loud and clear that this research is neither professional, nor meant to be taken into consideration by anyone trying to get an overview of virtualization products in a scientific scenario.

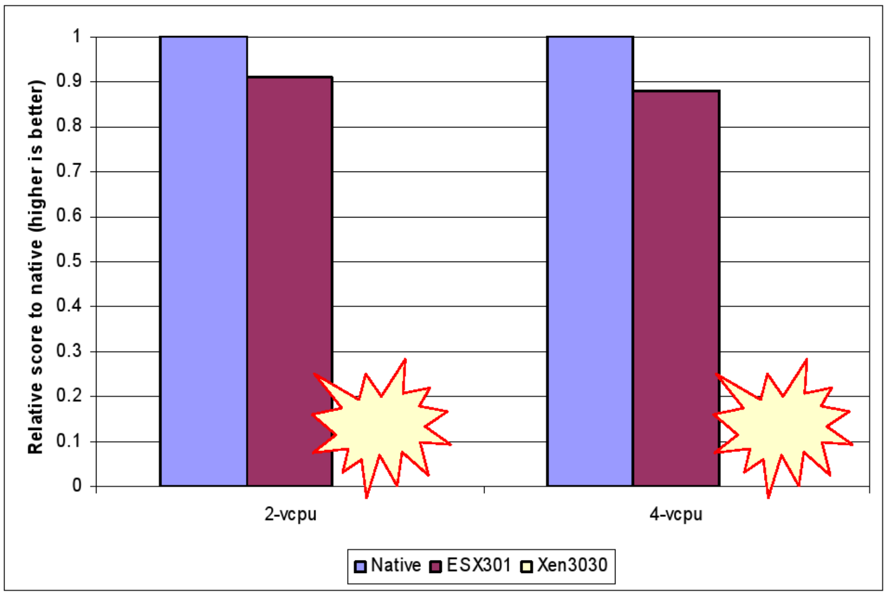

Figure 9 — SPECjbb2005 results compared to native (higher values are better)

Figure 2.1: Figure taken from a VMware white paper comparing Xen 3.0.3 and VMware ESX 3.0.1

After submitting, and getting an approval from VMware, Xensource published their own white paper entitled ”A Performance Comparison of Commercial Hypervisors”. This paper did the same comparison except they used the XenSource XenEnterprise 3.2 release which is the Xen equivalent to VMware’s ESX solution. Both products being commercial. Figure 2.2 shows one of many graphs in the white paper XenSource published as an answer to the VMware comparison released earlier the same year (2007). Figures 2.1 and 2.2 both show the result from SPECjbb2005 benchmarking.

As the graphs show, XenSource’s results might seem more plausible, however, XenSource could not refrain from returning the somewhat sarcastic wording in VMware’s comparison. The following quote is taken from the XenSource published comparison:

> Additionally, VMwares implementation and configuration of the Xen hypervisor could not be checked, and it is likely that inadvertent incorrect configuration by the testers influenced their results. It should be stressed that Xen 3.0.3 is a code base, and not a commercial product, and so it is difficult to understand what VMware was

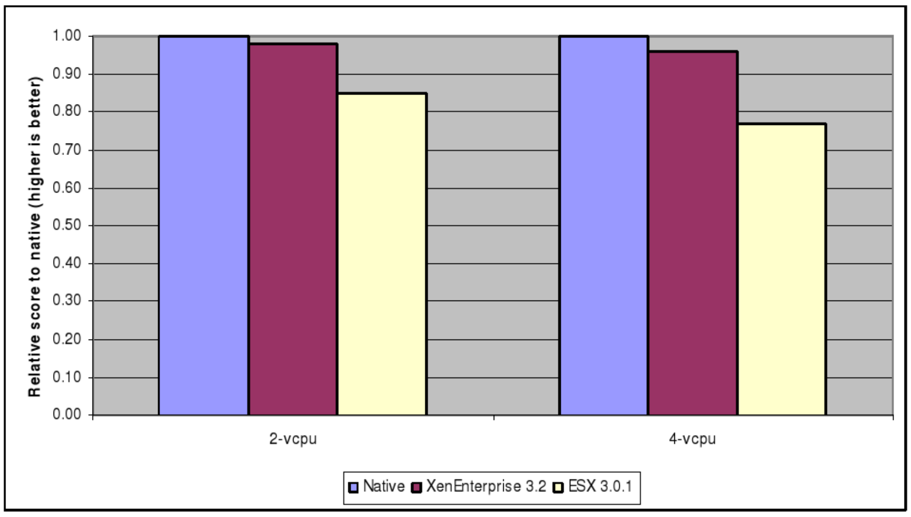

Figure 11 – SPECjbb2005 results compared to native (higher values are better)

Figure 2.2: Figure taken from a XenSource white paper comparing XenEnterprise 3.2 and VMware ESX 3.0.1

> trying to achieve. Finally, the description of the test setup for the VMware benchmarks is incomplete, making it difficult to understand how some results are generated (for example their network performance appears to exceed the bandwidth capacity of a GigE NIC). As a result, the VMware results should be viewed as being of little value. They relate to a VMware specific implementation of a developing open source code base and not to any vendors product.

2.7 Resource utilization

Resource utilization in expensive servers are often at 25% average[35]. As mentioned earlier the practice of assigning one application or service to its own physical server can quickly be costly. A virtual infrastructure improves resource utilization and high priced servers can be justified. Administrators can look at their servers as a big pool of resources and map virtual machines with running services to any part of this resource pool without having to think about available resources on a specific host. This in turn also prevents the proliferation of physical servers.

2.8 Summary

The biggest drawback to the paravirtualization technology is incompatibility. A paravirtualized environment requires modification of the OS to be able to run on the somewhat different virtualized hardware. This results in incompatibility with possibly mission-critical applications and legacy operating systems. In the years of development of the x86 architecture it has allways been backwards compatible which again has resulted in huge amounts of legacy software still in use. This means that giving up compatability is not a trivial undertaking.

However, building a VMM that supports Popek and Goldbergs requirements, and also supports legacy operating systems and its software base, has proven quite a difficult enginering task. VMware, which is the marked leader for x86 virtualization, has done a good job with its on-the-fly binary translator, but academic research projects are favoring the paravirtualized approach[6].

Recent development in the x86 CPU architecture from both AMD and Intel, with their respective Pacifica and Vanderpool technologies, has introduced support for direct execution virtualization. This looks like something that could change the whole virtualization marked as Xen's, and others, paravirtualized environment would no longer require guest host OS modification and VMware's, and others, binary translators would no longer be a necessary mediator for executing privileged instructions like disable interrupts or perform I/O operations.

Virtual machine technology and its applications, as discussed in this work, brings new key features to the table and this time around it seems to be here to stay. The past has shown that hardware supported virtualization can decrease overhead to the point that the value of having a fully compatible virtual machine abstraction overrides any performance benefits gained by the fully compatible binary translation solution.

Chapter 3

Methodology

This chapter will describe the essential pieces of software used to build and manage the virtual infrastructure as well as give a short introduction on how to achieve a remote storage solution supporting live migration of virtual marchines.

3.1 Equipment and Software

All pieces of equipment used during this research period has been provided by Linpro AS. The physical servers, network infrastructure devices and accessories are hosted remotely at Linpro's offices, but has been managed remotely by the author. There is a total of five physical servers where four of them are used for hosting virtual machines. These four servers are addressed by hostname ″nodeX″ where ″X″ is a number ranging from 0 through 3. They have equal specifications, hardware wise, and are installed with the same operating system and complimentary packages required for putting them to use as Xen hosts. The last, fifth, server is addressed by hostname ″clumaster″, which is short for clustermaster. As the name implies, this is the ″master″ of the cluster, acting as both network gateway for the rest of the servers and as the storage backend for any virtual machines running on node0-3. This server has a slightly different hardware specification as table 3.1 shows. The network infrastructure device combining these five servers is a 1000 MBit ethernet switch. As for the power distribution of the physical servers, they are run through an American Power Conversion (APC) rack Power Distribution Unit (rPDU). This device has, among others, a telnet interface which lets an administrator remotely control the ″powerbutton″ of the connected system. This device proved to be very helpful.

Hardware specifications		
	Clumaster	NodeX
CPU	DualCore Intel P4 3 GHz	AMD Sempron 3100+
Memory	2048 MB	1024 MB
Harddrive	2 x WD 150 GB SATA	Samsung 40 GB SATA
NIC	2 x Intel PRO 1000 Mbps	Intel PRO 1000 Mbps

Table 3.1: Hardware specification for the physical servers

3.1.1 ATA over Ethernet

ATA over Ethernet, also known as AoE, is a protocol that encapsulates ATA commands in ethernet packages. What it does essentially is to allow any supported operating system to mount block devices across an ethernet connection seemingly as though the device was attached locally to the physical server. Xen can utilize this to allow live migration of virtual machines as the exported block device is visible on all the nodes in the network. This is one of the solutions available to turn a commodity system into a "home made" Storage Area Network (SAN). However, AoE by itself can not be comapared to a real SAN in any way. The "SAN" used in this thesis does not have any form of redundancy implemented and is not a setup intended to reflect a professional production environment. It is merely a proof-of-concept model intended for research in virtual machine management, monitoring and consolidation. The author suggests the use of for instance DRBD (Distributed Replicated Block Device)[36] to replicate the storage backend to make it redundant. As it is now, any problem inflicting instability on the storage server will result in downtime for all virtual machines using it.

To export an AoE block device the package entitled "vblade" needs to be installed on the storage server. In this case, Ubuntu Dapper Drake, is the operating system of choice and to install the package the following command is used from a terminal:

```
apt-get install vblade
```

This package provides, among others, a binary named "vbladed". The following command is used to export a block device:

```
vbladed 0 0 eth1 /dev/sda1
```

Now to make the shared block device visible to a node in the network, the node needs to have the "AoE" module loaded. The Ubuntu Dapper OS has this module preinstalled, but to import an AoE shared device another set of binaries must be available. The package needed for the importing part is entitled "aoetools". The following commands are used to install the tools needed for importing AoE shares and load the AoE module.

```
apt-get install aoetools
modprobe aoe
```

Once this is done the AoE share can be made visible by using the following command:

```
aoe-discover
aoe-stat
```

Which will result in a similar output as this:

```
e0.0       107.374GB   eth0 up
```

After this the AoE share is made available locally as the device "/dev/etherd/e0.0". The two numbers following "e" in the device name reflect the numbers given when exporting the device from the storage server. These two numbers are, in AoE terms, referred to as "shelf" and "slot".

3.1.2 Logical Volume Management

Logical Volume Manager (LVM) is, as the name describes, a set of tools to manage logical volumes on the Linux operating system. There are two version of LVM

- LVM 1 - The version that is in the 2.4 kernel series
- LVM 2 - New and improved version that uses the device mapper kernel driver introduced in the 2.6 kernel series

LVM (1) is available in many 2.4 linux kernels, but is no longer in active development. Only bugfixes and security updates, if any, will be introduced in the future. It is, however, a mature product and has been considered stable for a few years now and there are still some linux distribution using the 2.4 kernel version. At present, the development is focused on the LVM2 project. This project is almost completely backwards compatible with its predecessor, except for management of volume snapshots. LVM2 uses the device mapper kernel driver which is available in the 2.6 linux kernel series. This project is using the Ubuntu Dapper Drake[1] Linux distribution which uses the 2.6.15 kernel as default.

LVM operates with the following terms to describe different "parts" of the storage area:

[1]www.ubuntu.com

- PV Physical Volume - This is the physical block device
- VG Volume Group - The volume group holds all logical volumes. A volume group can be spread across several physical volumes
- LV Logical Volume - This is the abstraction used in LVM2 to symbolize what could be compared to a partition on a regular harddrive

3.2 Testbed setup

The Linpro virtualization testbed consists of 4 nodes running Xen-unstable and 1 control node, which is both the gateway of the network and the "SAN" host. The Xen-nodes are connected internally over gigabit Ethernet. The Xen-nodes will be addressed as node0 through node3, and the gateway as clumaster, short for clustermaster. These are the hostnames used in the actual testbed and so all examples, figures, screendumps, etc will be coherent with the text. The equipment mentioned in section 3.1 is coupled together in the testbed setup as figure 3.1 shows.

For any virtual machine to be able to migrate to another host, some prerequisites needs to be taken care of:

- Harddrives, either file or block device, needs to be freely available to all potential Xen hosts
- Amount of configured, allocated memory for the virtual machine needs to be available on all potential Xen hosts
- If the virtual machine is a HVM domain, the physical CPU of any potential host needs to support hardware virtualization

The first prerequisite is dealt with by using Clumaster as the storage backend for all virtual machines. As figure 3.2 shows, all harddrives for all the virtual machines are really located on Clumaster, but being visible from all the Xen nodes makes them mountable on any potential migration target. The software used to accomplish this is AoE, see section 3.1.1, and LVM, see section 3.1.2. This will be further explained in the next section.

As for the other prerequisites mentioned previously, memory usage, and free memory, is dynamic on each Xen host as the configuration for each virtual machine differs and how many are running at the same time. Hardware virtualization is not supported by any of the nodes in this testbed and is therefor not considered.

On a sidenote, having the storage server on the same network as the "data"

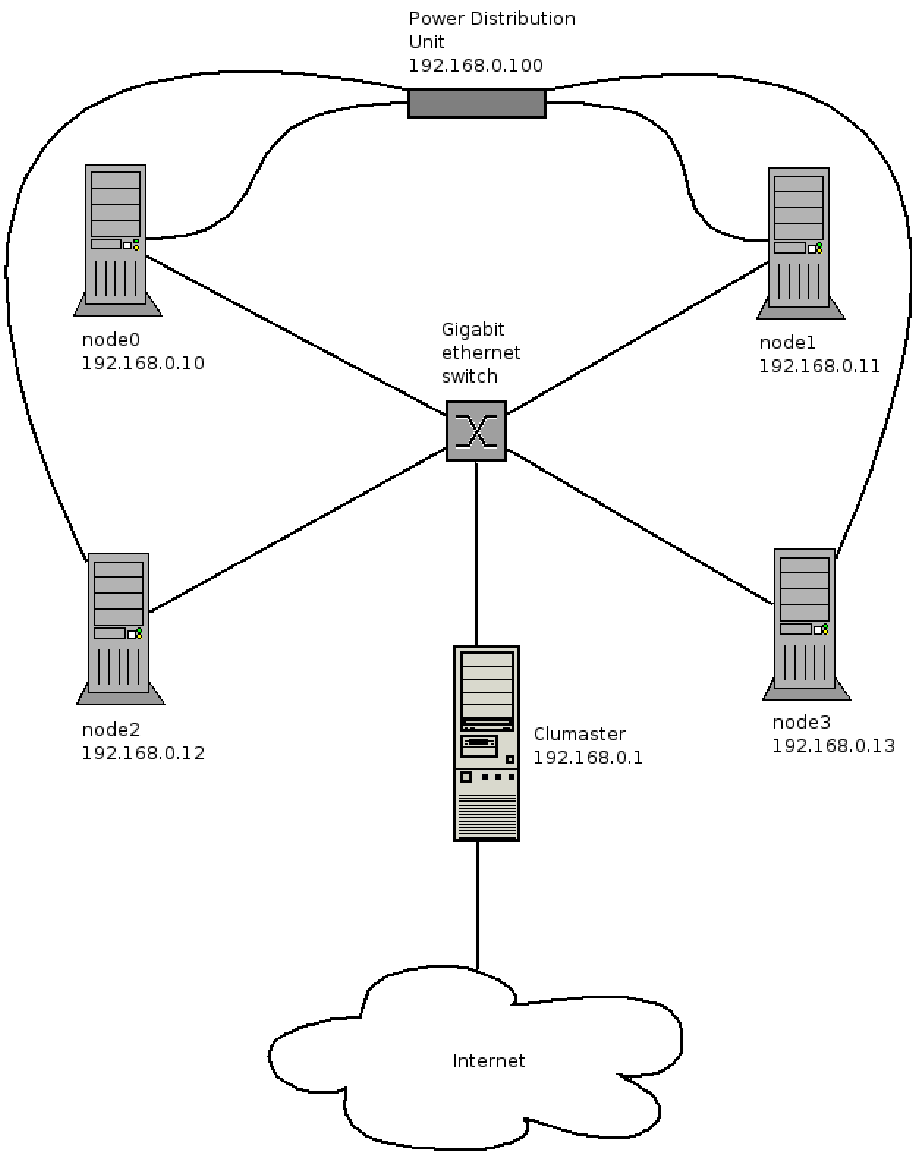

Figure 3.1: Physical network setup

network is not a good idea. Network traffic is generated by all input and output operations on each of the virtual machine's harddrive(s). This added network traffic could act as a limiting factor in both migration speed, and available bandwidth for any public service running on any of the virtual machines. However, the network traffic does not play any part in this work and is only concidered in section 6.9, Future work.

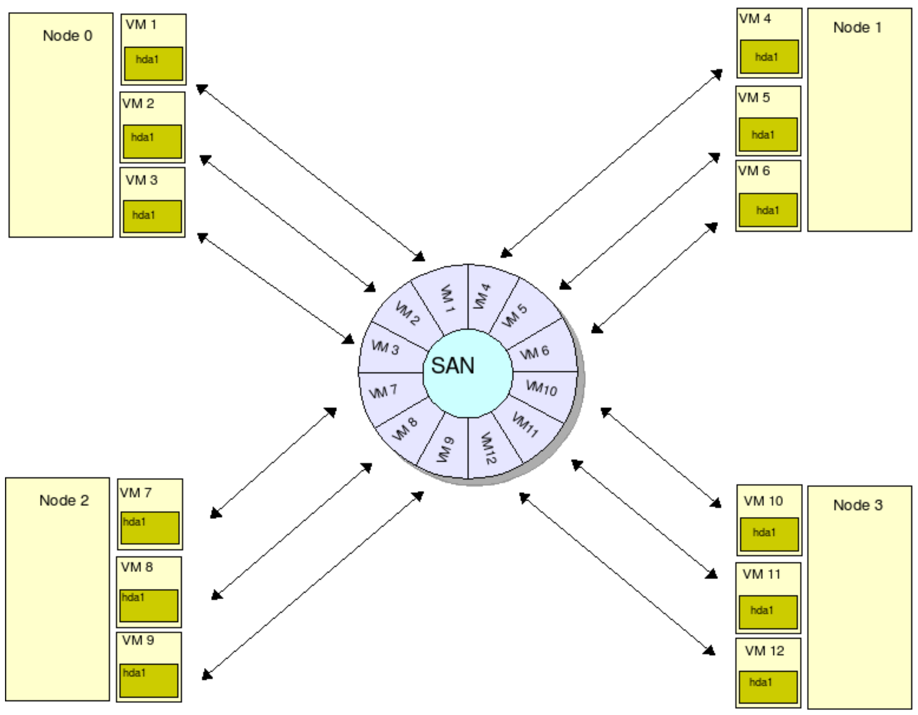

Figure 3.2: SAN illustration

3.2.1 SAN setup

As mentioned in the previous section, the remote storage must be mountable on all xen hosts in order for live migration to be possible. We use AoE to accomplish the block device sharing, and LVM to configure the logical volumes used as each virtual machine's harddrive. AoE, which was discussed in section 3.1.1, presents each shared device as */dev/etherd/eX.Y*, where X and Y represents the AoE *shelf* and *slot*. This shared device is the configured to be a LVM *PV* (Physical Volume) which again can be used to create a LVM *VG* (Volume Group). For this project the volume group *vg_xen2* was created on top of the */dev/etherd/eX.Y* device which was first configured to be a LVM *PV*. The *vg_xen2* volume group was then ”found” each xen server in the infrastructure by issuing the following command on each of the servers:

```
/sbin/vgscan
```

The *vgscan* command produced the following ouput:

```
  Reading all physical volumes.  This may take a while...
  Found volume group "vg_xen2" using metadata type lvm2
```

After the volume group was recognized, logical volumes for each virtual machine could be created seperately on each of the xen servers. Since the same volume group is shared, and used, throughout the infrastructure, each logical volume created is visible, and mountable, from all the xen servers. However, mounting a logical volume which does not have a cluster friendly filesystem, can destroy the filesystem. When a virtual machine is live migrated, it is first unmounted on the source server before it is mounted again on the destination. The xen hypervisor takes care of this.

3.3 Xen API

As the need for more complex network scenarios arise, using a management software like MLN will be more and more valuable in terms of ease of management and configuration. However, the configuration of virtual machines is not the only thing related to a virtual infrastructure which implies the necessity of a management solution. Other important issues to address are for instance redundancy and monitoring. There are several good open and closed source monitoring solutions on the marked, but most of them require some sort of software being installed on each host we want to monitor. As of Xen 3.0.4, XenSource included their prerelease code for a management and monitoring API. The API is meant to take over from its predecessors and be maintained as the only way of communicating with Xend in the future. The following list contains the predecessors of the API and will be phased out in future releases of Xen:

- xend-http-server: Very old and totally broken HTML interface and legacy, generally working SXP-based interface, on port 8000
- xend-unix-server: Same as above, using a Unix domain socket
- xend-unix-xmlrpc-server: Legacy XML-RPC server, over HTTP/Unix, the recommended way to access Xend in 3.0.4
- xend-tcp-xmlrpc-server: Same as above, over TCP, on port 8006

In terms of resource consumption on both Dom0 and all its DomUs, the API is all that is needed to gather the statistics which are generated by Xend itself. The following are some of the metrics that the API produces:

- For each host:
  - Physical CPU utilization float [0,1]
  - Physical NIC utilization kbps in/out
  - Physical MEM utilization total/free
- For each VM:

  - Type PV/HVM
  - Virtual CPU utilization float [0,1]
  - Virtual MEM utilization kbps in/out
  - Virtual HDD utilization kbps in/out
  - Virtual NIC utilization kbps in/out

All values are computed as an average of last two seconds.[2]

Using these metrics, it is possible to give a "calculation" of redundancy[37] which tells us how many of the physical hosts can go down before there is no more free resources to keep all the VMs up. The redundancy metric can be calculated based on different algorithms. The following are suggested by Begnum[37]:

- Least migrations - reduce the risk of a virtual machine dying during the migration process.
- Least memory copied - For a virtual machine to be live migrated, all its memory has to be copied. Less memory reduces migration time.
- Most important last - Some virtual machines are more important than others, risk less by touching these last.

3.4 MLN

To combat the problem of complex configuration files, network preparations and file/partition setup that comes hand in hand with installing a Xen domU, this project has chosen MLN as its companion to take care of the virtual infrastructure management part. MLN, Manage Large Networks[1][2], is, simply put, a tool for administration of virtual machines. It is written in Perl and supports both Xen and User Mode Linux (UML) VMs and is developed with simplicity and long term management in mind. The goal of simplicity is managed by using a "higher" level configuration file with keywords describing the virtual machine in many aspects:

- Virtualization type - UML or Xen
- Network connection(s) - Connect one or more VMs in private networks or connect (bridge) one VM to the existing physical infrastructure
- Network setup - Network interface(s) and their IP address(es)

[2]Information from email correspondance with Ewan Mellor @ XenSource

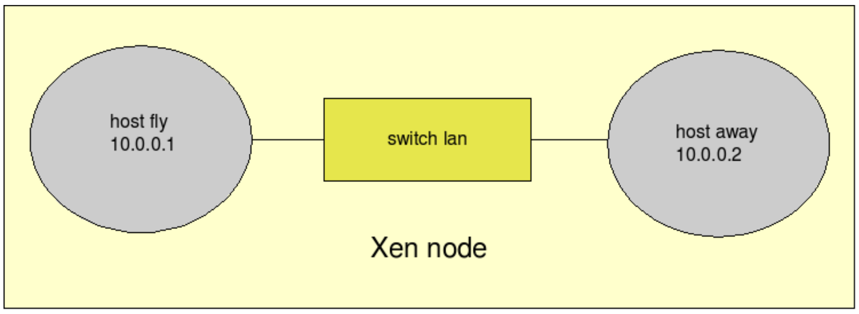

Figure 3.3: 1 MLN project, 1 Virtual switch, 2 Virtual machines

- Users management - Define users, passwords and groups
- Operating system - Many different templates are available as the base operating system installation
- Virtualized hardware specification - Amount of memory, diskspace and CPUs as well as what kind of storage to use as "local" harddrive (local file or partition, or remote block device exported over the network)
- Other configuration - Startup commands and various other configuration of services

These are only a few of the possibilities the high level configuration file offers. The important thing is the ease of creating virtual machines "ready for deployment" and the possibility of connecting them together in customized ways. Again with simplicity in mind, the creation of more complex networks utilizes inheritance inside the configuration file to keep the file short and tidy, yet powerful and simple.

```
global {
        project simple
}
switch lan {
}
host away {
        xen
        network eth0 {
                switch lan
                address 10.0.0.2
                netmask 255.255.255.0
        }
}
host fly {
        xen
        network eth0 {
                switch lan
                address 10.0.0.1
                netmask 255.255.255.0
        }
}
```

The above example shows a very simple project containing two virtual machines in a private network. This configuration has a very limited number of keywords added to it, yet it will create a fully functional private network with two virtual machines connected to it as seen in figure 3.3. A project with the example configuration file as shown will assume the MLN defaults for all the fields not specified. For instance, the virtual machine will utilize a file as its harddive, it will be based on the default template which is a Debian stable minimum installation and it will have only the user root with no password added.

To enable MLN to manage virtual machines across a virtual infrastructure it has a built in daemon feature. Each physical host which has virtual machines running must also run the MLN daemon. The following command, or similar, is used to start MLN in daemon mode:

```
/usr/local/bin/mln daemon 2>&1 > /var/log/mlnd.log &
```

It is possible to query MLN for the status of all virtual machines running on any given host in the virtual infrastructure as long as the MLN daemon is running. The issued command and an example response from the MLN daemons at the Linpro testlab are shown below:

```
cmd: /usr/local/bin/mln daemon_status

Server       #proj #vms Mem.Used Mem.Ava Groups
----------------------------------------------------
192.168.0.10     1   10      416     n/a  n/a
192.168.0.11     2   12      480     n/a  n/a
192.168.0.12   n/a  n/a      n/a     n/a  n/a
192.168.0.13     1   10      416     n/a  n/a
----------------------------------------------------
Total:           4   32     1312     n/a

Hosts that did not answer:
----------------------------------------------------
192.168.0.12: n/a

192.168.0.10  name.project   ID  Mem.Used  VCPU  Status  VCPU.sec  RX.bytes  TX.bytes
192.168.0.10  Domain-0        0       257     1  r-----   25456.8 6082159924010578849
192.168.0.10  away.ft3       60        32     1  -b----     179.1     19408    159435
192.168.0.10  east.ft3       57        32     1  -b----    2423.2         0   1497234
192.168.0.10  fly.ft3        51        32     1  -b----    2615.6   9145739 367759685
192.168.0.10  h1.ft3         53        64     1  -b----    2331.8   2808104 256349399
192.168.0.10  h2.ft3         50        65     1  -b----    2409.5         0   1497234
192.168.0.10  h3.ft3         58        64     1  -b----    2249.4         0   1497234
192.168.0.10  home.ft3       54        32     1  -b----    2346.8         0   1502400
192.168.0.10  north.ft3      52        32     1  -b----    2339.8        28   2999634
192.168.0.10  south.ft3      59        32     1  -b----    2223.8         0   1497234
192.168.0.10  west.ft3       56        32     1  -b----    2172.9         0   1497234
192.168.0.11  name.project   ID  Mem.Used  VCPU  Status  VCPU.sec  RX.bytes  TX.bytes
192.168.0.11  Domain-0        0       257     1  r-----  490702.0   64969673391315244
```

```
192.168.0.11  away.simple    34        32     1  -b----    3810.0 118920760 140988940
192.168.0.11  away.ft32       1        32     1  -b----    5153.7
192.168.0.11  east.ft32       2        32     1  -b----    4970.9       420   2146550
192.168.0.11  fly.simple     35        32     1  -b----    3758.7 239826250 298959232
192.168.0.11  fly.ft32        3        32     1  -b----    5068.3   3953406 901546667
192.168.0.11  h1.ft32         4        64     1  -b----    5165.4   3927420 896721838
192.168.0.11  h2.ft32         5        64     1  -b----    4979.2       364   2146392
192.168.0.11  h3.ft32         6        64     1  -b----    5034.2       336   2146434
192.168.0.11  home.ft32       7        32     1  -b----    4859.7       196   2146728
192.168.0.11  north.ft32      8        32     1  -b----    4915.2       980   4294568
192.168.0.11  south.ft32      9        32     1  -b----    4841.3       476   2146528
192.168.0.11  west.ft32      10        32     1  -b----    4866.9       504   2146822
192.168.0.13  name.project   ID  Mem.Used  VCPU  Status  VCPU.sec  RX.bytes  TX.bytes
192.168.0.13  Domain-0        0       257     1  r-----   34401.5 9755559704166925523
192.168.0.13  away.ft31       1        32     1  -b----    4625.7
192.168.0.13  east.ft31       2        32     1  -b----    4599.5       196   2148388
192.168.0.13  fly.ft31        3        32     1  -b----    4671.6   4005844 217206157
192.168.0.13  h1.ft31         4        64     1  -b----    4815.2   3947632 375336906
192.168.0.13  h2.ft31         5        64     1  -b----    4632.4       196   2148220
192.168.0.13  h3.ft31         6        64     1  -b----    4643.1       196   2148220
192.168.0.13  home.ft31       7        32     1  -b----    4596.0       504   2148804
192.168.0.13  north.ft31      8        32     1  -b----    4615.7      1904   4298588
192.168.0.13  south.ft31      9        32     1  -b----    4540.3       504   2148814
192.168.0.13  west.ft31      10        32     1  -b----    4604.2       504   2148772
```

As we can see from the output there are four Xen servers with ips from 192.168.0.10 to 192.168.0.13 (node 0 through 3) hosting a total of 32 virtual machines from 4 different projects. A project is denoted with:

```
global {
        project simple
}
```

in the configuration file, one project per file. The example project "simple" is running on node1 (ip 192.168.0.11) as we can see in the screendump:

```
192.168.0.11  name.project   ID  Mem.Used  VCPU  Status  VCPU.sec  RX.bytes  TX.bytes
.
.
192.168.0.11  away.simple    34        32     1  -b----    3810.0 118920760 140988940
.
192.168.0.11  fly.simple     35        32     1  -b----    3758.7 239826250 298959232
.
.
```

MLN uses a naming scheme for the virtual machines consisting of the hostname specified in the configuration file and the project name put together as "hostname.projectname". The hostnames are defined in the configuration file, as we saw earlier, like this:

```
 host away {
  .
  .
 }
```

where every virtual machine has its own specific settings defined inside this segment.

As mentioned earlier, the screendump shows that there are four physical servers running the MLN daemon and a total of 32 virtual machines alltogether. However, the command *mln daemon_status* was not run from any of these four servers, but from a fifth "control node". This node does not run the Xensource virtual machine monitor, it is solely used for management purposes. The "management node" has a list of ip addresses of nodes running the MLN daemon, this is how it knows which nodes are in the virtual infrastructure, and the address on which to contact them. The following segment is an example from the configuration file of MLN itself */etc/mln/mln.conf* on the management node:

```
templates /opt/mln/templates
files /opt/mln/files/root/$USER
projects /opt/mln/projects/root
uml /opt/mln/uml
default_kernel /opt/mln/uml/uml-2.6.12-rc2-mm3/linux
default_modules /opt/mln/uml/uml-2.6.12-rc2-mm3
daemon_status_query 192.168.0.10
daemon_status_query 192.168.0.11
daemon_status_query 192.168.0.12
daemon_status_query 192.168.0.13
service_host 192.168.0.1
```

The *daemon_status_query 192.168.0.1[0-3]* lines has the keyword "daemon_status_query" which, as already mentioned, tells MLN about the different physical nodes which hosts the virtual infrastructure. On the other end of the communication, each MLN daemon needs to have a line similar to this:

```
daemon_allow 192.168.0.1
```

to allow for "management" commands to be accepted and dealt with. If this line is missing from the configuration, all queries to the MLN daemon will be dropped. The *service_host* keyword seen in the configuration file screendump of the management host, is also needed in the configuration file of the virtualization nodes. This is to tell each host using MLN which ip address each host should be responsible for. This setting is necessary in distributed management when the same keyword *service_host* is used in a projects configuration file where it symbolizes which physical host should take care of building the virtual machines, and their virtual infrastructure devices if any, in that project. Now to use the example with project simple again, this time the keyword *service_host* is added to "distribute" this project to be built on one of the virtualization nodes instead of the node where the build command is issued.

```
global {
        project simple
$sh = 192.168.0.11
}
switch lan {
service_host $sh
}
```

The variable *$sh* is defined in the *global* block where it can be utilized by any other block in the configuration file. Defining the variable in the global block makes changing the *service_host* option in the project more convenient. Upon building the project, the command is issued from the management node, the "service host" will be contacted to start the build process locally. The following excerpt is an example on building the *simple* project and shows a part of MLNs output referring to the distributed building process.

```
root@clumaster:/home/i/mln-files# mln build -f simple.mln
..
Sending project to 192.168.0.11:34001
Collecting reports from servers...
.
.
```

MLN has recently been further developed to support live migration of virtual machines, however this feature is at the time of writing only available in the beta release. It is the Xen functionality described in section 2.5.4 that MLN now is able to take advantage of. As upon creation, MLN also uses the configuration files to update, upgrade or (live) migrate virtual machines. In order to migrate the project, the only change that needs to be made is to redefine the *$sh* variable in the global block. After that, the following command is issued:

```
mln upgrade -f simple.mln
```

MLN will then proceed to calculate the difference(s) between the stored, and possibly running, configuration of the project, and the newly edited file. If any differences are found, in this case the *service_host* keyword is affected, appropriate actions to adjust the differences will be taken. When *service_host* is changed, it means the project should move to a the new defined host. MLN will try to contact the host with the specified ip address and send the project there.

In order for live migration to work, the file system of the virtual machines needs to be freely available to all possible receiving hosts. Presently, this is another beta feature, MLN supports block device sharing across the network with LVM (Logical Volume Manager, see section 3.1.2) and AoE (ATA over Ethernet, see section 3.1.1).

LVM has been supported in MLN for use as hardrives in VMs for quite awhile. It is not until recently that AoE was utilized as well. For more detail regarding the setup of the shared storage, see section 3.2.1. Using AoE makes all partitions for each virtual machine visible, and mountable, to all Xen nodes. It is the same for each VM itself, hence live migration is possible. To configure MLN to utilize LVM for harddrive storage, something similar to the following is added to the configuration file of MLN:

```
lvm_vg vg_xen2
```

This line states that the VG to be used for new LVs is called "vg_xen2", and hence all LVs on top of it are expected to be found under "/dev/vg_xen2/". This keyword, and feature, however can be used without having a distributed setup such as the one described in this thesis. This introduces the need for another keyword, *san_path*, which is in many ways the same as *lvm_vg*, except this keyword explicitly tells MLN that the VG accompanying this keyword can be found on all MLN nodes. Something like the following is added to MLNs configuration file:

```
san_path vg_xen2
```

Now in order to use the example project "simple" in a live migration scenario, it needs to be configured for using the networked storage instead of local partition or file. The modified configuration of project "simple", fully supporting live migration, is shown below:

```
global {
        project simple
        $sh = 192.168.0.11
}
switch lan {
        service_host $sh
}
superclass {
        hosts {
                lvm
                xen
                service_host $sh
                network eth0 {
                        switch lan
                        netmask 255.255.255.0
                }
        }
}
host away {
        superclass hosts
        network eth0 {
                address 10.0.0.2
        }
}
host fly {
        superclass hosts
        network eth0 {
                address 10.0.0.1
        }
}
```

As mentioned earlier in this section, MLN utilizes inheritance in the configuration file of virtual machines. In the example above, the superclass *hosts* is introduced. As you can see, all the common variables are kept in a "superclass" segment and the separate host config segment inherits the values from the superclass *hosts* with the simple statement *superclass hosts*. At the same time, the network devices are configured individually by putting the address statement in each hosts network settings. This will produce a network with hosts *away* and *fly* connected to a virtual switch "lan" on eth0, they will both use harddrives mounted from a LV, they will both be running on service host *192.168.0.11* and they will have ip addresses *10.0.0.1* and *10.0.0.2* respectively. This private network does not have a connection to the internet nor any part of the existing network, virtual or physical. To connect the private network to another network, for instance the physical network which allows for access to the internet, another network device is added to one of the hosts. This will in effect mean that this virtual machine will act as the gateway for the rest of its private network. The following is an example of a modified host block for the virtual machine *fly*:

```
host fly {
        superclass hosts
        network eth0 {
                address 10.0.0.1
        }
        network eth1 {
                address dhcp
        }
}
```

Since the network block for eth1 on host fly does not contain the keyword *switch*, MLN will assume that this particular network interface card should be connected to the default Xen bridge *xenbr0*. This is the default bridge created by Xen which is connected to the underlying physical network. Hence, host fly will send and receive dhcp requests and offers from a dhcp server either connected to the physical network itself or through the virtualization layer.

The small private network is now connected to the rest of the network, but routing is not in place. This is not something that Xen, itself, deals with, but again MLN proves helpful. By adding a few more keywords to the configuration file we get a fully virtualized, working, private network which is bridged to the physical network and does routing between the two networks:

```
host away {
        superclass hosts
        network eth0 {
                address 10.0.0.2
                gateway 10.0.0.1
        }
}
host fly {
        superclass hosts
        network eth0 {
                address 10.0.0.1
        }
        network eth1 {
                address dhcp
        }
        startup {
                iptables -t nat -A POSTROUTING -o eth1 -j MASQUERADE
                echo 1 > /proc/sys/net/ipv4/ip_forward
        }
}
```

Now if either the dhcp server does not tell the requesting host about the dns servers of the network, or simply overriding the settings is wanted, the keyword *nameserver* is added to the configuration of each virtual machine in the host block similar to the following:

```
host away {
        superclass hosts
        nameserver 192.168.0.100
        network eth0 {
                address 10.0.0.2
                gateway 10.0.0.1
        }
}
```

Chapter 4

Software development

Developing a prototype tool that would be able to deal with virtual infrastructure management issues like load balancing, monitoring and provisioning has been a goal for this research. A tool for monitoring the virtual infrastructure is a prerequisite when doing load balancing based on live migration, i.e. re-provisioning, of virtual machines. In order to monitor a Xen based virtual infrastructure one can either use Xen's built in tools locally on the command line, or use its API which can be reached remotely. A newly introduced, brushed up, API appeared on a testing, not completed, stage in Xen 3.0.4. That version of the API was only intended to be a preview, followed by a final release in the coming version. The next Xen release, 3.1.0, was released in late May 2007 with a fully working finalized Xen-API version 1.0. This is the API that this project based its virtual infrastructure monitoring on, and it is the only interface to Xend that will be backward-compatible maintained in the long term. This very interface allows for the extraction of some, if not all, of the metrics that the xm and xentop programs provide.

4.1 XeniNFO - A Xen API example

The challenges met while testing the API can be summarized as follows:

- Decide on a programming language
- Find a suitable XMLRPC implementation
- Figure out the Xen API by reading source code, unfinished documentation and by testing
- Implement

As mentioned in section 3.3 the Xen API uses XML Remote Procedure Calls (RPC) to communicate with Xend. After deciding on a programming language to use, a suitable xmlrpc implementation had to be found. I ended up using Perl as programming language and so the XML-RPC client module by Randy Ray was chosen for the API interaction. It is freely available from, among other places, CPAN.

When the work of this project started in January 2007, no examples existed for connecting to the Xen API using Perl. For the problems related to the XML-RPCs the use of the interactive python shell "iPython" was used. An interactive shell like this is very suitable for testing before implementation as variables are stored and function calls return output directly in the shell as they are typed.

While having spent quite a few hours on exploring the features of the API with iPython, implementing it in Perl was a different story. Both while testing and implementing a lot of effort had to be put in to mapping out the different RPCs needed to collect the resource consumption statistics needed. This was primarily due to the fact that the API documentation was not coherent with the actual source code and secondly because the API itself was still heavily developed with changesets being submitted to the repository allmost daily. Some of the features that was not yet implemented was listed in the API documentation and vice versa. Differences and bugs, both contributed to making this a tedious process, but the XenSource mailinglist for the API proved to be helpful, more specificly the answers provided by XenSource developer Ewan Mellor. The dialogue with him, both regarding changesets in the API and bugs, was very helpful. The author was also asked to contribute some of this project's work to XenSource for use as an example on how to implement API communication using Perl.

The full script, entitled "XeniNFO", which was contributed to XenSource can be found in section A.1.1.

4.2 MLN Xenguard

The work described in section 4.1 was further developed to make dynamic adjustments to the virtual infrastructure as resource consumption throughout the infrastructure varied. Since the XeniNFO application was contributed to Xensource, the work was forked and renamed to XenGuard. Figure 4.1 shows how XenGuard interacts with the the Xen-API.

The goal of this application was to create a proof-of-concept model which could utilize resource consumption monitoring to make dynamic live migration decisions to provide load balancing for the whole infrastructure. If one

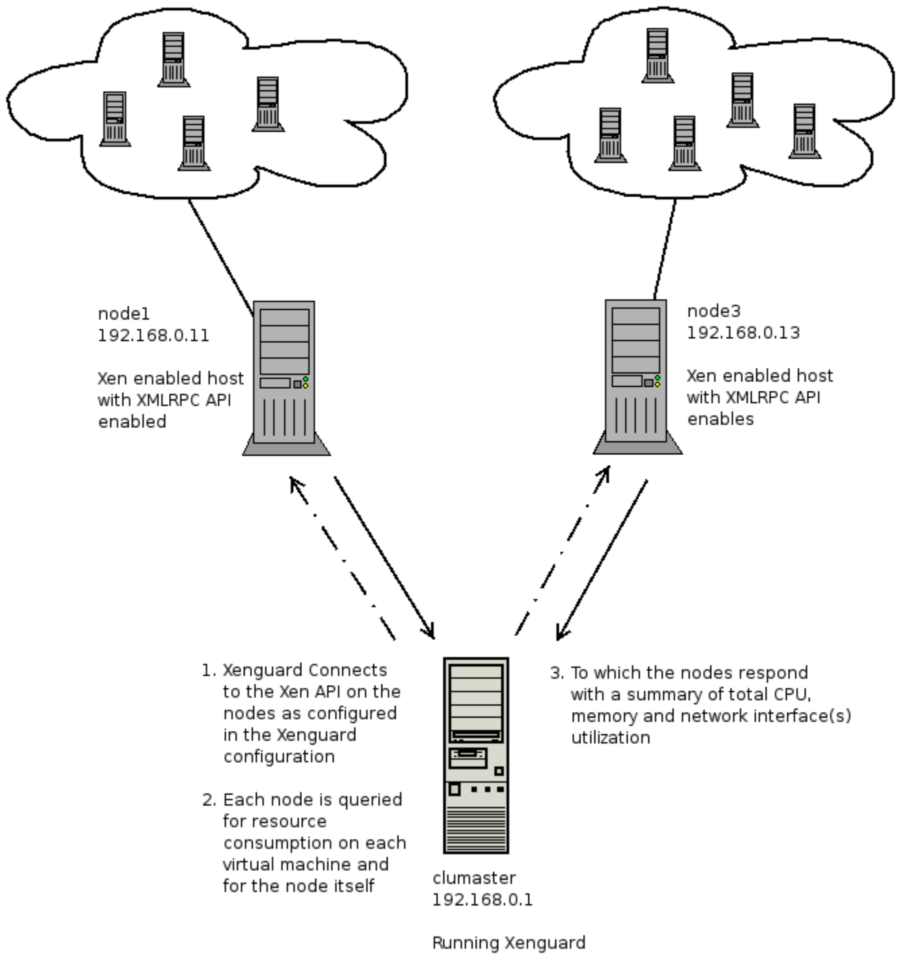
node1
192.168.0.11
Xen enabled host
with XMLRPC API
enabled
node3
192.168.0.13
Xen enabled host
with XMLRPC API
enables
1. Xenguard Connects to the Xen API on the nodes as configured in the Xenguard configuration
2. Each node is queried for resource consumption on each virtual machine and for the node itself
3. To which the nodes respond with a summary of total CPU, memory and network interface(s) utilization
clumaster
192.168.0.1
Running Xenguard

Figure 4.1: Xenguard, running on clumaster, connects to the Xen API on node1 and node3

or more virtual machines consumes more than a configured max level of the total resources on a physical server, live migration would be used as a means to distribute resources more evenly amongst all the virtual machines.

At first, Xenguard was calling MLN with command line arguments to make MLN take care of the reprovisioning of virtual machines. However, after some tests revealed that calling MLN through Xenguard resulted in noticeable execution time overhead, XenGuard was implemented as a command line argument in MLN and added to its source code. Both applications are written in Perl and so this process was rather quick. In order to accomplish this, however, MLN had to undergo some rewriting to make the Xenguard part of it able to utilize existing code from within the application and not provoking reprovisioning through MLN command line arguments only.

In order for XenGuard to be considered for inclusion in MLN, the code had to meet certain quality standards. However, MLN-XenGuard is still at a beta stage and only the future holds the answer to wether or not we will see an official release of MLN including the XenGuard functionality. As of May 2007, future work during summer and fall is planned to make this happen.

The following is a summarized step-by-step list of how XenGuard works:

1. Gather resource consumption statistics from all physical and virtual servers.
2. Evaluate resource consumption and decide if reprovisioning is necessary
3. Calculate possible targets for reprovisioning based on current resource consumption on targets plus added load from all virtual machines to be moved there
4. Find virtual machine dependencies
5. Call appropriate MLN functions to update configuration of virtual machines and their connected networking infrastructure devices to be reprovisioned
6. Call MLN function to upgrade projects changed by the previous command

These steps might be thought to be trivial to implement, however, calculating virtual machine dependencies in context of private networks and their virtual internal networking equipment was not. A MLN project can consist of virtually an indefinite number of virtual machines connected over equally many private networks. What has to be detected is which virtual machines are connected in a private network, and thus needs to be located on the same physical server. One can state that a MLN project can consist of many independent "chunks" of virtual machines, a chunk being a number of virtual machines and their interconnected networking devices. To discover these chunks, a recurring function call is used to loop through all the virtual machines of a project:

1. Find first virtual machine in a project
2. Check if this virtual machine is connected to a virtual networking device
3. If found, try to find (other) hosts connected to this device
4. For each host found to be connected, goto step 2

5. Return complete chunk

As far as the author could find, no other virtual infrastrcuture management tool has the capability to manage, monitor *and* use live migration as a means to distribute resources dynamically across a virtual infrastructure in a load balancing matter. MLN-XenGuard introduces this feature not only for single non-networked virtual machines, but also for private networks of unlimited size. The additional live migration of whole networks enables an administrator to have the flexibility for all non- and networked hosts which has only been seen for single virtual machines in the past.

Chapter 5

Measurements and results

This project is largely based on an open source solution to the combined features of what VMware calls ”VMotion” and ”Distributed Resource Scheduling” (DRS). These features, respectively, are the key words for what can be described as the second and third generation virtualization:

- 1st generation: Servers run on virtualized hardware. More than one operating system can run at the same time on one physical machine creating the abstraction that a single physical server appear as multiple separated ones

- 2nd generation: Virtualized servers are no longer bound to a physical host. While running and without loss of uptime, servers can be relocated to other parts of the physical infrastructure as long as the target location provides enough free resources

- 3rd generation: Automated relocation of running virtual machines based on configurable policy parameters

5.1 Building a MLN project

In this section we will discuss the use of MLN as a virtual infrastructure management tool. More specific, MLN is used to configure, build and start a set of apache webservers based on the Ubuntu server template provided by MLN. If it is not already present, the template of choice needs to be downloaded from the sourceforge webserver. The following command will start the *template download selection* where the user will be prompted to download each of the templates offered on sourceforge.net where MLN is hosted:

```
mln download_templates
```

Each of the templates created by the MLN authors are meant to be building blocks for users that have different requirements for different needs. The templates vary in both make and distribution to support flexible solutions. However, if the operating system of choice is not presented as an option when issuing the template download command above, it is easy to create customized ones based on any make or distribution the user might see fit. Creating customized templates, however, will not be covered in this work. If creating a customized template is the best solution for an administrator, then installing customized software for different services is a natural addition. MLN supports the creation of additional plugins for exactly this need with its plugin API. Plugins can be created, and their feature keywords can be used in the configuration file just as any other "original" options the user chooses. The creation of these plugins will not be covered here, but the example that follows will demonstrate the use of a template with the apache webserver pre-installed and the use of a custom plugin to configure webserver specific options such as *document root* and *domain name*. These options are specified in the *webserver* block for each host in the configuration file:

```
global {
        project simpleweb
        $nameserver = 192.168.0.100
}
superclass {
        hosts {
                lvm
                xen
                free_space 500M
                memory 128M
                nameserver $nameserver
                network eth0 {
                        address dhcp
                }
        }
}
host dagbladet.no {
        superclass hosts
        service_host 192.168.0.10
        webserver{
                domain www.dagbladet.no
                doc_root /var/www
        }
        mount {
                192.168.0.101:/var/www/dagbladet.no /var/www nfs defaults
        }
}
host vg.no {
        superclass hosts
        service_host 192.168.0.11
        webserver{
                domain www.vg.no
                doc_root /var/www
        }
        mount {
                192.168.0.101:/var/www/vg.no /var/www nfs defaults
        }
}
host aftenposten.no {
        superclass hosts
        service_host 192.168.0.13
        webserver{
                domain www.aftenposten.no
                doc_root /var/www
        }
        mount {
                192.168.0.101:/var/www/aftenposten.no /var/www nfs defaults
        }
}
```

This configuration file is a slightly modified version of the *simple* project used in section 3.4. The *simpleweb* project, as this is called, has three virtual machines which are all meant to provide a webserver service each. The three webservers are configured to be hosted on separate physical machines, note the ip address following the *service_host* keyword, which indicates that MLN will distribute the separate parts of the project to the *service_host* indicated. The MLN daemon on these hosts will build the separate virtual machines and configure them as specified by the separate blocks in the configuration file. Figure 5.1 shows the network topology for the current setup. The *mount* block specifies the NFS (Network File System) share which holds the actual content to be hosted by the webserver. Management of this content can be done in any number of ways, this is merely an example of one possible solution. The project is built with the following command:

```
mln build -f simpleweb.mln
```

And upon completion of the building, and configuring, process, the whole project is started with:

```
mln start -p simpleweb
```

This will, ultimately, result in the Xend command *xm create* being called by MLN on the different service hosts defined. The command will be accompanied by the Xen configuration file, created by MLN, for the individual virtual machine, which will cause the booting process to start.

5.1.1 Independent virtual machines

The three web servers is an example on how to quickly provision a fully working set of servers. Fully working meaning allmost no more configuration is

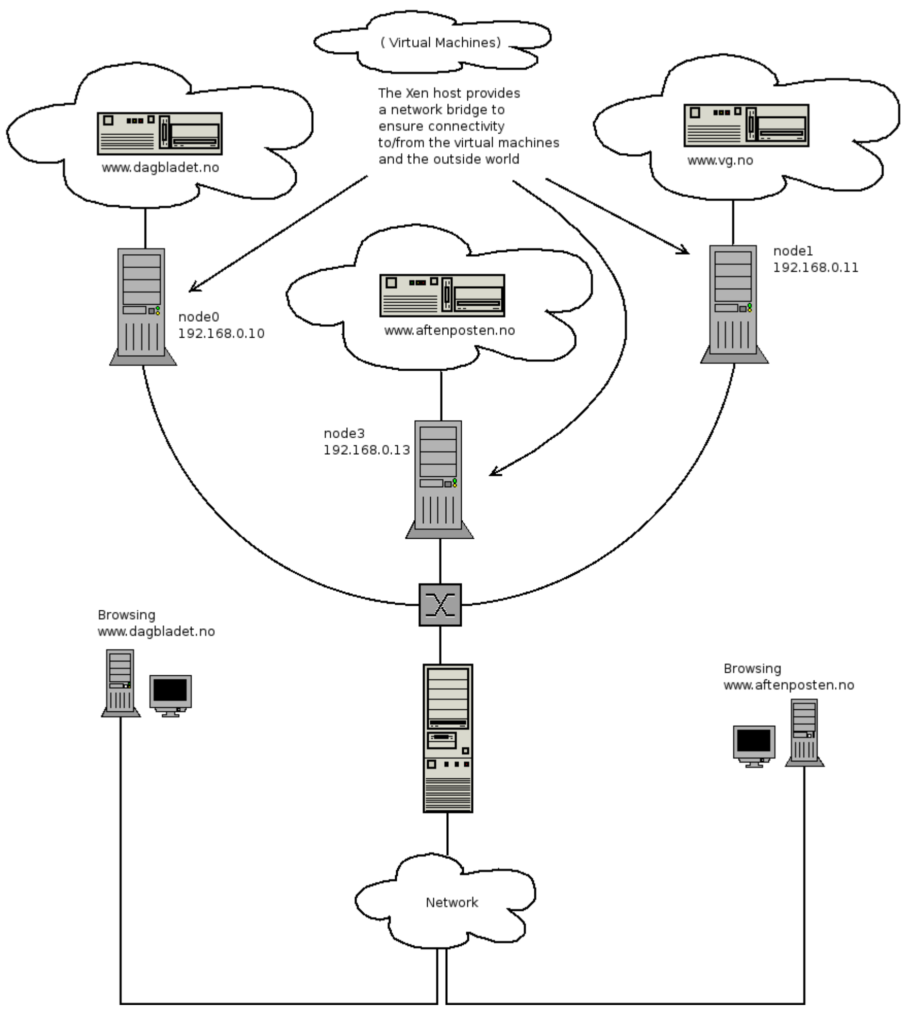
(Virtual Machines)
The Xen host provides a network bridge to ensure connectivity to/from the virtual machines and the outside world
www.dagbladet.no
www.vg.no
www.aftenposten.no
node0 192.168.0.10
node1 192.168.0.11
node3 192.168.0.13
Browsing www.dagbladet.no
Browsing www.aftenposten.no
Network

Figure 5.1: Project simpleweb with 3 Xen hosts each running 1 VM with a specific webserver service

needed before any of these servers could act as a production webserver. This is of course relative to the magnitude of customized tweaking in every aspect of software configuration, but again it also comes down to the complexity, and reach, of customized plugins one chooses to write. The plugin interface allows for customized additions to MLNs configuration files which leads to customized configuration of any templates the plugin is meant for. For example, the two Linux distributions SUSE and Ubuntu might do things differently in regard to where configuration files of the Apache webserver are placed or what version of the software is included, and their respective settings.

The templates are also meant to be user customized. Different companies will have different preferences on what operating systems to use for different services and so they should make their own templates with appropriate software preinstalled. Having such templates, and or plugins, will allow for a speedy provisioning of new servers, decrease time spent on migrating servers or their services and minimize downtime due to maintenance or hardware errors. For example, if usage statistics over time implies that one or more of the webservers from the previous example are being used excessively, another virtual machine serving the same content in a high availability scenario could be brought up instantly if templates and/or plugins are available. Both the dispatcher itself and the webservers it dispatches requests to can be ran as virtual machines. With proper configuration, additional virtual machines can be brought up and connected to the dispatcher on a demand only base. Figure 5.2 shows such a setup with one dispatcher and two webservers hosting the same service.

Provisioning, and configuring, a new server can be time consuming when done with manual labour. Using a management tool, not only to provision, but also to configure the new system, will greatly reduce the total time spent until the new service is up and running. Apart from the webserver example, other virtual machines fitting together, not necessarily by type, but maybe by for instance client/customer, can be put in the same project. Allthough the virtual machines have nothing, directly, to do with each other, it can be convenient to "group" them together in the same configuration. Examples could be a set of gaming servers such as World of Warcraft (WoW), Battlefield or Counter Strike (CS), or a set of servers belonging to a customer such as separated email-, web- and database servers.

Allthough the examples with web- and gaming servers mentioned above are somewhat modest, it does not imply a limitation. MLN has a plugin called *autoenum* which purpose is to easily be able to configure and deploy any number of similar hosts only modifying their *service_host*, *ip address* and *hostname*. This feature can be used to deploy for instance a virtual High Performance Computing (HPC) cluster[4] or an on-demand rendering farm[2]. Large virtual clusters, limited only by subnet size, can be configured and brought up in a fraction of the time it would take to deploy them on physical hardware. The solution is extremely flexible as the cluster state can be saved, taken offline, and brought back up on demand. CPUs of today are seeing a tremendous performance increase in very short timeframes as development improves their inner workings. As such, in an environment where computing power is the key element such as in clusters it is important to keep the underlying hardware updated to maximize performance. If need be, the hardware of the underlying physical nodes in the cluster can be replaced or upgraded without affecting the virtual cluster at all.

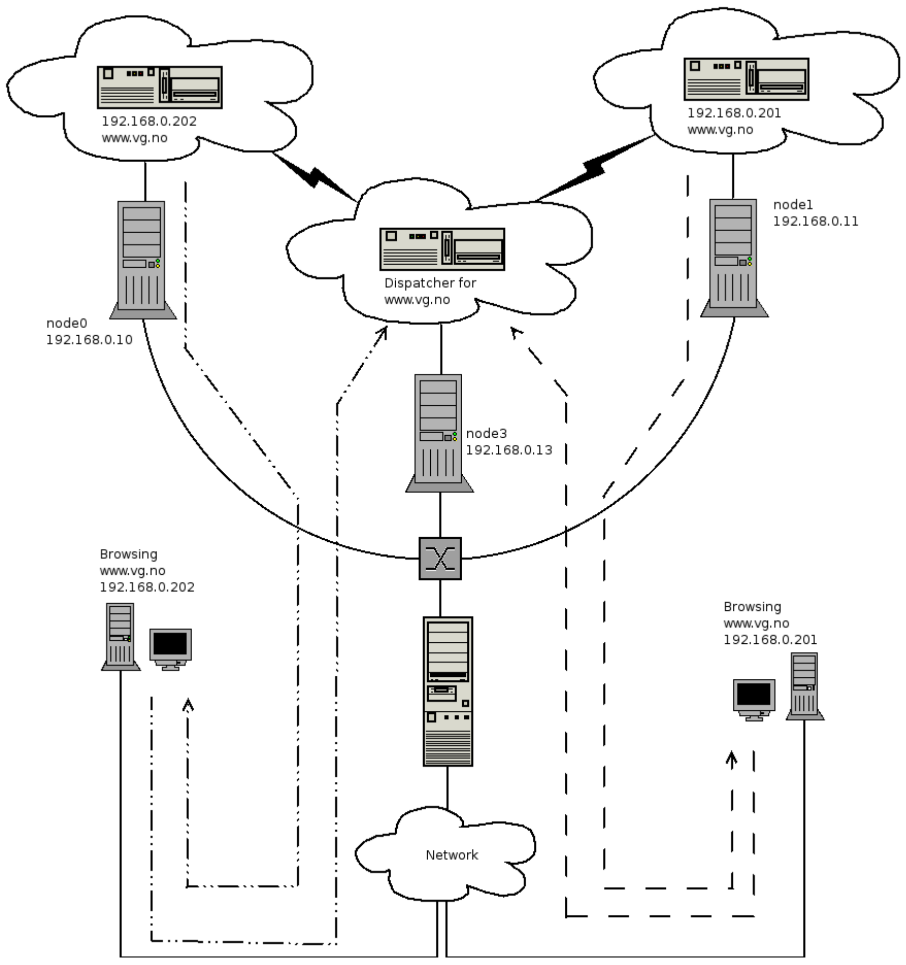

Figure 5.2: Three Xen hosts running a high availability setup with one dispatcher and two webservers on virtual machines

It is important to note that allthough this report goes on to describe more advanced features of virtual networking and management of the virtual infrastructure, the needs and demands of many will stop on independent virtual machines such as the examples described in this section. Xenguard, described in section 4.2, will deal with these independent hosts in the same way it deals with bigger, interconnected groups of virtual machines when it comes to reconfiguration of for instance the *service_host* of each virtual machine. The following sections will talk about building private, isolated networks, which is a feature of MLN (section 3.4), and how Xenguard controls the live migration process of virtual machines based on resource consumption analysis. Section 5.2 will then go on to describe the inner processes of Xenguard in more detail while live migrating bigger, interconnected private networks.

5.1.2 Private networks

In some situations and scenarios virtual, private networking between virtual machines is wanted for higher efficiency, lower latency, less traffic on the physical network or other similar reasons.

In a learning environment, students and their teachers can experiment with networks as big as physical resources allow instead of single virtual machines connected on a network where changes might inflict loss of connectivity and damage to the rest of the physical network. A private network of virtual machines can act as a sandbox for one or more students in a learning environment where practical experience is key to learning. The virtual network, both hosts and network infrastructure devices, can be rebuilt, reset or reconfigured without influencing anyone, or anything, else. Figure 5.3 shows such an example.

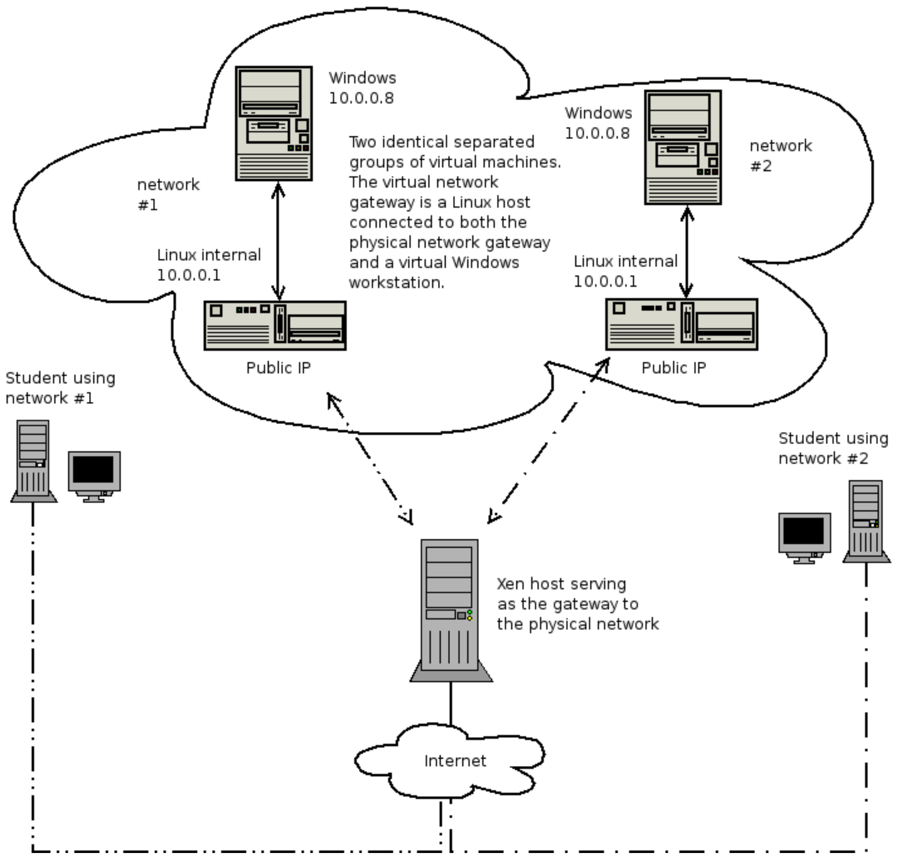

Figure 5.3: Two completely separate networks which can be reconfigured in any way without influencing each other.

Another scenario where private networking can be usefull is in a typical webserver frontend and database backend setup as figure 5.4 shows. The webserver needs to be publicly reachable, while the database does not. In fact, it does not need to be reached by anyone except for the webserver. All network traffic to and from the database server, such as requests and downloads of software updates, configuration management, logging and maintenance sessions will be routed through the frontend node. An example configuration of a small private network with two nodes and internal routing as described in this paragraph can be found in section A.2.1.1.

Currently, a private network with virtual machines, handled by MLN, uses

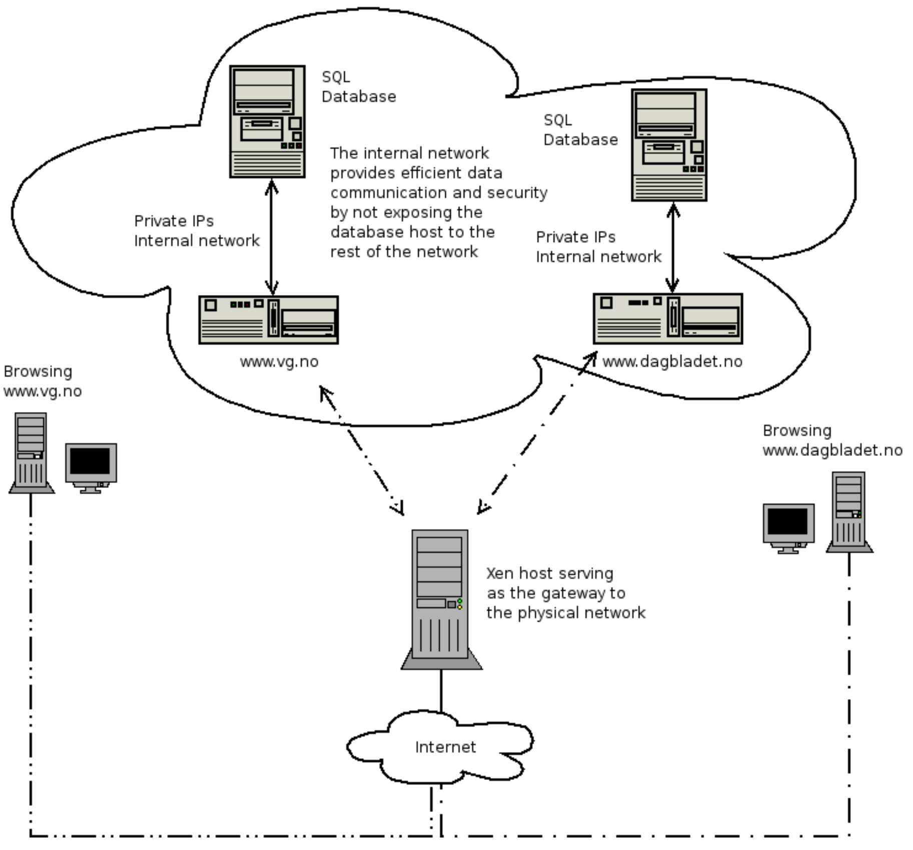
SQL Database
The internal network provides efficient data communication and security by not exposing the database host to the rest of the network
SQL Database
Private IPs Internal network
Private IPs Internal network
www.vg.no
www.dagbladet.no
Browsing www.vg.no
Browsing www.dagbladet.no
Xen host serving as the gateway to the physical network
Internet

Figure 5.4: Two public webservers hosting separate domains supported by a backend database on a private network

the *brctl* command to create virtual switches, or bridges. The snippet of the configuration file for the example project which details the virtual switch is the following:

```
switch {
        lan {
                service_host 192.168.0.11
                bridge lan.simple
                hosts {
                        away eth0
                        fly eth0
                }
        }
}
```

MLN is, at the time of writing, still not officially supporting Virtual Private Network tunnels (VPN), but the code is allmost ready. Configuring VPN tunnels between bridge devices on need would remove the last obstacle of complete resource sharing between virtual machines on private networks. However, it will introduce an additional layer of networking, more potential points of error, re-introduce the private network traffic on to the physical network and at the same time re-introduce the network limitations of the physical network.

As it is today, the virtual machines connected over a virtual network (bridge or virtual switch) has to be on the same physical Xen server in order to communicate over a specific bridge. Inter-VM communication on the virtual network has no networking limits except for available CPU resources and networking drivers used by the guest operating systems. The physical network neither sees, or is impacted by the network traffic running over the virtual networking devices. The fact that the complete network, including the switch and all connected VMs, has to be migrated at the same time, and to the same place to maintain network connectivity, is a downside to virtual private networks. If for instance one of the virtual machines acting as the network gateway, or the bridge device over which the internal networking happens, is migrated to another physical host than the rest of the private network group, it will be as if the switch or network gateway was removed from the physical network. I.e. there will no longer be a network, but a group of (partially connected) hosts without the ability to communicate internally or externally of the ”network”.

When doing virtual machine migration, it is important to have the discussed dependencies related to virtual private networks in mind. A MLN project with more than one private network group is not considered to be dependant of each other in a full mesh, i.e. complete connectivity, kind of way. There are only two dependencies:

- All virtual machines connected to the same private network group has to be on the same physical machine in order to maintain an open communication channel
- Upon (live) migration, there has to be enough available resources on any potential target host for the whole private network

However, separate private networks does not, even if they are in the same project, depend on each other. As seen with standalone virtual machines with bridged networking in project *simpleweb* earlier in this section, the separate networks can be distributed to different service hosts for maximized resource utilization and performance. The resource utilization however, will not be distributed internally in the private network group itself since all elements of this group has to be residing on the same physical host. A set of virtual machines and its interconnected virtual network infrastructure devices, is introduced as a *chunk*. The *chunk* is not limited by for instance subnets, a better way to think of it would be: As far as the virtual network can reach before traffic is sent over physical hosts or networking equipment. With multiple virtual switches (bridge devices), a chunk can consist of unlimited subnets as long as the virtual machines takes care of the internal routing of the virtual private network. The term *chunk* will be used as a descriptor for VMs and bridge devices which depend on each other in order to maintain communication.

5.2 Automated live migration with Xenguard

Xen is undergoing rapid changes and as an opensource solution with its sourcecode repository open for public checkouts, end users can take advantages of newly introduced features, bugfixes and updates at any time they wish. With the introduction of hardware virtualization support in the commodity and server CPU market, Xensource, and their competitors, is working hard to achieve live migration of hardware virtualized domains. This feature, at the time of writing, has not yet been fully introduced, meaning it does work partially in some of the latest *unstable* versions, but not reliably or stable. However, these changes are very recent and this project has based itself on the assumption that the test- virtual infrastructure does not serve any hardware virtualized guests.

This experiment is a typical proof-of-concept test of a scenario where available computing resources are being depleted on the physical host. The project *simple* with hosts *fly* (frontend/gateway node) and *away* (backend node) was introduced for the first time in section 3.4. (See figure 3.3) This experiment will use the same project again. The project could have been a typical setup for a webserver frontend connected to a database on the backend network.

The assumption for the remainder of this experiment is that these two virtual machines could both introduce substantial CPU load through either the database running heavy sql queries or the webserver for instance pulling heavy php/jsp/asp pages. However for simplicity, instead of running these services on *fly* and *away* and generating high CPU utilization through them, excessive resource consumption is introduced in the control domain.

To accomplish this, the control domain (Domain-0) was set to compile the latest revision of the Xen sourcecode. First all source updates are pulled from the repository using the command *hg pull -u* inside the xen-unstable source directory. This command will update all files affected by changes in the source committed after the source was downloaded or the last time it was updated locally. Thereafter the source is compiled using the command *make world*. This command uses all available CPU resources and on a previously idle node in the Linpro virtualization testbed it took 99.46 minutes, i.e. approximately 1.5 hours.

This is a fictional scenario where the control domain itself consumes all available CPU resources, however this project is merely a proof-of-concept that a virtual infrastructure can adapt itself to resource consumption based on policy decisions. The Xenguard script has the following configuration options which acts as policy defined limits to when server relocation needs are apparent:

```
##### CONFIG ######
%xenhosts = ("192.168.0.13" => {"port" => "9363"},
             "192.168.0.11" => {"port" => "9363"});
.
.
$host_cpu_utilization_low = 0.1;
$host_cpu_utilization_high = 0.7;
.
##### CONFIG END ###
```

This configuration is somewhat simple, containing only adjustable parameters for which hosts to monitor and CPU utilization limits to check for. As the configuration reflects, CPU was chosen as the most important parameter to look at when creating this proof-of-concept model. This is not solely attributed to the importance of CPU resources however. The virtualization testbed used in this research does not reflect how a production setup would, or should, look like. Both storage and "real" network traffic runs on the same network equipment and through the same network interfaces on the physical hosts. This makes it hard to isolate, and perform tests, on separate parameters such as I/O intensity or network throughput. In a production environment it would be natural to isolate the storage traffic by installing a second network only for remote storage traffic.

The configuration shows which Xen enabled hosts to monitor in the *%xenhosts* variable. As figure 5.5 shows, two Xen nodes in addition to the control node are used in this experiment.

The following text will explain the logic of this proof-of-concept model. Snippets of MLNs verbose output will be included as appropriate while the progress is detailed:

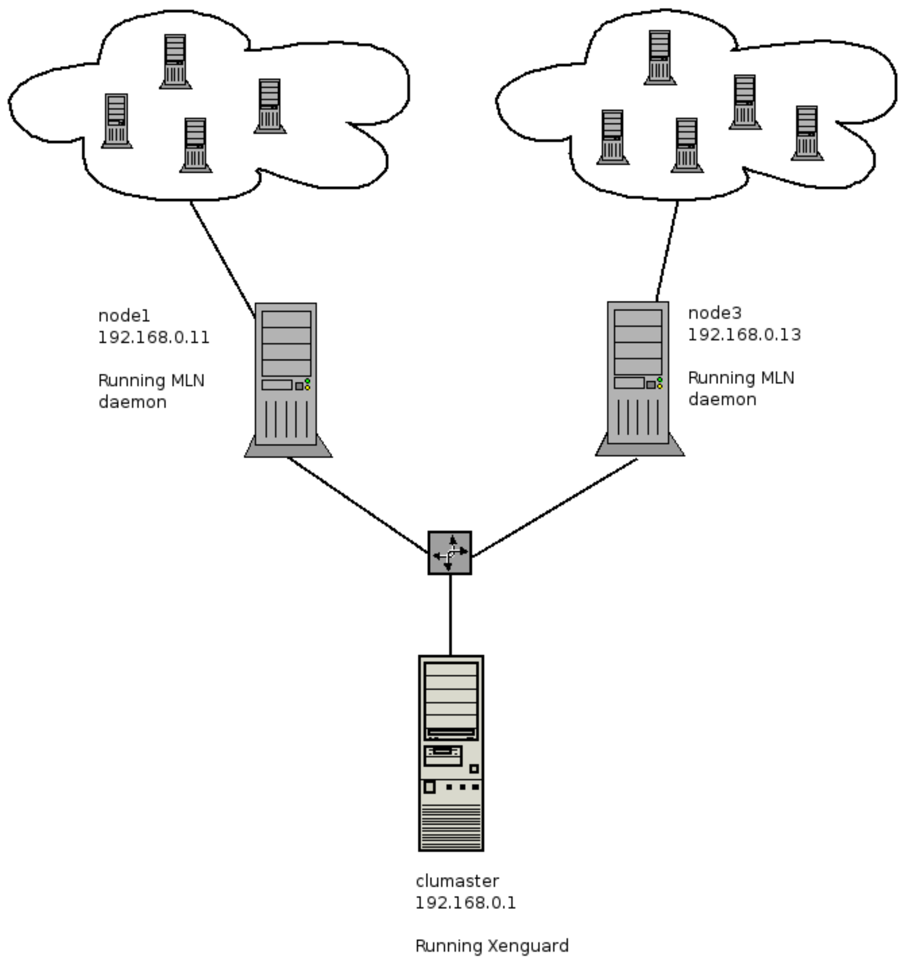

Figure 5.5: The control node, clumaster, is used to monitor the two Xen nodes; node1 and node3

```
root@clumaster:/home/i/tmp# ./mln xenguard
Connected successfully to 192.168.0.13..
Connected successfully to 192.168.0.11..
-----------------------
## 192.168.0.11 ##
-----------------------
     CPUiNFO: 0.962862884660060043
     MEMiNFO: Total: 959.62109375 MB - Free: 204.12109375 MB
     PiFiNFO: peth0 READ: 502.95419299561399 - WRITE: 1014.9075663827996


-----------------------
## 192.168.0.13 ##
-----------------------
     CPUiNFO: 0.0040338969971721256
```

```
    MEMiNFO: Total: 1007.55859375 MB - Free: 315.77734375 MB
    PiFiNFO: peth0 READ: 96.001396199504896 - WRITE: 32.000465399834965
```

The script connects to Xen's API and pulls resource consumption statistics for the physical hosts. The screendump snippet details the current CPU utilization displayed as a number between 0 and 1, total, and free, physical memory in megabytes and transfer rates (RX and TX) for the physical network interface displayed in KiB/s.

```
node1 cpu utilization = 0.96286288466006.. Supporting 2 VMs..
Starting migration logic..
```

This information is then analyzed, and if the cpu utilization is above the defined *$host_cpu_utilization_high* value, measures for possible relocation of the virtual machines are taken. Illustration in figure 5.6

```
Dom0 is using 0.96286288466006, Lets evacuate all possible VMs..
Evacuating all VMs from node1..
We found 192.168.0.13 possible target hosts with less cpu usage than 0.1..
```

Domain0 (the control domain) was found to consume 96% of the available CPU resources. Since this domain can not be migrated off of the physical host, the only course of action in order to make more resources available to the virtual machines is by moving the virtual machines themselves. As the snippet above shows, node3 (192.168.0.13) was found to be a suitable target as it has less that 0.1 CPU utilization.

```
..Found suitable target: 192.168.0.13.. With 315.77734375 MB free mem.. (we need 64 MB..)
Service_host on fly set to 192.168.0.13
Service_host on away set to 192.168.0.13
Service host on lan set to: 192.168.0.13
```

Xenguard, at the time of writing, will only make sure that the two most critical system resources are available before classifying another physical host as a potential recipient for one or more virtual machines:

- Total CPU utilization - If this number (ranging from 0 to 1) is below the configured *$host_cpu_utilization_low* value, it will be added to the potential targets list
- Available memory - After the "low CPU usage" list has been generated, each element will be checked for available memory

The first host that passes both of these requirements will be used as the recipient for the virtual machines. Xenguard then updates the configuration file of the project by inserting the *service_host* keyword in the *host* block for each virtual machine and in the *switch* block for its connected bridge device(s) to point to the newly found target.

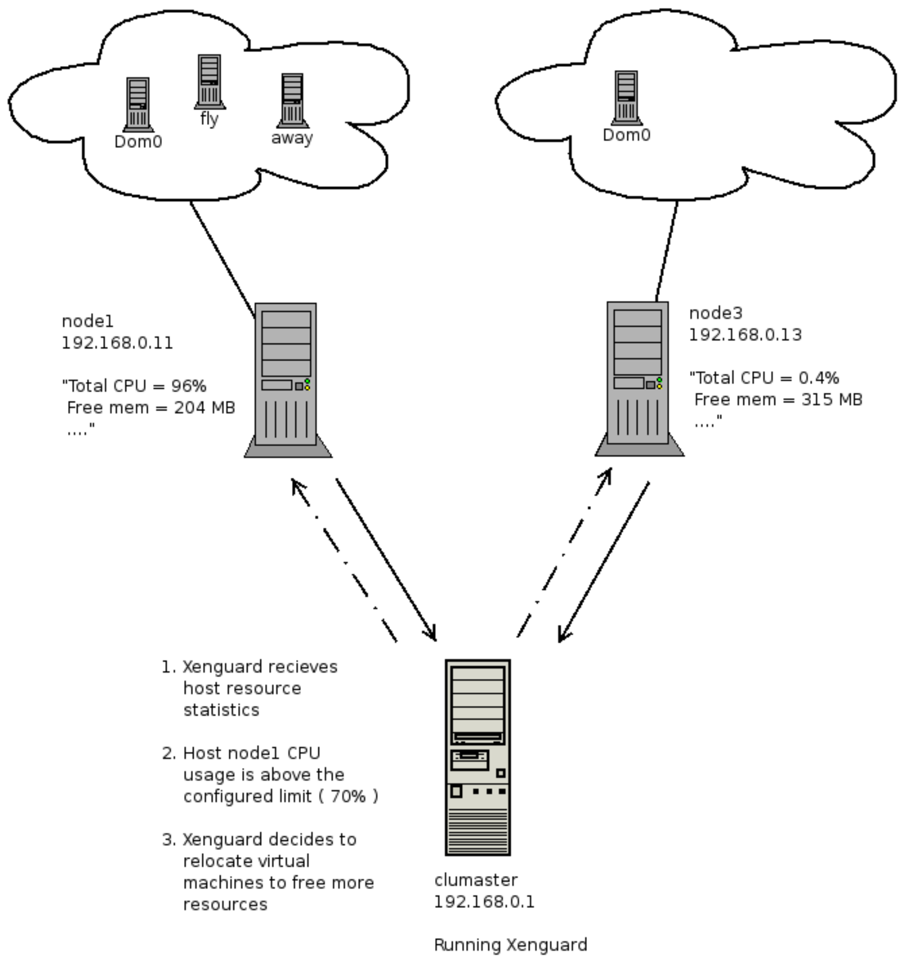
Dom0
fly
away
Dom0
node1
192.168.0.11
"Total CPU = 96%
Free mem = 204 MB
...."
node3
192.168.0.13
"Total CPU = 0.4%
Free mem = 315 MB
...."
1. Xenguard recieves host resource statistics
2. Host node1 CPU usage is above the configured limit (70%)
3. Xenguard decides to relocate virtual machines to free more resources
clumaster
192.168.0.1
Running Xenguard

Figure 5.6: Xenguard, as resource consumption policy dictates, decides to relocate virtual machines away from host node1 to free more resources

```
+---> Upgrade Info:
The Following Diff has been calculated
host {
        fly {
                service_host 192.168.0.13
        }
        away {
                service_host 192.168.0.13
        }
}
+---> UPGRADING simple
- fly will migrate from 192.168.0.11 to 192.168.0.13
- away will migrate from 192.168.0.11 to 192.168.0.13
Sending project to 192.168.0.13:34001
```

```
Done
Saving Config file: /opt/mln/projects/root/simple/simple.mln
```

If any changes were made to a projects configuration file, MLN will accomodate the changes in the best possible way. In this experiment only the *service_host* was changed. As described in section 3.4, this will result in a live migration of the virtual machines as well as the recreation of any virtual networking devices that might have been attached to it. As the snippet above entails, the configuration of hosts *fly* and *away* was changed to reflect the Xen host on ip address ”192.168.0.13” to be their new host and so the MLN daemon is contacted to live migrate the project to its new destination. An illustration can be found in figure 5.7.

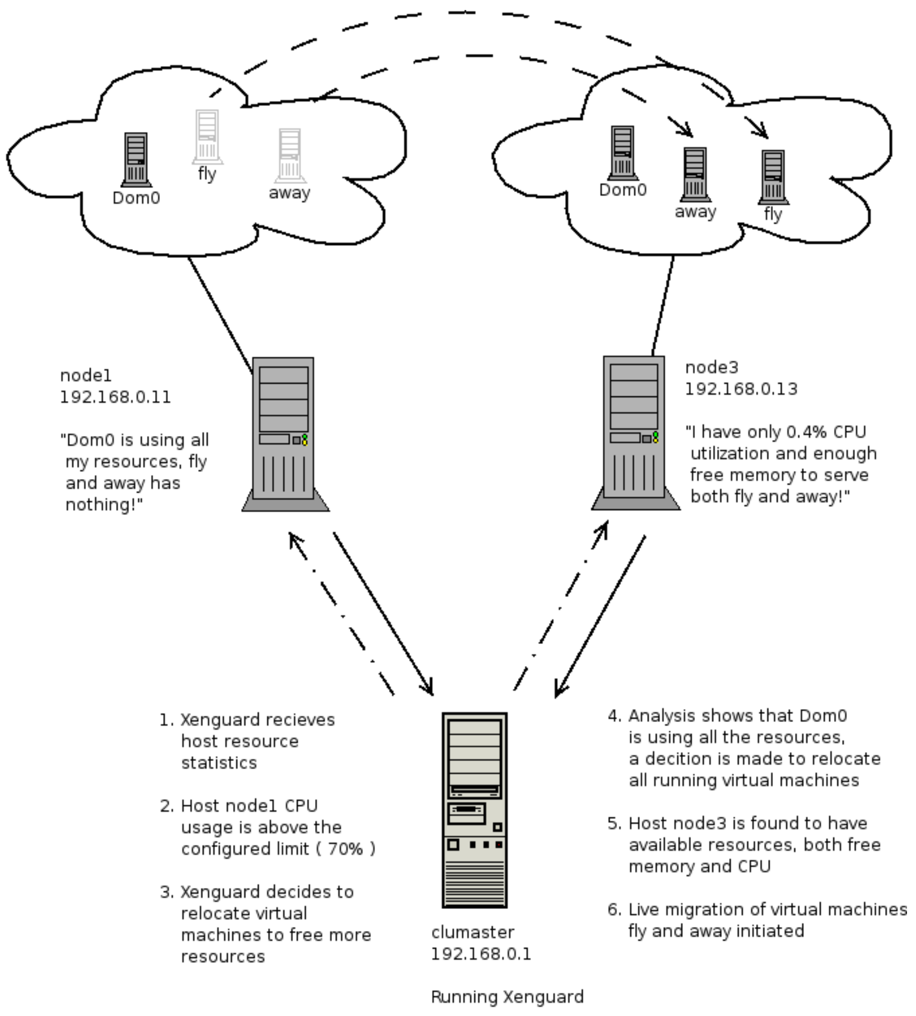
Dom0
fly
away
Dom0
away
fly
node1
192.168.0.11
"Dom0 is using all my resources, fly and away has nothing!"
node3
192.168.0.13
"I have only 0.4% CPU utilization and enough free memory to serve both fly and away!"
1. Xenguard recieves host resource statistics
2. Host node1 CPU usage is above the configured limit (70%)
3. Xenguard decides to relocate virtual machines to free more resources
clumaster
192.168.0.1
Running Xenguard
4. Analysis shows that Dom0 is using all the resources, a decition is made to relocate all running virtual machines
5. Host node3 is found to have available resources, both free memory and CPU
6. Live migration of virtual machines fly and away initiated

Figure 5.7: Virtual machines *fly* and *away* are live migrated from node0 to node3

5.3 Chunk migration

This experiment will take the previous one a step further and at the same time shed light on some of the more detailed internal function calls. The project called *simple*, which has been used up until now, will now be replaced with three identical projects called *ft1*, *ft2* and *ft3*. Each of these projects has a configuration file similar to that of project *simple*, but with 10 networked hosts on three different subnets divided in two separate networks as figure 5.8 shows. The complete configuration file for projects *ftX* can be found in the appendix section A.2.1.2. The configuration related to Xenguard remains similar to the previous experiment except this time all four Xen servers are used:

```
##### CONFIG ######
%xenhosts = ("192.168.0.10" => {"port" => "9363"},
             "192.168.0.11" => {"port" => "9363"},
             "192.168.0.12" => {"port" => "9363"},
             "192.168.0.13" => {"port" => "9363"});
.
.
$host_cpu_utilization_low = 0.1;
$host_cpu_utilization_high = 0.7;
.
##### CONFIG END ###
```

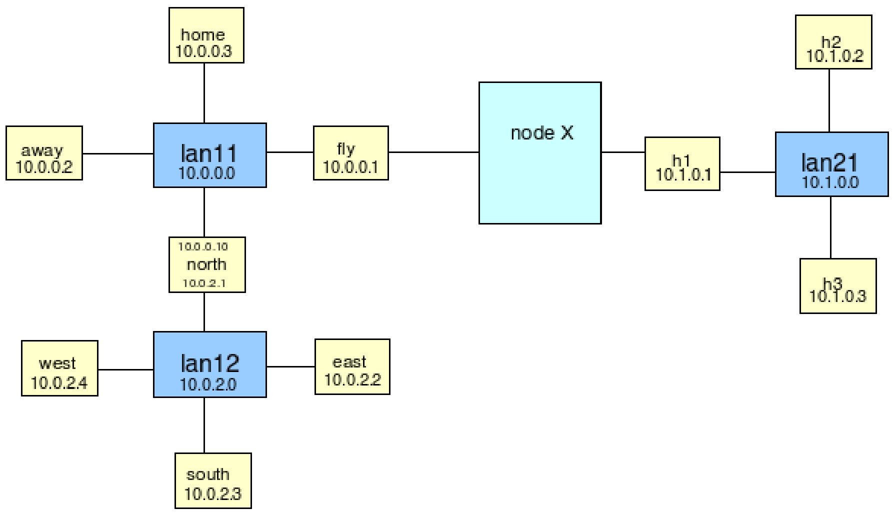

Figure 5.8: Project ft with 10 hosts on 2 networks divided in 3 subnets

As mentioned in section 4.2, there are certain restrictions when creating private networks of virtual machines. Project *ft* has 10 hosts connected over 3 virtual switches, but completely separated in 2 different networks as figure

host / network	lan21 10.1.0.0	lan12 10.0.2.0	lan11 10.0.0.0	chunk #1	chunk #2
h1	x			x	
h2	x			x	
h3	x			x	
home			x		x
fly			x		x
away			x		x
north		x	x		x
east		x			x
south		x			x
west		x			x

Table 5.1: 10 virtual machines divided into 3 subnets and 2 chunks

5.8 shows. All elements in these separated network segments, with both connected hosts and virtual network devices, has to exist on the same physical host. Each of these segments are referred to as *chunks*. Table 5.1 shows which hosts and bridge devices that are connected to each chunk. For each of the different subnets involved in this project there is a bridgedevice. (see table 5.1) Instead of the virtual machine's network interface being bridged directly to the physical network through Xenbr0, hosts on the same subnet are connected to the same bridge device and use only one of the virtual machines as the gateway to reach the rest of the network. Figure 5.8 shows hosts *fly* and *h1* to be gateways to the physical network and host *north* being the bridge between *lan11* and *lan12* (10.0.0.0 and 10.0.2.0). As host *north* connects the two subnets, it means they are both part of the same chunk as they are both dependant on host *north* being alive and forwarding network traffic in order to maintain network communication. As mentioned earlier, all "members" of a chunk needs to exist on the same physical host, which means they all have to be (live) migrated to the same destination. With this in mind, if automated live migrations are to take place as a result of heavy resource consumption, the combined resources used by all members of a chunk needs to be considered when finding a potential target for migration.

This experiment is constructed to force a split of the hosts in the project upon migration as no potential receiving host will have enough free memory to host all the virtual machines. Xenguard will be forced to decide which hosts, and switches, are dependant on each other, construct the chunks, and then, if possible, move each of them to a different physical host. Table 5.2 shows the details of the memory utilization in the *ftX* projects while table 5.3 shows the utilization across the whole virtual infrastructure. If the two tables are compared, it is obvious that both chunks of the project can not fit on the same

host	chunk #1	chunk #2
h1	64 MB	
h2	64 MB	
h3	64 MB	
home		32 MB
fly		32 MB
away		32 MB
north		32 MB
east		32 MB
south		32 MB
west		32 MB
TOTAL	192 MB	224 MB

Table 5.2: Memory usage by hosts in project ft

Server	Projects	VMs	Mem used	Mem free
192.168.0.10	1	10	416 MB	267 MB
192.168.0.11	1	10	416 MB	267 MB
192.168.0.12				
192.168.0.13	1	10	416 MB	267 MB

Table 5.3: Memory usage on physical nodes in the virtual infrastructure

target host upon migration. These details, as well as information about cpu and network usage, can also be found in the output of the *mln daemon_status* command. There is an example of this command in section 3.4 when it has been run while all *ftX* and the *simple* projects were running.

As with the previous experiment, compiling the Xen sourcecode inside the control domain makes sure all available CPU resources are spent. The following text will explain what happens in the background when Xenguard is run, and has to make a decision on which virtual machines to move where. Figure 5.9 shows the complete virtual infrastructure before Xenguard is run.

```
root@clumaster:/home/i# mln xenguard
Connected successfully to 192.168.0.13..
Can't connect to 192.168.0.12 :(
Connected successfully to 192.168.0.11..
Connected successfully to 192.168.0.10..

-----------------------
## 192.168.0.10 ##
-----------------------
```

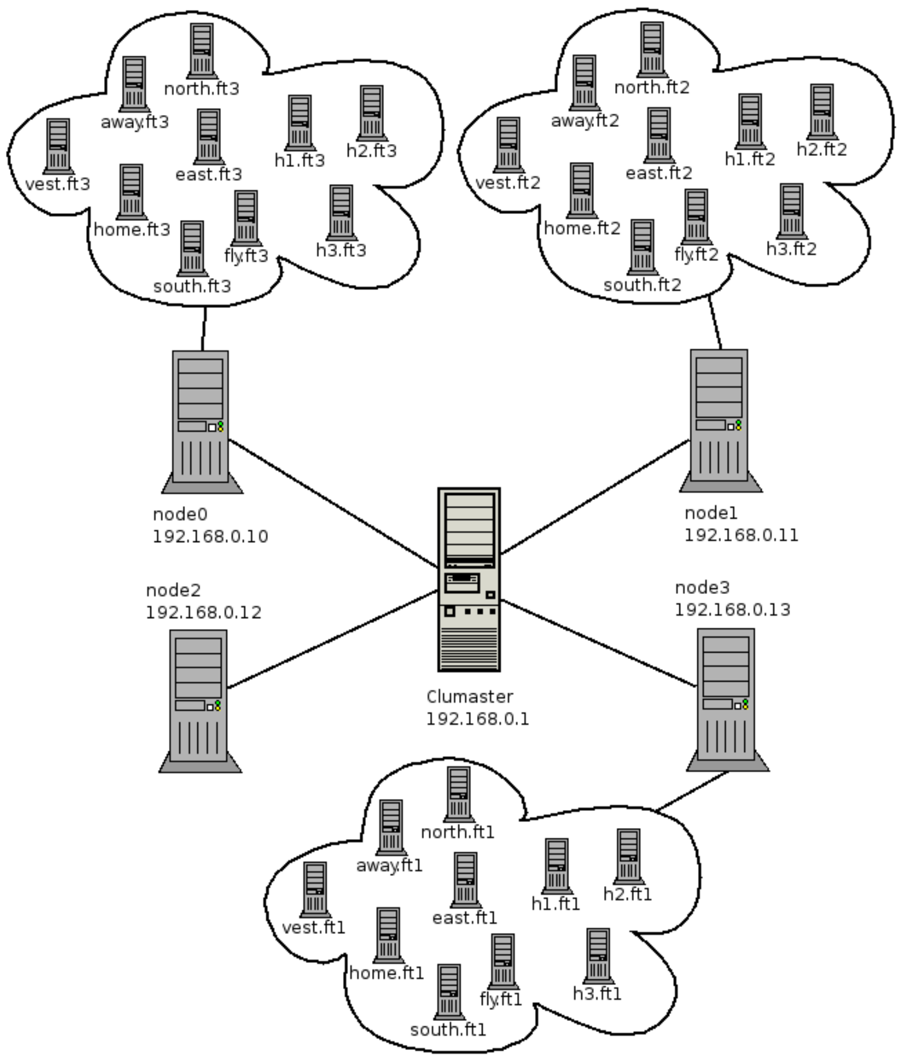

Figure 5.9: The virtual infrastructure consists of 4 Xen enabled hosts supporting 30 virtual machines in 3 MLN projects

```
     CPUiNFO: 0.945633182636637197
     MEMiNFO: Total: 959.62109375 MB - Free: 267.2109375 MB
     PiFiNFO: peth0 READ: 559.59314363205294 - WRITE: 1343.0235447169271


-----------------------
## 192.168.0.11 ##
-----------------------
     CPUiNFO: 0.30627105141596539
     MEMiNFO: Total: 959.62109375 MB - Free: 265.390625 MB
     PiFiNFO: peth0 READ: 69.995869641843711 - WRITE: 133.99209331438652
```

```
-----------------------
## 192.168.0.13 ##
-----------------------
     CPUiNFO: 0.0040290885864005463
     MEMiNFO: Total: 1007.55859375 MB - Free: 315.77734375 MB
     PiFiNFO: peth0 READ: 64.000221253206291 - WRITE: 0.0
```

In this experiment all physical nodes available are used. The host *node2* with ip address 192.168.0.12 is not included due to hardware failure, hence the experiment is constructed with two chunks instead of three to demonstrate the splitting of virtual machines to the remaining two physical hosts when all resources are consumed on *node0*. As shown in the above snippet, Xenguard is unable to connect to *node2*, but continues execution and displays information about the nodes it was able to query. *node0*, where the load has been artificially increased, shows CPU utilization of 94%, *node1* shows 30% utilization while *node3* has 0.4% utilization.

```
node0 cpu utilization = 0.945633182636372.. Supporting 11 VMs..
Starting migration logic..
```

Xenguard will attempt to balance the load distribution if one or more nodes are found to be above the *host_cpu_utilization_high* limit. It finds that *node0* is using excessive resources and starts the process of finding alternative solutions.

```
Looks like h2.ft3 has higher load than 0 with 0.00286752124916157
        which is higher than 0..
Looks like home.ft3 has higher load than h2.ft3 with 0.00316418524360976
        which is higher than 0.00286752124916157..
Looks like Domain-0 has higher load than home.ft3 with 0.945633182636372
        which is higher than 0.00316418524360976..
Dom0 is using 0.945633182636372, Lets evacuate all possible VMs..
Evacuating all VMs from node0..
```

At this point the decision to migrate the virtual machines has been taken and as such the dependencies involved with the migration of whole projects has to be dealt with. Xenguard must determine which of the virtual machines needs to to be on the same physical host after relocation as well as their virtual networking equipment. A recursive function is called to determine which parts of the project is connected and divides these parts into chunks. The result is an associative table where node-, and bridgedevice names are grouped together and tagged with their current *servicehost*. The snippet below shows a part of this datastructure with the two chunks that make up project *ft3*:

```
$VAR1 = 'ft3';
$VAR2 = [
          {
            'servicehostip' => '192.168.0.10',
            'hosts' => [
                         'south',
                         'north',
                         'away',
                         'fly',
                         'home',
                         'west',
                         'east'
                       ],
            'servicehostname' => 'node0',
            'switches' => [
                            'lan12',
                            'lan11'
                          ]
          },
          {
            'servicehostip' => '192.168.0.10',
            'hosts' => [
                         'h1',
                         'h3',
                         'h2'
                       ],
            'servicehostname' => 'node0',
            'switches' => [
                            'lan21'
                          ]
          }
        ];
```

After the "migration logic" is started, Xenguard will try to determine which of the virtual machines is causing the excessive resource consumption. To accomplish this, it will loop through an internal list of statistics per virtual machine based on blocks of information like the following snippet:

```
'home.ft3' => {
        'memory' => {
                'static_min' => '32',
                'static_max' => '40',
                'dynamic_min' => '32',
                'dynamic_max' => '40'
        },
        'vifs' => {
                'eth0' => {
                        'read' => '0.0',
                        'write' => '0.0'
                }
        },
        'vbds' => {
                'hda1' => {
                        'read' => '0.0',
                        'write' => '0.0'
                }
        },
        'vcpus' => {
                '0' => '0.0028142869686770462'
        },
        'type' => 'PV'
},
```

To adjust, if it finds more than one virtual CPU (vCPU), it will add the utilization per vCPU together and divide by number of vCPUs to get a number between 0 and 1. This number is compared to the next virtual machine in the list, and it will recursively go through the list until it has found the virtual machine with the highest resource consumption. Note that the output said "supporting 11 VMs", this is because the control domain (Dom0) is also counted as a virtual machine, but it can not be moved away from its physical host. In this case, as in the previous experiment, Xenguard finds that it is the control domain which is using the most resources. Since it can not be moved, the best thing to do would be to evacuate all possible virtual machines while the control domain finishes the work it is currently doing in order to balance available resources more appropriately.

```
We found 192.168.0.13 possible target hosts with less cpu usage than 0.1..
..Found suitable target: 192.168.0.13.. With 315.77734375 MB free mem.. (we need 193 MB..)
Service_host on h1 set to 192.168.0.13
Service_host on h3 set to 192.168.0.13
Service_host on h2 set to 192.168.0.13
Service host on lan21 set to: 192.168.0.13
this is our local copy
Saving Config file: /home/i/mln-files/ft3.mln
Config saved:
reconfigure chunk completed..
```

As shown in the introductory resource consumption statistics, *node3* was only using 0.4%, i.e. 0.004, which is well below the defined *host_cpu_utilization_low* value and as such that node was, as the snippet above details, chosen as a potential target. Xenguard thereafter continues to check if enough free memory is available for the whole chunk on the target host. In the first pass the chunk with hosts *h1*, *h2* and *h3* was found to utilize 193 MB of total memory which was within the limits of *node3* with its 315 MB free memory. When this is confirmed, Xenguard proceeds to change the *service_host* variable for each host and bridge device in the chunk in the configuration file of the project.

```
DOH! We can't migrate chunk
$VAR1 = [
          'south',
          'north',
          'away',
          'fly',
          'home',
          'west',
          'east'
        ];
.... from node0 to 192.168.0.13 as this host only has 122.77734375 MB and we need 224 MB.. :(
```

node3 was found to have only 122 MB free memory after the reconfiguration of the first chunk which was not enough to support the requirements of the second chunk. This second chunk, as seen above, consists of hosts *north, east, south, west, fly, home* and *away* and has a total memory cost of 224 MB.

```
Evacuating all VMs from node0..
We found 192.168.0.13 possible target hosts with less cpu usage than 0.2..
DOH! We can't migrate chunk
```

Xenguard has so far managed to evacuate the first chunk with the first three hosts and their connected bridge device. In order to migrate the rest of the virtual machines Xenguard searches for potential recipients by adding 10% CPU utilization to the *host_cpu_utilization_low* value. On the second pass, as seen above, Xenguard looks for potential targets with CPU utilization below 20%.

```
Evacuating all VMs from node0..
We found 192.168.0.13 possible target hosts with less cpu usage than 0.3..
DOH! We can't migrate chunk
```

The low utilization parameter is raised again, but the search still yielded only one result and the same result of nothing happening stays the same.

```
Evacuating all VMs from node0..
We found 192.168.0.11 192.168.0.13 possible target hosts with less cpu usage than 0.4..
..Found suitable target: 192.168.0.11.. With 265.390625 MB free mem.. (we need 224 MB..)
Service_host on south set to 192.168.0.11
Service_host on north set to 192.168.0.11
Service_host on away set to 192.168.0.11
Service_host on fly set to 192.168.0.11
Service_host on home set to 192.168.0.11
Service_host on west set to 192.168.0.11
Service_host on east set to 192.168.0.11
Service host on lan12 set to: 192.168.0.11
Service host on lan11 set to: 192.168.0.11
this is our local copy
Saving Config file: /home/i/mln-files/ft3.mln
Config saved:
reconfigure chunk completed..
```

On the forth pass the low CPU utilization parameter is raised yet another time, and this time it is above the CPU utilization of the second host in the virtual infrastructure with ip address "192.168.0.11". *node3* can not receive the last chunk still, but *node1* has 265 MB of free memory which is more than sufficient. All hosts and bridge devices are reconfigured to use "service host" 192.168.0.11 and the project file is saved again.

```
Invoking local upgrade method

+---> UPGRADING ft3
Collecting Status information for ft3
- south will migrate from 192.168.0.10 to 192.168.0.11
- h1 will migrate from 192.168.0.10 to 192.168.0.13
- fly will migrate from 192.168.0.10 to 192.168.0.11
- west will migrate from 192.168.0.10 to 192.168.0.11
- east will migrate from 192.168.0.10 to 192.168.0.11
- h2 will migrate from 192.168.0.10 to 192.168.0.13
- north will migrate from 192.168.0.10 to 192.168.0.11
- away will migrate from 192.168.0.10 to 192.168.0.11
- h3 will migrate from 192.168.0.10 to 192.168.0.13
- home will migrate from 192.168.0.10 to 192.168.0.11
```

All hosts, both chunks, have been reconfigured and its time to execute the relocation. Xenguard calls MLNs internal upgrade function and the live migration process is started.

```
Checking if all MLN daemons are running on new service hosts
.
.
[192.168.0.13] ---> Building switch lan21
[192.168.0.13] h1: creating start and stop scripts
[192.168.0.13] h2: creating start and stop scripts
[192.168.0.13] h3: creating start and stop scripts
.
.
[192.168.0.11] Saving Config file: /opt/mln/projects/root/ft3/ft3.mln
node1 cpu utilization = 0.306271051415965.. Nothing to be done..
node3 cpu utilization = 0.00402908858640055.. Nothing to be done..
```

After first checking if the MLN daemon is running on the selected target host(s), Xenguard proceeds to live migrate the virtual machine to their new physical location. The start and stop scripts which MLN uses in order to start and stop projects are generated, and stored, on the new physical host. Also the bridge device(s) that might have been connected to that chunk must also be created. After all this is accomplished, the updated configuration file is saved on the new location of the chunk. The two remaining hosts, *node1* and *node3*, are checked as well, but as mentioned earlier, and seen above, they have CPU utilization well below the configured threshold and no action is required. Figure 5.10 shows the virtual infrastructure and its running virtual machine after the experiment has been conducted.

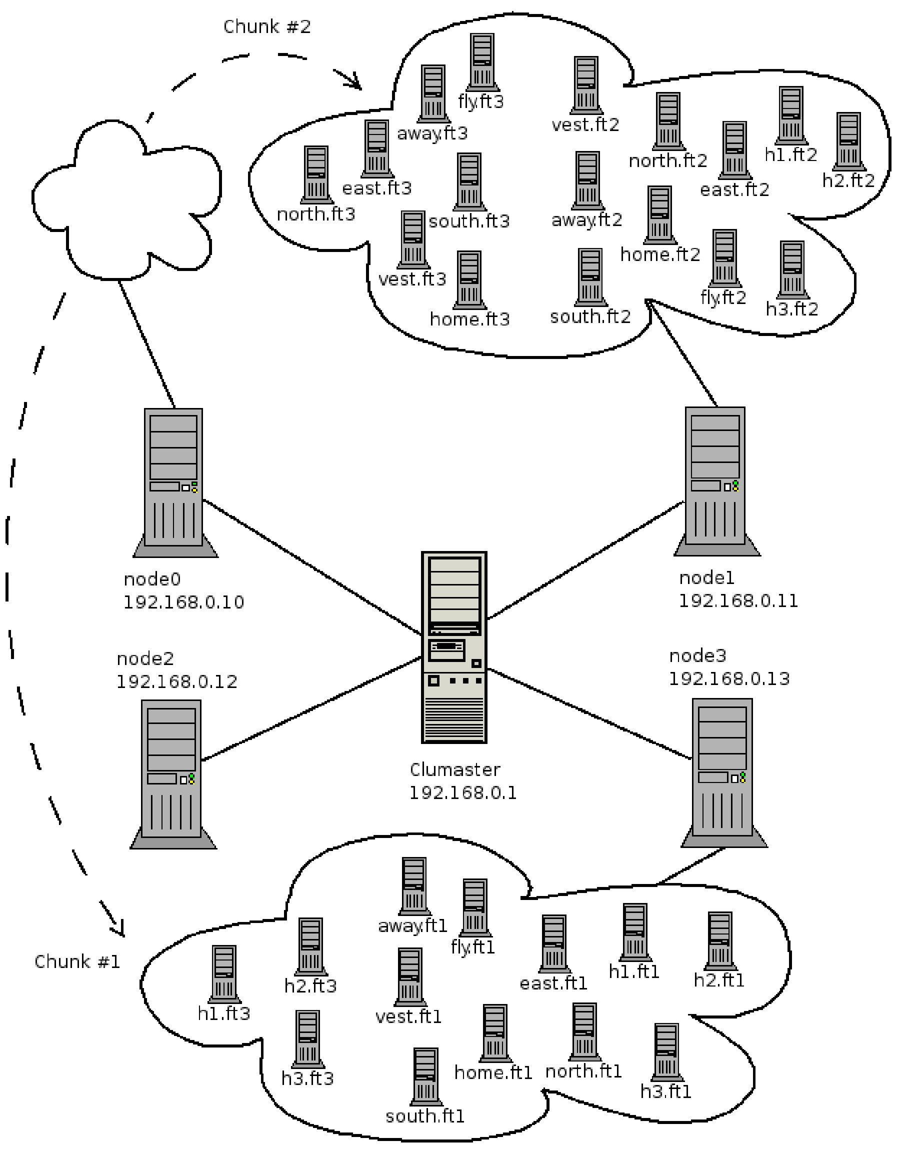

Figure 5.10: The virtual infrastructure and its 30 supported virtual machines after relocation based on resource sharing has been performed by Xenguard

5.4 Chunk complexity VS Network reconnection time

During a live migration, the virtual machine is essentially put in a ”sleep” state which allows for its memory to be copied to the target host on which the virtual machine will resume its responsabilities. The virtual network interfaces of the specific VM, if any, needs to be re-created on the new physical host if the network is to be reconnected upon resurrection of the VM. While the Xen hypervisor takes care of this, MLN takes care of any virtual network infrastructure devices that might be connected to the VM as well. Previous experiments[3] shows that network reconnection time is low enough that users of such services as live radio broadcasts or gameing servers did not notice any connectivity outage. The experiments conducted by E. Braastad, however, only deals with one xen host being live migrated. As shown by the following experiments, the network reconnection time is extended by far when a more complicated private network of xen hosts are to be migrated, and reconnected, at once.

5.4.1 pingtest

For this experiment we have an imaginary setup of two virtual machines connected over a private network in which one of them acts as the gateway to the virtual infrastructure. Upon migration, MLN takes care of the recreation of the virtual switch, while the Xen hypervisor takes care of the virtual network interfaces of the virtual machines. This is an imaginary scenario where for instance we could have a webserver frontend communicating with a database backend over a private network. The reason for doing this could be, separation (and protection) of the database from the webserver and also for faster network communication since the two hosts would communicate over a virtual switch and thus the internal traffic would not be seen outside of the physical host and would not be limited by the specifications of a physical networking device such as a switch or hub. Both virtual machines are based on the Debian-3.0r0-V1.1.ext2 default template provided by MLN and they both have the debian "ssh" package, which contains the OpenSSH client and server, installed.

The following is the result of running the command "ping 192.168.0.1" on host fly. The ip-address being pinged here belongs to the SAN/gateway node "clumaster" which is the network gateway for the whole virtual infrastructure. The host fly is the frontend server in the private network which for instance could be a webserver connected to a database backend over a private network.

```
fly:
64 bytes from 192.168.0.1: icmp_seq=130 ttl=64 time=0.3 ms
64 bytes from 192.168.0.1: icmp_seq=131 ttl=64 time=0.8 ms
64 bytes from 192.168.0.1: icmp_seq=133 ttl=64 time=0.3 ms
64 bytes from 192.168.0.1: icmp_seq=134 ttl=64 time=0.5 ms
```

As we can see here we have a packetloss of exactly 1 packet which makes the network reconnection time for this node approximately 1 second.

The following is the result of running the command "ping 10.0.0.1" on host away. This host is the backend server in the private network which could be for instance a database backend to a webserver.

```
away:
64 bytes from 10.0.0.1: icmp_seq=19 ttl=64 time=0.5 ms
64 bytes from 10.0.0.1: icmp_seq=20 ttl=64 time=0.9 ms
64 bytes from 10.0.0.1: icmp_seq=39 ttl=64 time=0.5 ms
64 bytes from 10.0.0.1: icmp_seq=40 ttl=64 time=0.5 ms
```

We can see that the packetloss for this host has increased noteworthy with as much as 18 ICMP echo requests being unanswered.

The following is the result of running the command "tcpdump -i eth0" on host fly.

```
fly:
19:40:44.240042 IP away > fly: icmp 64: echo request seq 5120
19:40:44.240109 IP fly > away: icmp 64: echo reply seq 5120
19:41:03.119057 IP away > fly: icmp 64: echo request seq 9984
19:41:03.119126 IP fly > away: icmp 64: echo reply seq 9984
```

As we can see there is a gap in the packet capturing of 19 seconds spanning from 19:40:44 until 19:41:03. This means that the "backend" host away is not completely reconnected to the private network until after 19 seconds have passed making the total network reconnection time being not 1 second, but 19 for the whole network.

Towards the end of this project, the author was made aware of the Spanning Tree Protocol (STP) used in Linux bridge devices. The default settings for a newly created bridgedevice can be displayed using the following command:

```
 brctl showstp <name of bridge device>
```

The output of this command ran on one of the bridge devices from the *ft3* MLN project is shown below:

```
root@node3:~# brctl showstp lan.ft3

lan2.ft3
 bridge id              8000.feffffffffff
 designated root        8000.feffffffffff
 root port                 0                    path cost                  0
 max age                  20.00                 bridge max age            20.00
 hello time                2.00                 bridge hello time          2.00
 forward delay            15.00                 bridge forward delay      15.00
 ageing time             300.00
 hello timer               1.40                 tcn timer                  0.00
 topology change timer     0.00                 gc timer                   0.05
 flags
```

The important fields to note here are the *bridge forward delay* and *bridge hello time* entries. As seen from the above snippet the hello time is set to *2.00* while the forward delay is set to *15.00*. Added together the total becomes *17.00* (seconds) of wait before any packet forwarding is done. The author draws the conclusion that these two settings make out 17 of the total 19 seconds of the total network reconnection time. When factored in that the live migration itself takes some time as well as the bridge device has to be created, the conclusion must be that the default settings of the bridge device creation is what generates the extended total network reconnection time.

Further inquiries on this problem was not done as the management tool that creates the bridgedevices must be used to set the values of the bridgedevices in order for the whole process to be automated. This, in turn, would require added / modified code in the management tool.

Chapter 6

Discussion and Conclusion

In chapter 1, Introduction, three main challenges or points of interest were defined:

- Configuration and change management for virtual- machines and infrastructure
- Focus on open source
- Automated load balancing for a virtual infrastructure based on resource consumption statistics

These challenges have been met and the work has been discussed extensively in the previous chapters. This chapter will summarize and briefly discuss the following points, and last but not least give a conclusion and summary of the entire project work.

1. Technical challenges
2. Remote storage solutions in context of live migration of Xen virtual machines
3. Virtual infrastructure management
4. This projects involvement in open source development
5. Xen hypervisor interaction
6. XeniNFO and XenGuard development
7. An analysis of the XenGuard logic and lab results
8. XenGuard and high availability
9. Future work and further development suggestions

Sections 6.1 and 6.2 will cover the preface, section 6.3 will cover the virtual infrastructure management, section 6.4 will cover the focus on open source contribution while sections 6.5, 6.6, 6.7 and 6.8 will cover the development, and usage, of the tools that are used to do automated load balancing for the virtual infrastructure. Section 6.9 will then cover future work before section 6.10 summarizes and concludes the total work and experiment results during this project period.

6.1 Technical challenges

In order to complete the goals of this project a few prerequisites had to be in place, namely four ”xenified” servers and an additional server meant to be used as remote storage as well as the management and monitoring node of the setup. The Xen-unstable release, which is a continuous work-in-progress, was chosen to be used in this project as the latest official release, Xen 3.0.4, only had a preview release of the new standardized Xen-API. This, in turn, meant that the source code had to be mirrored to the testbed and compiled locally before it could be installed on the Xen servers. In previous work[3], GNBD (Global Network Block Device)[1] was used for remote storage. Compiling the GNBD module for the latest Xen-unstable release, as of Feb 2007, proved to be difficult and after much time was spent, the effort was discontinued. Time restraints suggested to try a different solution, and so using AoE (ATA over Ethernet) combined with LVM (Logical Volume Manager)[38] became the solution. The Xen-unstable release already contained the AoE module, hence this solution was easily deployed. Other alternative solutions will be discussed more in the next section.

6.2 Remote Storage

In section 6.1, some experienced difficulties that occurred while installing and configuring the virtual infrastructure were discussed. There are several solutions for remote storage that could have been tried, and probably used, for the testbed infrastructure. However, the challenges this project is trying to address, and the experiments conducted, was based on a experimental setup and does not reflect how a production environment would, or should, look like. The solutions introduced are merely proof-of-concept ideas which are not influenced by a less-than-optimal remote storage solution.

In a production environment, first of all, we would want redundant and highly available storage without any single point of failure. With an enterprise SAN

[1]http://sourceware.org/cluster/gnbd/

solution also comes an ″enterprise solution″ pricetag. For an as-good-as-it-gets ″home made″ remote storage with modest hardware costs involved the author would suggest DRBD (Distributed Replicated Block Device)[36] for the redundant storage as well as heartbeat and a redundant set of dispatcher servers and network equipment to process the I/O operations of the remote storage. DRBD is a piece of software that mirrors any storage device between two or more servers so that in the case of one storage server going down a completely mirrored storage server would be able to take over and process all I/O operations without the virtual machines knowing about it or any operations being interrupted.

Apart from the redundant storage itself, the software that actually shares the devices across the network is merely one of many possibilities. Another open source solution which has gotten substantial publicity is open-iscsi[2] which is based on the iSCSI protocol.[39] Open-iscsi, in effect, does allmost the same as AoE; sharing a block device across the network. However, AoE is limited by the hardware layer, i.e. switching, while iSCSI shares the block device across an IP routed network. XenEnterprise, which is XenSource's commercial enterprise version of the Xen hypervisor, includes iSCSI support for remote storage and live migration for its virtual machines.[40]

6.3 Virtual Infrastructure Management

Virtual infrastructure management is not necessarily a complex task, it all boils down to the number of physical and virtual machines to control. This project, at max, had 5 physical servers with 40 virtual machines running. Unfortunately, one of the ″Xenified″ servers died in the middle of the project period resulting in a maximum of 30 virtual machines hosted on 3 physical servers with remote storage connected from the last physical host running at the same time. This was also the number of virtual machines used in the most extensive experiment.

Managing 30, let alone 40, virtual machines divided over a set of physical servers require structured management to maintain uptime. Having a management software which can configure, provision and maintain any number of virtual machines is very important, and usefull, in terms of both scalability of the infrastructure and management costs. As mentioned in section 1.3, there are many alternatives both commercial and open source to perform such a task. MLN (Manage Large Networks) is one of them. This tool is written in the Perl programming language, which the author is familiar with. One of

[2]http://www.open-iscsi.org/

its two main programmers works at OUC (Oslo University Collage), which proved to be very helpful. The author had some experience with it from the virtualization lab at OUC, and last but not least it is fully command line based which was necessary for XenGuard to utilize its functionality. These were all factors that contributed to the selection of MLN as the infrastructure management tool.

MLN makes management easy by using very high level, easy to understand, configuration files for its virtual machines as well as templates for various operating systems. The configuration files can contain a lot of options which are not set by Xen itself, such as hostname, users and passwords, routing table, firewall rules and even service specific configuration such as for instance *www_root* for webservers. The templates makes bringing up new services very easy as one can preconfigure and preinstall software needed for certain services and then use MLNs configuration files to do the last configuration upon (re-)provisioning of a virtual machine.

6.4 Contributing to Open Source development

Contributing to open source development has been a strong motivator for this project. Both the virtual infrastructure management tool (MLN) and the Xen hypervisor itself are renowned open source projects used by many teaching institutions and companies worldwide. A variety of commercial products exist for a lot of what has been done in this project, but with a conciderable pricetag attached. Open source projects such as Linux itself, the MySQL database and the Apache webserver are all well known products of a ever growing trend of sharing experience and expertice in development of equal, and sometimes better, competitive solutions to well known commercial products. XeniNFO and XenGuard can hopefully contribute, and inspire, others to continue the development of other similar tools which again will aid the opensource community in competing against the commercial products in virtual machine management.

This project has produced two tools; XeniNFO which can already be found in the Xen-unstable repository as a Xen-API interaction example written in Perl:

```
XeniNFO:
http://folk.uio.no/ingardm/thesis/xensource.xeninfo.pl
http://xenbits.xensource.com/xen-unstable.hg?rev/6145e5508d6b
http://xenbits.xensource.com/xen-unstable.hg?diff/6145e5508d6b/
        tools/examples/xeninfo.pl
```

This work was also discussed by Ewan Mellor in his talk about the Xen API at the Xen Summit conference of April 2007 in New York:

```
Xen Summit:
http://www.xensource.com/xen/xensummit.html
http://www.xensource.com/files/xensummit_4/
        XenSummit_API_Slides_2007-04-18_Ewan.pdf
```

The second part of this project, XenGuard, has been contributed to, and implemented in, the MLN management solution for virtual machines:

```
MLN official release:
http://mln.sourceforge.net/

MLN-XenGuard (beta, not official):
Will be found in a future release of MLN
Temporary: http://folk.uio.no/ingardm/thesis/mln-xenguard
```

Apart from these products, this project while using the Xen-unstable beta version of the Xen-API found, and solved the following bug:

```
Xen-API bugfix: Fix VM_metrics.get_vcpus_utilisation.
http://xenbits.xensource.com/xen-3.1-testing.hg?rev/1e592e4557c1
```

A quick run-through of the prices VMware operates at is a strong signal that such a feature that Xenguard introduces in the opensource community is strongly needed. VMware Infrastructure Enterprise (USD 5750 for 2 processors + support), VMware VirtualCenter Management Server (USD 5000 per VirtualCenter application instance + support), VMware VMotion add-on (USD 1400 for 2 processors + support) and VMware DRS (USD 2000 for 2 processors + support) is the total cost on a software setup that would allow automated resource balancing to become reality in a VMware powered virtual infrastructure.[41]

6.5 Hypervisor interaction

This project was focused primarily on the Xensource Xen hypervisor, but the virtual infrastructure monitoring and migration analysis applies to all virtualization technologies that allows for live migration of guests. Xenguard was designed to connect to the Xen hypervisor using its newly introduced API with XML-RPC calls.

Further development will not require much additional changes in the program logic to support other VMMs such as VMware's ESX server, Microsoft's Windows Server Virtualization or Kernel-based Virtual Machines (KVM) as the analysis of resource consumption will stay the same. However, the process of interacting with these VMMs will have to be added to the XenGuard code as each has its own API and different function calls to gather the information needed. This is where the LibVirt[3] library could be a solution. LibVirt can be looked at as an additional layer between the program code and the product API that standardizes the communication between them and by only changing certain parameters allows for communication with the different APIs with equal function calls.

This is first and foremost a C library for interaction with the Xen hypervisor, but both Perl and Python bindings exists. If the bindings to the LibVirt library was used consequently, then supporting other virtualization technologies, as LibVirt is further developed, would require little or no changes to the Xenguard code. LibVirt currently supports Xen, QEmu and KVM virtualization and products such as for instance the RedHat Virt-Manager is based on this library.

Xenguard, however, as developed for the Xen hypervisor, connects directly to the XML-RPC server in the Xen-API and as such might be faster than tunneling the connection through LibVirt Perl bindings, LibVirt itself and then connecting to the Xen-API. This is not tested.

6.6 Software development

Software development, and as mentioned open source contribution, was a big part of this project. The author wanted a technical programming challenge alongside with continued research in virtual infrastructure management.

Before starting the development some prerequisites needed to be in place:

1. Documentation on how to interact with the Xen-API
2. A final decision on which programming language to use
3. Finding a module that would allow for the selected programming language to interact with the Xen-API

6.6.1 Xen-API documentation

When this project started in January of 2007, Xen 3.0.4 was the latest official build of the open source Xen hypervisor. With it, XenSource had included a

[3]http://libvirt.org/

preview of the upcoming Xen-API which was not complete, nor well documented. The author quickly realized that the API version included in 3.0.4 was not sufficient for this projects needs. As mentioned in section 6.1, the Xen-unstable development repository was selected as install base when the virtual infrastructure was installed and configured. The installed version of Xen-unstable had a more complete API version included, and was continuously updated throughout this projects time frame. Never the less, the documentation was not very good and since the API kept changing as more features was added by the XenSource developers, the interaction between the developed application and the API kept changing as well.

To compensate for somewhat lacking and incorrect documentation, the author partly had to explore the API source code itself, which was written in python, as well as a period of trial and error testing. This testing was done in an interactive python shell which was a good choice for this task. The iPython[4] shell gave responses as the commands were typed in, i.e. much like programming and getting feedback on the program itself while actually writing it. XenSource did provide some python examples which made this process fairly convenient. After some weeks of testing, enough information about the various function calls to the API was documented and the development process could go to the next step.

6.6.2 Programming language selection

The selection of programming language for the development process was based on former experience with serverside scripting. Both Perl and Python were strong candidates, but Perl was chosen in the end. There were pros and cons with either language; XenSource has written their tools using Python, and as such provided simple examples on how to use Python to interact with their API. However, the author had limited experience with Python. Perl, on the other hand, is a familiar scripting language for the author as well as being the langauge used by the MLN developers. In the beginning of the project, it was not clear wether or not the software which was going to be developed would be included in MLN. At a later stage the decision was made to include it, but as this was not certain from the beginning, choosing Perl for this projects development would make an addoption by MLN more comprehendable. However, there was no examples or documentation to guide the way for a Perl implementation of interaction with the Xen-API using XML-RPC.

In the end Perl was chosen to be the programming language of choice.

[4]http://ipython.scipy.org

6.6.3 XML-RPC module selection

After the selection of programming language, the next step was to find a suitable XML-RPC module as this is the selected way of communication with the Xen-API by XenSource. The official XML-RPC webpage[42] was consulted to find available Perl implementations. The webpage listed three alternatives where only two had working links. Randy Ray's XML::RPC::Client module[5] was selected after a trial and error period where both alternatives where tested.

6.7 XenGuard analysis and lab results

As a proof-of-concept virtual infrastructure load-balancing model, Xenguard has proven its point. Experiments on migration of both singel nodes and networks of various sizes has been conducted with satisfying results.

As discussed in section 6.3, MLN was chosen to be the infrastructure management tool used in this project. With its high level configration files, it is easy to configure, deploy and manage virtual machines throughout a virtual infrastructure. It does not necessarily have to be virtual machines connected in a network, but as the name implies (Manage Large Networks), it supports the management of extensive networks of virtual machines as well. In chapter 5, we concider management of both single nodes, for instance a set of web servers, and various size networks, for instance a web server frontend with a database backend. We show that with live migration based on resource consumption statistics is possible, not only for the non-networked virtual machines, but also for the ones that are connected over backend networks using bridge devices. The bridge devices, which acts as virtual switches on the private lan, binds the virtual machines connected to it to the same physical host. If either of the connected virtual machines are migrated to a different location than the rest of the private network, they will experience a loss of connectivity.

With the private network connectivity in mind, XenGuard had to be developed to support detection of these networks and make sure all connected nodes would be migrated to the same place upon a relocation decision. Consider a scenario where a school is building a virtualization lab hosting 20 sets of private networks meant for networking education. MLN can build, deploy and manage all these virtual machines including their internal networking from 1 single configuration file. However, upon XenGuard finding the resource consumption to be too high on one of the physical nodes, it is important that all nodes in each of the private networks is live migrated to the same location and are not split up.

[5]http://search.cpan.org/ rjray/RPC-XML-0.59/

Hence, XenGuard needed a way to tell apart the different virtual machines that are bound together in a private network. To solve this problem, a recursive function was developed to scan through all virtual machines of a MLN project. The function will find all connected virtual hosts and switches by simply checking for network interfaces, and what they are connected to, and build a list of "chunks" (see section 5.3), a chunk meaning all hosts and switches which needs to be on the same physical host in order to maintain connectivity.

The live migration process it self, is triggered by the total resource consumption on a physical server surpassing a defined maximum limit. Upon detection of excessive resource use, XenGuard will build a list of potential live migration targets based on the physical servers with the most resources available. It will then calculate the total memory consumption for each *chunk* of virtual machines, and check the list of potential targets to see if there is enough available memory to live migrate them all.

As far as the author could find, only VMware supports *similar* load balancing functionality through its Distributed Resource Scheduling (DRS). Allthough VMware's product is far more mature and supports a wider variety of configured policies and maintenance scenarios, this work has shown that XenGuard is capable of using live migration as a tool to balance out resource consumption throughout the virtual infrastructure and by this making sure the virtual machines have sufficient resources to maintain proper responsetime.

6.8 Xenguard and High Availability

DRS is not the only impressive feature VMware has introduced to the virtualization market. They also, through their HA add-on (USD 2000 + support for 2 processors)[41], provide High Availability (HA) using a similar solution as the Linux project Heartbeat[6]. Research in this field, based on the Heartbeat project and the Xensource hypervisor, was conducted by E. Braastad in his master thesis of May 2006 entitled *Management of high availability services using virtualization*. Braastads research showed that Xen+Heartbeat could monitor physical servers, and their virtual machines, and perform automatic live migration upon server failure.

This feature could be included in XenGuard by allowing either XenGuard or the management software of choice to update Heartbeats configuration as to the physical location of each virtual machine that is re-located. Braastad[3] has

[6]http://www.linux-ha.org/

already created an addition to Heartbeat itself to perform the live migration. Only additional code to interact with Heartbeat to provoke the live migration would have to be introduced to perform this. By estimate, this should not be substantial additional work as the key parts are already in place due to this project and Braastads work.

6.9 Future work and further development

In Xenguards analysis, only CPU resources was taken into account. Simple services such as for instance webservers without any heavy scripting language such as JSP, ASP or PHP would not require much CPU and as such might benefit more from an analysis of network throughput on the physical server and its virtual machines all together in order to find the most appropriate *service_host* in the infrastructure.

The ideal approach for Xenguard would be to concider all of the following factors before making a relocation decision for any virtual machine:

- Local: Total CPU utilization by all other virtual machines, including the control domain - Identify where most resources are spent, and try to be smart about even load distribution throughout the virtual infrastructure
- Local: CPU utilization over time by the virtual machine in question - Identify "peaks"; Is high resource consumption at the moment caused by bursty use of a particular service? Can we find trends and take preemptive action?
- Local: Magnitude of I/O operations - Live migration is based on the filesystem being stored on a storage network. All I/O will result in additional network traffic which again might have an impact on network services provided by other virtual machines communicating over the same network segment
- Local: Network traffic - Does the virtual machine provide a service that results in high network load?
- Local: Consumed memory - How much memory do we need?
- Remote: Available CPU resources over time - Was a potential target host added to the list because of less CPU utilization caused by bursty usage? Trend analysis and load averages could prove beneficial.
- Remote: Available memory - Free memory has to be available on the target host

- Remote: Network traffic - How much throughput does the target host have? Will the added network utilization of the added virtual machine have any impact?
- Remote: Magnitude of I/O operations - Again, the added network traffic from a migrated domain might influence the other virtual machines already communicating over the same network segment
- General: Type of service provided by the virtual machine - Type of server, or service, could be taken into account when load balancing of the virtual infrastructure is being done. Plausible resource consumption statistics could be matched so that virtual machines running on the same physical host utilized different aspects of the available resources.
- General: Uptime priority - When live migration to balance load is going to take place, which virtual machine should be moved first? If a choice has to be made, which server should be moved and which should be shut down? There is also a risk involved with migration, maybe a certain virtual machine should be kept alive by any means possible. This virtual machine should stay while others are migrated when load balancing has to be done in order to preserve responsiveness.

6.10 Conclusion

This project has shown that open source software has come a long way in creating highly conciderable, free of charge competition to commercial long lived projects to manage and control a virtual infrastructure. XeniNFO and XenGuard have introduced features in MLN that are, as far as the author can tell, only available in high priced commercial products such as VMware and XenEnterprise. MLN, as shown through experiments, is highly helpful in managing the virtual infrastructure in a non- time consuming, intuitive and scalable way by its high level configuration and use of preconfigured templates.

This project went through technical difficulties such as exploring the newly introduced XenAPI, in constant development, without completed documentation and the challenge of finding a suitable remote storage solution that would allow for live migration of virtual machines. GNBD was given up after much compiling hassle and AoE was introduced to replace its predecessor from last years work.[3]

The open source principle was a strong motivator throughout the project work as well as the lacking of educational licenses for VMware promptly stopped any development towards management of their virtual infrastructure solution. XeniNFO was developed as a resource consumption monitoring tool using the new API introduced in the latest Xen opensource hypervisor release. The development of this tool resulted in it being included as an API interaction example for the Perl programming language in the Xen-Unstable branch being publicly available under the creative commons license. XenGuard, which holds the logic of the distributed resource management for the virtual infrastructure, was merged with MLNs source to be included in a later official release of MLN supporting dynamic CPU resource sharing throughout the total infrastructure.

The interaction itself with the Xen API was programmed directly and currently suffers the inability to support other virtual infrastructure solutions. However, the programmed logic in XenGuard holds the same value for other solutions and only interaction with other product's API needs to be included in order to utilize this in resource management for for instance KVM or VMware virtual machines. The LibVirt[43] library could be implemented in MLN in the future to for instance control resource distribution for KVM hosts as a solution to this.

As described in section 6.6, the Perl programming language was chosen for both tools developed during this project work. It could very well have been developed in Python as XenSource presented programming examples for this language, but the merging with MLN would have been difficult if Python had been chosen instead. Never the less, Perl is an equally flexible language and was fully fit to take on this programming challenge.

The XenGuard logic, limited by the duration of the project, was meant to be a proof-of-concept that an open source tool could perform such advanced features as VMware has included in their virtual infrastructure enterprise solution. XenGuard, at present, only conciders CPU usage upon deciding of optimal resource distribution in the virtual infrastructure, but as stated, it was meant as a proof-of-concept model. However, experiments have shown that CPU based load balancing of a virtual infrastructure is possible using live migration of its hosted virtual machines. Experiments have been conducted with stand-alone nodes as well as with large private networks including 2 switches, 3 subnets and 7 virtual machines. In order to maintain connectivity, all these virtual machines and their interconnected virtual switches need to reside on the same physical server. Upon a live migration decision by XenGuard, this connectivity dependency is detected using a recursive algorithm checking all virtual machines for connected network devices. If a network device is found, it will continue to scan all connected host on this network device for connected networking devices. The algorithm will recurse into the network finding all connected items and store this information in a list. This list is then consulted upon live migration to make sure all interconnected virtual machines are live migrated to the same physical location so they can maintain network connectivity.

This is a unique feature of MLN-XenGuard as far as the author could find. By detecting network connectivity dependencies upon live migration, we allow for the same flexibility in management of virtual private networks as seen with stand-alone virtual machines.

Can we trust the uptime of virtual machines with an added layer of software which might include bugs, maintenance downtime and additional management? The answer to this has got to be yes for anyone to concider changing to a virtual infrastructure. The advantages are numerous and ever growing towards features and capabilities much beyond what can be done with physical servers. High availability is in demand as internet size, and growth, is climbing. Research on high availability with virtual machines has been conducted[3], and has proven satisfactory results. This project somewhat builds on Braastads work[3] on high availability services using virtual machine and with some additional integration with heartbeat, MLN-XenGuard could easily adapt support for high availability using live migration. The missing piece of code is the interaction with heartbeat itself, but the author concludes that this should be a minimal effort to implement in the future.

The future looks bright for system administrators, and other interested parties, with the option to convert to a virtual infrastructure. At present, VMware is still ahead on well developed, advanced features such as dynamic resource allocation and distribution across the virtual infrastructure, but this project has proven that an open source competing solution could emerge as an official, competitive product at a reasonably soon point in time.

During the mere five months this project has been going, satisfactory results have been provided through experiments showing that a dynamic resource distribution in a virtual infrastructure can be achieved with open source tools. Two applications, XeniNFO and XenGuard, for resource consumption monitoring and load balancing of the virtual infrastructure using live migration, have been developed. XeniNFO, the resource statistics collection part, has been contributed to XenSource and is publicly available from the Xen-Unstable repository while XenGuard has been merged with the open source virtual machine management tool MLN and will be publicly available in a future release of MLN.[37] Having a solid management solution for the virtual infrastructure is a must when resource and configuration management, let alone provisioning and monitoring, of virtual machines grows in proportion to an expanding infrastructure.

List of Figures

List of Tables

Appendix A

Appendix

A.1 Source Code

The source code of the two application this project produced was found to be too extensive to be included here. Visit the urls mentioned in the following sections to download either application.

A.1.1 XeniNFO

The source code of XeniNFO can be found online on either of the following web addresses:

- http://xenbits.xensource.com/xen-unstable.hg?rev/6145e5508d6b
- http://folk.uio.no/ingardm/thesis/xensource.xeninfo.pl

A.1.2 MLN Xenguard

The source code of MLN-XenGuard (beta) can be found online on the following web address:

- http://folk.uio.no/ingardm/thesis/mln-xenguard

A.2 Configuration files

A.2.1 MLN

A.2.1.1 Project Simple

```
global {
        $sh = 192.168.0.10
        project simple
}
switch {
        lan {
                service_host $sh
                xen 1
                bridge lan.simple
                hosts {
                        away eth0
                        fly eth0
                }
        }
}
superclass {
        hosts {
                lvm
                xen
                service_host $sh
                nameserver 192.168.0.100
                network eth0 {
                        switch lan
                        netmask 255.255.255.0
                }
        }
}
host away {
        superclass hosts
        network eth0 {
                address 10.0.0.2
                gateway 10.0.0.1
        }
}
host fly {
        superclass hosts
        network eth0 {
                address 10.0.0.1
        }
        network eth1 {
                address dhcp
        }
        startup {
                iptables -t nat -A POSTROUTING -o eth1 -j MASQUERADE
                echo 1 > /proc/sys/net/ipv4/ip_forward
        }
}
```

A.2.1.2 Project ftX

```
global {
        project ft3
}
switch {
        lan12 {
                service_host 192.168.0.10
                xen 1
                hosts {
                        north eth0
                        south eth0
                        easth eth0
                        west eth0
                }
        }
        lan11 {
                service_host 192.168.0.10
                xen 1
                hosts {
                        away eth0
                        home eth0
                        fly eth0
                        north eth1
                }
        }
        lan21 {
                service_host 192.168.0.10
                xen 1
                hosts {
                        h1 eth0
                        h2 eth0
                        h3 eth0
                }
        }
}
superclass {
        hosts {
                lvm
                xen
                nameserver 192.168.0.100
                term screen
                free_space 100M
                network eth0 {
                        netmask 255.255.255.0
                }
        }
}
host {
        h1 {
                memory 64M
                service_host 192.168.0.10
                superclass hosts
                network {
                        eth1 {
                                address dhcp
                        }
                        eth0 {
                                switch lan21
                                address 10.1.0.1
                        }
                }
                startup {
                        iptables -t nat -A POSTROUTING -o eth1 -j MASQUERADE
                        echo 1 > /proc/sys/net/ipv4/ip_forward
                }
        }
        h2 {
                memory 64M
                service_host 192.168.0.10
                superclass hosts
                network {
                        eth0 {
                                switch lan21
                                address 10.1.0.2
                                gateway 10.1.0.1
                        }
                }
        }
        h3 {
                memory 64M
                service_host 192.168.0.10
                superclass hosts
                network {
                        eth0 {
                                switch lan21
                                address 10.1.0.3
                                gateway 10.1.0.1
                        }
                }
        }
```

```
fly {
        service_host 192.168.0.10
        superclass hosts
        network {
                eth1 {
                        address dhcp
                }
                eth0 {
                        switch lan11
                        address 10.0.0.1
                }
        }
        startup {
                iptables -t nat -A POSTROUTING -o eth1 -j MASQUERADE
                echo 1 > /proc/sys/net/ipv4/ip_forward
        }
}
home {
        service_host 192.168.0.10
        superclass hosts
        network {
                eth0 {
                        switch lan11
                        address 10.0.0.3
                        gateway 10.0.0.1
                }
        }
}
away {
        service_host 192.168.0.10
        superclass hosts
        network {
                eth0 {
                        switch lan11
                        address 10.0.0.2
                        gateway 10.0.0.1
                }
        }
}
north {
        service_host 192.168.0.10
        superclass hosts
        network {
                eth1 {
                        switch lan11
                        address 10.0.0.10
                        netmask 255.255.255.0
                        gateway 10.0.0.1
                }
                eth0 {
                        switch lan12
                        address 10.0.2.1
                }
        }
        startup {
                iptables -t nat -A POSTROUTING -o eth1 -j MASQUERADE
                echo 1 > /proc/sys/net/ipv4/ip_forward
        }
}
east {
        service_host 192.168.0.10
        superclass hosts
        network {
                eth0 {
                        switch lan12
                        address 10.0.2.2
                        gateway 10.0.2.1
                }
        }
}
south {
        service_host 192.168.0.10
        superclass hosts
        network {
                eth0 {
                        switch lan12
                        address 10.0.2.3
                        gateway 10.0.2.1
                }
        }
}
west {
        service_host 192.168.0.10
        superclass hosts
        network {
                eth0 {
                        switch lan12
                        address 10.0.2.4
                        gateway 10.0.2.1
                }
```

```
                    }
          }
}
```

Bibliography

[1] http://mln.sourceforge.net. Accessed April 2007.

[2] Begnum K. Manage large networks of virtual machines. In *Proceedings of the Twentieth Systems Administration Conference (LISA 2006)*, page 101, 2006.

[3] Espen Braastad. Management of high availability services using virtualization. Master's thesis, Oslo University Collage, 2006.

[4] Begnum K. and Disney M. Scalable deployment and configuration of high-performance virtual clusters. In *CISE/CGCS 2006: 3rd International Conference on Cluster and Grid Computing Systems*, 2006.

[5] R. P. Goldberg. Survey of virtual machine research. *IEEE Computer Magazine*, 7(6):34–45, July 1974.

[6] Rosenblum M. and Garfinkel T. Virtual machine monitors: current technology and future trends. *Computer*, 38(5):39–47, May 2005.

[7] http://en.wikipedia.org/wiki/virtualization. Accessed April 2007.

[8] R. P. Goldberg. Architecture of virtual machines. In *Proceedings of the workshop on virtual computer systems*, pages 74–112, New York, NY, USA, 1973. ACM Press.

[9] Popek J. Gerald and Goldberg P. Robert. Formal requirements for virtualizable third generation architectures. *Communications of the ACM*, 17(7):412–421, July 1974.

[10] J. S. Robin and C. E. Irvine. Analysis of the intel pentium's ability to support a secure virtual machine monitor. In *In Proceedings of the 9th USENIX Security Symposium*, pages 129–144, August 2000.

[11] R. J. Creasy. The origin of the vm/370 time-sharing system. *IBM Journal of Research and Development*, pages 483–491, September 1981.

[12] R. A. MacKinnon L. H. Seawright. Vm/370 - a study of multiplicity and usefulness. *IBM Systems Journal*, pages 4–17, January 1979.

[13] http://en.wikipedia.org/wiki/vm_(operating_system). Accessed April 2007.

[14] http://en.wikipedia.org/wiki/x86_virtualization. Accessed April 2007.

[15] Andrew Whitaker, Marianne Shaw, and Steven D. Gribble. Scale and performance in the denali isolation kernel. *SIGOPS Oper. Syst. Rev.*, 36(SI):195–209, 2002.

[16] http://en.wikipedia.org/wiki/paravirtualization. Accessed April 2007.

[17] Paul Barham, Boris Dragovic, Keir Fraser, Steven Hand, Tim Harris, Alex Ho, Rolf Neugebauer, Ian Pratt, and Andrew Warfield. Xen and the art of virtualization. In *SOSP '03: Proceedings of the nineteenth ACM symposium on Operating systems principles*, pages 164–177. ACM Press, 2003.

[18] Nadir Kiyanclar. A survey of virtualization techniques focusing on secure on-demand cluster computing, November 2005.

[19] http://www.vmware.com. Accessed April 2007.

[20] Jeremy Sugerman, Ganesh Venkitachalam, and Beng-Hong Lim. Virtualizing i/o devices on vmware workstation's hosted virtual machine monitor. In *Proceedings of the General Track: 2002 USENIX Annual Technical Conference*, pages 1–14, Berkeley, CA, USA, 2001. USENIX Association.

[21] Carl A. Waldspurger. Memory resource management in vmware esx server. *SIGOPS Oper. Syst. Rev.*, 36(SI):181–194, 2002.

[22] Greg A. Koenig Nadir Kiyanclar and William Yurcik. Maestro-vc: On-demand secure cluster computing using virtualization. In *7th LCI International Conference on Linux Clusters*, 2006.

[23] http://en.wikipedia.org/wiki/system/370. Accessed April 2007.

[24] Beng-Hong Lim Michael Nelson and Inc. Greg Hutchins, VMware. Ibm virtual machine facility/370: Introduction. ibm systems reference library. December 1977.

[25] http://en.wikipedia.org/wiki/system/390. Accessed April 2007.

[26] http://en.wikipedia.org/wiki/z/architecture. Accessed April 2007.

[27] http://en.wikipedia.org/wiki/zseries. Accessed April 2007.

[28] Intel. Vanderpool technology. http://www.intel.com/technology/computing/vptech. Accessed April 2006.

[29] David A. Patterson. A simple way to estimate the cost of downtime. In *LISA '02: Proceedings of the sixteenth Systems Administration Conference*, pages 185–188. USENIX Association, 2002.

[30] David E. Lowell, Yasushi Saito, and Eileen J. Samberg. Devirtualizable virtual machines enabling general, single-node, online maintenance. In *ASPLOS-XI: Proceedings of the 11th international conference on Architectural support for programming languages and operating systems*, pages 211–223, New York, NY, USA, 2004. ACM Press.

[31] S. Hand J. G. Hansen E. Jul C. Limpach I. Pratt C. Clark, K. Fraser and A. Warfield. Live migration of virtual machines. In *NSDI'05: In Proceedings of the 2nd ACM/USENIX Symposium on Networked Systems Design and Implementation*, 2005.

[32] Beng-Hong Lim Michael Nelson and Inc. Greg Hutchins, VMware. Fast transparent migration for virtual machines. In *USENIX Annual Technical Conference 2005*, pages 391–394.

[33] Aravind Menon, Jose Renato Santos, Yoshio Turner, G. (John) Janakiraman, and Willy Zwaenepoel. Diagnosing performance overheads in the xen virtual machine environment. In *VEE '05: Proceedings of the 1st ACM/USENIX international conference on Virtual execution environments*, pages 13–23. ACM Press, 2005.

[34] VMware Technology Network. A performance comparison of hypervisors. Technical report, January 2007.

[35] Tom Bittman and Donna Scott. Gartner symposium presentation - the new infrastructure: Real time, virtual and connected. Nov-Dec 2004.

[36] http://www.drbd.org/. Accessed May 2007.

[37] M. Disney E. Frisch K. Begnum, I. Mevaag. Towards a policy for virtual machine management. In *Submitted to the 21st Systems Administration Conference (LISA 2007)*, 2007.

[38] Live migration of xen domains
http://www.linux.com/article.pl?sid=06/07/17/1916214. Accessed February 2007.

[39] Rfc 3720 - internet small computer systems interface
http://www.faqs.org/rfcs/rfc3720.html. Accessed May 2007.

[40] http://www.xensource.com/products/xen_enterprise/index.html. Accessed May 2007.

[41] http://www.vmware.com/vmwarestore/buyvi3.html. Accessed May 2007.

[42] http://www.xmlrpc.com/. Accessed February 2007.

[43] http://www.libvirt.org/. Accessed May 2007.